JTRON 2.1 仕様

Ver.2.01.00

社団法人 トロン協会 ITRON 部会

坂村 健 監修／JTRON 仕様 WG 編

JTRON2.1 仕様(Ver2.01.00)

本仕様ならびに本仕様書に関する問い合わせ先は、下記の通りです。

(社)トロン協会

〒108-0073 東京都港区三田 1 丁目 3 番 39 号勝田ビル 5 階

TEL:03-3454-3191　FAX:03-3454-3224

---

§TRON は“ The Real-time Operating system Nucleus ”の略称です。
§ITRON は“ Industrial TRON ”の略称です。
§JTRON は“ Java Technology on ITRON ”の略称です。
§μITRON は“ Micro Industrial TRON ”の略称です。
§BTRON は“ Business TRON ”の略称です。
§CTRON は“ Central and Communication TRON ”の略称です。
§TRON, ITRON, JTRON, μITRON, BTRON, および CTRON は、特定の商品ないしは製品群を指す名称ではありません。

監修の言葉

トロンプロジェクトのルーツは、リアルタイム、組込み制御用の OS 仕様である ITRON である。ITRON サブプロジェクトが正式に始まったのは 1984 年であるが、リアルタイム OS に対する要求仕様の調査は、さらにその 2〜3 年前から始まっていた。当時のマイクロコンピュータは、誕生から約 10 年が経過し、電子レンジやエアコン等の家電製品への採用が増えてはいたものの、当時はまだ「マイコン内蔵」「マイコン制御」がそのまま宣伝文句になるような、裏返すとまだマイコン制御が珍しい時代であったと言える。

当時の制御用マイコンの大部分は OS とは無縁であり、OS やドライバの階層に相当する部分も、すべてアプリケーションプログラムの一部として一体に開発していた。これは、CPU やメモリ資源が乏しく、OS を載せる余裕が無かったということもあるが、適当な OS が存在しなかったという理由も大きい。リアルタイム OS の技術そのものはそれ以前から存在していたが、軍事関連など大きめのシステムを対象としたものであり、家電や OA 機器向けのものではなかった。

しかし、家電や OA 機器の機能が複雑化する一方で、制御用ソフトウェアの開発コストが無視できなくなってくると、ソフトウェアの生産性や保守性、拡張性を向上する目的で、OS を導入するメリットが出てくる。
こういった時代背景から、特に家電など低コストのシステムにも採用しやすいコンパクト性を重視して設計されたリアルタイム制御用 OS が ITRON である。

ITRON は、最初の製品が開発された後も継続してバージョンアップを行い、その後 15 年近くに渡って 100 種類以上の実装が行われた。
これだけ多数の実装例をもつ OS 仕様は、ほかに存在しない。
特に昨今では、普及のめざましい携帯電話、デジタルカメラやビデオカメラなどの高機能 AV 機器、自動車のエンジン制御や車内情報システムなど、幅広い情報関連機器が ITRON によって制御されており、まさに IT 革命の影の主役となっている。

ITRON の特長であるコンパクト性は、他の技術や他のシステムと組み合わせて利用しやすいというメリットにも通じる。

その例の 1 つが、Java との組み合わせである。
ご承知のように、Java はプログラムのポータビリティに優れ、その上の各種ミドルウェア群(Java 流に言えばクラスライブラリ)も充実しているのが特長である。一方、Java の欠点としては、実行環境が重くてリアルタイム制御はできないし、優れたポータビリティ裏返しとしてハードウェアを直接操作することは不得意である。しかし、これらの欠点は ITRON の利用によって補うことができる。
すなわち、Java と ITRON を融合させた実行環境によって、プログラムのポータビリティにもリアルタイム制御にも配慮したシステムを構築することができる。
これが、1997 年の末に発表された JTRON 仕様 OS である。

上記の経緯から分かるように、JTRON は ITRON から枝分かれしたサブプロジェクトであるが、ITRON のカーネル単体と比較して、より高機能で複雑なシステム(ビデオカメラ等)に向いている。
ITRON の場合と同様、エンドユーザが JTRON という OS を意識することはまずないが、JTRON の採用により制御用ソフトウェアの開発期間が大幅に短縮され、直接には開発メーカーに対して、間接的にはエンドユーザに対しても大きなメリットを生じている。

さて、本書は JTRON 仕様の最新の仕様書である。
1 つ前の版である JTRON 2.0 仕様との見かけの差異は小さいかもしれないが、この間に ITRON カーネルの最新仕様である μITRON 4.0 仕様が確定し、JTRON 2.0 仕様から JTRON 2.1 仕様へのバージョンアップにあたってはその最新成果を取り入れた。すなわち、本書で説明している JTRON 2.1 仕様は、20 年近くにわたって ITRON の上に蓄積されてきたリアルタイム制御のノウハウと、コンピュータ界の世界共通言語とも言うべき Java の技術とを融合した、世界的にも極めてユニークな成果の集大成である。

本書が、高機能リアルタイムシステムの開発を志す多くの技術者にとって、幅広く役立つものとなることを願っている。

2000 年 11 月
坂村 健

まえがき

当初JTRON仕様は、トロンプロジェクトにおいて仕様が検討され1997年に仕様が公開された。これがJTRON1.0仕様である。これを補完する形でITRON部会でリアルタイムOS（RTOS）とJava実行環境の連携を強化したJTRON2.0仕様の策定を行い1998年秋に仕様を公開した。JTRON2.0仕様策定時にJTRON1.0で規定されている内容の盛り込みを考えた。しかしこの時点ではそのもととなるμITRON仕様が4.0版を作成中であり、断念せざるをえなかった。μITRON4.0仕様は1999年6月に仕様が公開された。これに伴い、ITRON部会JTRON仕様WGではJTRON仕様の改定を開始し、今ここにJTRON2.1仕様を公開するに至った。

この間、(株)アプリックス、日本電気(株)、東芝情報システム(株)により、JTRON1.0仕様ないしJTRON2.0仕様の実装、あるいは製品化が行なわれた。これらの実装から多くの貴重なフィードバックがこの仕様にもたらされた。また、J Consortium事務局のWendy Fong氏からはJTRON2.0仕様の英語版仕様書に関して貴重なコメントを頂いた。改めて感謝の意を表す次第である。

今回の仕様改定で大きな点を以下に示す。
JTRON1.0からは、ベースとなるリアルタイムOSがμITRON4.0になり、仕様もこれに合わせて変更された。
JTRON2.0からは、Javaとネイティブ実行環境との新しいインタフェース仕様(JNI)に対応させたこと、実装方法を規定する仕様を削除したことである。

注意

・本仕様書では「Java 技術，もしくは Java 言語」を「Java」と省略している場合があります。
・本書の著作権は、社団法人トロン協会に属しています。
・本書の内容の転載、一部複製などには、トロン協会の許諾が必要です。
・本仕様書に記載されている内容は、今後の改良などの理由で断り無しに変更することがあります。
・仕様に関しては、以下にお問い合わせください。

社団法人トロン協会
〒108-0073 東京都港区三田 1 丁目 3 番 39 号勝田ビル 5 階

備考

・Java およびすべての Java 関連の商標およびロゴは、米国およびその他の国における米国 Sun Microsystems, Inc.の商標または登録商標です。
・Sun，Sun Microsystems は、米国およびその他の国における米国 Sun Microsystems, Inc.の商標または登録商標です。
・TRON は“ The Real-time Operating system Nucleus ”の略称です。
・JTRON は、“ Java Technology on ITRON ”の略称です。
・ITRON は“ Industrial TRON ”の略称です。
・μITRON は“ Micro Industrial TRON ”の略称です。

仕様書の構成

本仕様書は、JTRON2.1 仕様の C 言語 API および、Java 言語 API を規定するものである。仕様書のバージョン番号は、仕様書の表紙ならびに各ページの右上に表示されている。
本仕様書の構成は次の通りである。
第 1 章では、JTRON の概要と設計方針について解説する。
第 2 章では、JTRON 仕様における各種の概念と、複数のインタフェースに共通する定義を示す。
第 3〜6 章では、JTRON 仕様の各インタフェースをインタフェース毎に解説する。
第 7 章では、仕様書を読む上で参考となる一覧表などを掲載する。
仕様の利用条件や保守体制など仕様に関連する情報は巻末の付録にまとめられている。また、参考情報として実装例を 5 章と 6 章の章末に記述している。これらは、JTRON2.1 仕様書の本体ではない。

仕様書の記述形式

本仕様書では、次の記述形式を採用している。用語についてはμITRON4.0 仕様に準ずる。

【補足説明】

この項は、JTRON2.1 仕様の本体だけではわかりにくい点や誤解するおそれのある点について説明を補足するもので、JTRON2.1 仕様の本体ではない。

【JTRON1.0 仕様との相違】

この項では、JTRON1.0 仕様との主な相違と、変更の理由について解説する。この項の内容は、JTRON2.1 仕様の本体ではない。

【JTRON2.0 仕様との相違】

この項では、JTRON2.0 仕様との主な相違と、変更の理由について解説する。この項の内容は、JTRON2.1 仕様の本体ではない。

【仕様決定の理由】

この項では、仕様の決定理由として特に説明を必要とする点について説明する。この項の内容は、JTRON2.1 仕様の本体ではない。

【JTRON1.0 仕様との相違および仕様決定の理由】

この項では、JTRON1.0 仕様との主な相違と、変更の理由および仕様の決定理由として特に説明を必要とする点について説明する。この項の内容は、JTRON2.1 仕様の本体ではない。
また、第 3 章から第 6 章の ITRON API と Java クラスの機能説明では、上記に加えて次の記述形式を用いる。
ITRON API の機能説明は、次のヘッダで開始する。

ITRON API の名称　　仕様レベル

ITRON API の説明

「ITRON API の名称」には、サービスコールまたは静的 API の名称を記述する。「ITRON API の説明」では、サービスコールまたは静的 API

の機能を簡潔に説明する。「仕様レベル」には、標準仕様と拡張仕様の区別を記述する。標準仕様は、各タイプ毎に必要最低限実装しなければならない仕様である。但し、タイプ 1 の機能で下位のμITRON 仕様 OS にない機能は提供されなくてもよい。一方、拡張仕様は必ずしも必要でない機能であり、実装定義である。

ITRON API の機能説明では、次の記述形式を用いる。

【静的 API】

この項では、静的 API のシステムコンフィギュレーションファイル中での記述形式を示す。

【C 言語 API】

この項では、サービスコールの C 言語からの呼出し形式を示す。

【パラメータ】

この項では、サービスコールおよび静的 API に渡すパラメータをリストアップし、それぞれのパラメータのデータ型と名称、簡単な説明を示す。

【リターンパラメータ】

この項では、サービスコールが返すリターンパラメータをリストアップし、それぞれのリターンパラメータのデータ型と名称、簡単な説明を示す。

【エラーコード】

この項では、サービスコールが返すメインエラーコードをリストアップし、それぞれのエラーコードを返す理由を簡単に説明する。ただし、多くのサービスコールが共通の理由で返すメインエラーコードについては、サービスコール毎には記述しない(μITRON4.0 仕様書参照)。

【機能】

この項では、サービスコールおよび静的 API の機能について説明する。サービスコールや定数などの名称で、イタリック体で記述した文字は、他の文字に置き換えられるものであることを示す。例えば、cre_*yyy* (*yyy* がイタリック体)は、cre_tsk, cre_sem, cre_flg などを表す。
オブジェクト属性やサービスコールの動作モードなどのパラメータで、いくつかの値を選択して設定する場合には、次のような記述方法を用い

る。

[x] x を指定しても指定しなくてもよいことを示す。

x | y x と y をビット毎に論理和をとって指定することを示す。

x || y x または y のいずれか片方を指定できることを示す。

例えば、((TA_HLNG || TA_ASM) | [TA_ACT]) は、以下の 4 種類のいずれかの指定ができることを示す。

```
TA_HLNG
TA_ASM
(TA_HLNG | TA_ACT)
(TA_ASM | TA_ACT)
```

Java クラスの機能説明は、次のヘッダで開始する。

Java クラスの名称 仕様レベル

Java クラスの説明

「Java クラス階層図」には、クラスの機能説明の場合のみ階層図を記述する。

インタフェースの場合は階層図はない。「Java クラスの名称」には、クラスまたはインタフェースの名称を記述する。「Java クラスの説明」では、クラスまたはインタフェースの機能を簡潔に説明する。「仕様レベル」には、標準仕様と拡張仕様の区別を記述する。標準仕様は、各タイプ毎に必要最低限実装しなければならない仕様である。但し、タイプ 1 の機能で下位の μITRON 仕様 OS にない機能は提供されなくてもよい。一方、拡張仕様は必ずしも必要でない機能であり、実装定義である。

Java クラスの機能説明では、次の記述形式を用いる。

■ クラス定義

この項では、クラス定義の記述形式を示す。

■ 定数

この項では、クラスで定義される定数を示す。

■ 変数

この項では、クラスで定義される変数を示す。

■ コンストラクタ

この項では、クラスで定義される各コンストラクタについて、それぞれのパラメータ、戻り値、機能などを説明する。

■static メソッド

この項では､クラスで定義される static メソッド(またはクラスメソッドと呼ばれる)について、それぞれのパラメータ、戻り値、機能などを説明する。

■ メソッド

この項では、クラスで定義されるメソッドについて、それぞれのパラメータ、戻り値、機能などを説明する。

【パラメータ】

この項では、コンストラクタまたはメソッドに渡すパラメータをリストアップし、それぞれのパラメータのデータ型と名称、簡単な説明を示す。

【戻り値】

この項では、メソッドの戻り値のデータ型と名称、簡単な説明を示す。

【例外】

この項では、コンストラクタまたはメソッドで発生する例外をリストアップし、その例外のクラス名と簡単な説明を示す。

【機能】

この項では、コンストラクタまたはメソッドの機能について説明する。

目次

参考文献

[1]トロンプロジェクト，JTRON 仕様書，Dec. 1997.
[2]社団法人トロン協会，JTRON2.0 仕様書，1998 年 9 月 25 日 Ver2.00.00 Final，Java Technology on ITRON-specification OS 技術委員会
[3]μITRON4.0 仕様 Ver.4.00.00 社団法人トロン協会，ITRON 部会，坂村 健 監修／高田 広章 編
[4] Sheng Liang, The Java Native Interface: Programmer's Guide and Specification, Addison Wesley, 1999.
[5] J.Gosling, B. Joy and G. Steele, The Java Language Specification, Addison-Wesley, 1996.
[6] J.Gosling, B. Joy, G. Steele, and G. Bracha,The Java Language Specification,Second Edition, Addison-Wesley, 2000.

第１章 JTRON仕様の概要

Javaは組込みシステムのキーテクノロジーの一つとなりつつあり、幅広い組込みシステムのアプリケーションに適用可能と期待されている。Javaには以下のような組込みシステムにも有効な特徴がある。

* オブジェクト指向言語：Javaは強い型チェック機能をもった単純なオブジェクト指向言語である。クラスライブラリを利用してアプリケーションプログラムの生産性を向上させることができると期待される。Human Machine Interface が必要とされる組込みシステム、例えば、セットトップボックス、カーナビゲーションシステムなどでは、Javaのクラスライブラリを使って、GUI機能を容易に作成することができる。
* プラットフォームフリー：Javaプログラムは CPUとOSから独立の仮想マシン上で動作する。組込みソフトウェアの開発現場でも、開発用マシンとターゲットマシンの実行環境が同じになるので、開発用マシンで作成したJavaのプログラムを再コンパイルなしにターゲットマシンで実行させることができ、生産性が向上する。また、例えば、セットトップボックスがJavaを搭載している場合、サービス供給者から見ればJavaという単一の言語でインターネットサービスを始めとするサービスを提供することが可能となる。
* 頑強で高安全性：Javaはメモリポインタの機能がない。従ってC/C++言語で頻繁に発生したポインタ誤りによるプログラムミスがなくなる。また、動的なメモリの管理に関してはプログラマにより獲得、解放をさせるのではなく、システム側で使用しなくなったメモリを自動的に解放するので、メモリの解放忘れによるシステムダウンがなくなる。

上記のような特徴があるにもかかわらず、Javaを組込みシステムに適用するにあたって様々な問題があるとされている。例えば、

* インタプリタによる実行速度が遅い。
* メモリのガーベジコレクション、プログラムの動的リンクなど実行時間が予測不可能になる要素がある。
* Javaのセマンティクス、特にスケジューリングポリシなどがリアルタイムシステムに対して曖昧である。

などがあげられる。

こうした問題を解決し、Javaを組込みシステムに適用するアプローチには大きく次の２つのアプローチがあると考えられる。

* リアルタイム拡張アプローチ：Javaの言語仕様やクラスライブラリをリ

アルタイムシステム向けに拡張するものである。プログラム全体をJavaで記述しようとするものである。
* ハイブリッドアプローチ：組込みソフトウェアをリアルタイム処理、非リアルタイム処理に機能分割し、GUI部分、付加価値部分などの非リアルタイム処理をJava言語で記述し、 Java実行環境で実行する。リアルタイム処理は既存のプログラミング言語(C,C++)で記述し、リアルタイムOSで実行しようとするものである。このアプローチでは、Javaスレッドとリアルタイムタスクは協調動作することになる。

JTRONはハイブリッドアプローチをとるシステムにおいて、Javaスレッドとリアルタイムタスクの協調動作を支援する機能、すなわち、Javaスレッドからリアルタイムカーネルを呼び出すためのインタフェースとリアルタイムタスク間の通信インタフェースを標準化したものである。上述したように、ハイブリッドアプローチを取る場合、Javaスレッドからリアルタイムカーネルのサービスコールを呼び出すなど、リアルタイムタスクとJavaスレッド間でデータのやりとりをする必要がある。。また、Javaスレッドとリアルタイムタスクが協調動作する場合に、この両者の間の一種の通信プロトコルと関連したAPI(Application Program Interface)が必要になってくる。リアルタイムタスクカーネルの呼び出しのインタフェースやJavaスレッドとリアルタイムタスク間の通信インタフェースを標準化しておけば、Javaを組込みシステムに適用する場合に標準に従った設計をすればよく設計効率が上がること、Javaプログラム、リアルタイムプログラムの双方のポータビリティが向上し、ひいてはプログラムの流通が促進されることなどのメリットがある。

本仕様書では以下の仕様を規定している。

1. リアルタイムタスクとJavaスレッド間の関係の定義
   Javaスレッドをリアルタイムタスクの1対1でマッピングするものとする。このマッピング方法を規定する。
2. Javaプログラムとリアルタイムタスク間の協調動作のためのモデルとそのためのAPIを規定する。これには以下の3種類がある。
   (1) タイプ1(アタッチクラス)：Javaプログラムからリアルタイム OSのサービスコールが利用できるようにする。
   (2) タイプ2(共有オブジェクト)：Javaプログラムとリアルタイムタスクが共有オブジェクトにより通信する。
   (3) タイプ3(ストリーム通信)：リアルタイムタスクとJavaプログラムがストリームを使って通信する。

本仕様書では、リアルタイムOSとしてITRON仕様OSを想定しているが、他のリアルタイムOSでも多くの部分は適用可能とする。

タイプ1ではJavaスレッドはアタッチクラスと呼ばれるJavaのクラスを通じて、リアルタイムOSの資源にアクセスすることができる。アタッチクラスとそのインスタンスは、リアルタイムOSの資源の型と資源の実体に対応する。アタッチクラスのメソッドはリアルタイムOSの資源に対するAPIに対応する。例えば、セマフォのアタッチクラスを使って、リアルタイムの世界のセマフォはJavaの世界にアタッチされ、Javaスレッドはリアルタイムタスクとこれを使って同期をとる。さらにメモリ操作クラスを使って、Javaプログラムからリアルタイムタスクのデータをアクセスすることができる。

タイプ2では、Javaスレッドとリアルタイムタスクは Javaの世界から「エキスポート」された共有オブジェクトを介して通信する。タイプ2では、まずJavaスレッドがJavaのオブジェクトを共有オブジェクトとして登録する。リアルタイムタスクはこの共有オブジェクトに対して、オブジェクトのフィールド単位でAPIによりアクセスする。共有オブジェクトを安全にアクセスするための排他制御ができるロック機構を設けた。
さらに、リアルタイムタスクからJavaスレッド、スレッドグループを操作できるようにした。これは共有オブジェクトにアクセスする際に、リアルタイムタスクからJavaスレッドを制御することが必要になると考えたからである。

タイプ3では、Javaスレッドとリアルタイムタスクは Javaのストリームを使って通信することになる。Javaライブラリにおけるソケット APIのポートに似た機構を導入した。Javaプログラム側には、JTRON用ストリームクラスが提供される。これを利用して、Javaスレッドからは通常のストリームアクセスを行うことができる。リアルタイムタスク側にはストリーム機能を実現するAPIが提供される。

第 2 章 JTRON 仕様共通規定

本章では JTRON2.1 仕様全般に渡り使われる規則、用語の説明を説明する。

2.1 全体規則(ITRON カーネル)

2.1.1 命名規則

全般に ITRON の命名規則に準じる。プレフィクスとして JTI(JTron Interface)を付与する。

マクロ名： JTI_*ZZZ*
型名： T_JTI_*XXX*
関数名： jti_*XXX*_*YYY*: *XXX* は操作，*YYY* は操作対象オブジェクト

2.1.2 静的 API と動的 API

オブジェクトの生成や初期設定を行う各 API に対して、システム構成ファイル中にオブジェクトの構成や初期値設定を記述する API を用意する。これを静的 API と呼ぶ。システム構成ファイル中の静的 API はコンフィギュレータなどのツールにより処理され、システム初期化時にオブジェクトと関連情報が設定される。静的 API は、API 名を大文字で記述することで、通常の API(これを動的 API と呼ぶ)と区別する。

2.1.3 API の戻り値とエラーコード

各 API の戻り値は、ITRON 仕様の慣例に従い、エラーが発生した場合には負の値のエラーコード、正常に実行された場合は 0 または正の値とする。正常実行された場合の戻り値の意味は API 毎に定義される。
エラーコードは、下位８ビットをメインエラーコードと、それを除いた上位ビットのサブエラーコードで構成される。メインエラーコード、サブエラーコード共に負の値とする（サブエラーコードの値とは、エラーコードの値を符号付きで８ビット右シフトしたものである）。したがって、それらを組み合わせたエラーコードも負の値となる。
メインエラーコードの名称、意味、値は、ITRON 仕様共通マクロを利用し、ITRON 仕様共通定義で規定する。

ITRON 仕様共通マクロ
メインエラーコード：ER mercd = MERCD(ER ercd)
エラーコードからメインエラーコードを取り出す。
サブエラーコード：ER sercd = SERCD(ER ercd)
エラーコードからサブエラーコードを取り出す。

2.1.4 待ち状態とタイムアウト

ある事象が起こるまでプログラムの実行が中断することをリアルタイムタスクの場合、「待つ」、あるいは「待ち状態に入る」といい、Java スレッドの場合は「ブロックされる」という。待ち状態に入る可能性のある ITRON API には、タイムアウトを用意する。
タイムアウトは、一定時間経過しても処理が完了しない場合に、処理をキャンセルして API からリターンするものである(この時、API からは E_TMOUT エラーが返る)。そのため、タイムアウトが起った場合には、API を呼び出したところでオブジェクト状態は変化していないのが原則である。ただし、API の機能上、処理のキャンセル時にもとに戻せない場合は除く。
ポーリングとはタイムアウト時間を 0 としたタイムアウト処理である。API の中で待ち状態になっている場合、API による処理がペンディングしているという。
本仕様中の API の説明では、タイムアウト指定をした場合には、指定時間経過後に待ち状態が解除され、E_TMOUT を戻り値として API からリターンする。
タイムアウト値は、ITRON カーネル仕様にあわせて、正の値でタイムアウト時間 (単位はミリ秒を推奨)、TMO_POL(=0) でポーリング、TMO_FEVR(=-1)で永久待ちをあらわすこととする。

2.1.5 API とタスクの関係

本仕様の API は、パラメータが同じであれば、どのタスクから呼び出しても同じように機能する。すなわち、本仕様の API によってタスクに割り付けられる資源はない。
本仕様の API の中で待ち状態に入っているタスクに対して rel_wai を発行した場合、API から E_RLWAI エラーが返る。また、同じ状況で ter_tsk を発行した場合の振舞いは実装定義である。

2.2 共通定義

2.2.1 ヘッダファイル

ヘッダファイル名："jti_*XXX*.h"とする。
共通定義で使用されるヘッダファイル： "jti_common.h"
タイプ１で使用されるヘッダファイル： "jti_attach.h"
タイプ２で使用されるヘッダファイル： "jti_shared.h"
タイプ３で使用されるヘッダファイル： "jti_stream.h"

2.2.2 データ構造／データ型

(1)共有オブジェクトインタフェース用

```
JNO             整数型、長さは実装定義
```

(2)ストリームインタフェース用

```
typedef struct t_jti_cstm {
      VP      exinf;  /* 拡張情報 */
      ATR     stmatr; /* ストリーム属性 */
      VP      wbuf;   /* 送信バッファの先頭 */
      INT     wbufsz; /* 送信バッファのサイズ */
      VP      rbuf;   /* 受信バッファの先頭 */
      INT     rbufsz; /* 受信バッファのサイズ */
   (他に実装定義のフィールドがあってもよい)
} T_JTI_CSTM;

typedef struct t_jti_rstm {
      VP      exinf;  /* 拡張情報 */
      INT     wrisz;  /* 待たずに送信可能なデータ長 */
      INT     reasz;  /* 待たずに受信可能なデータ長 */
   (他に実装定義のフィールドがあってもよい)
} T_JTI_RSTM;
```

2.2.3 定数

(1)Java スレッド／リアルタイムタスク優先度対応指定

JTI_DFL_HPR	Java	実行環境を実装するリアルタイムタスクのデフォルト最高優先度値、値は実装定義である。

(2)ストリームインタフェース用

JTI_MAIN_STREAM	1	主ストリームの ID
TA_WRITE	0x01	ストリーム属性。送信を可能にする。
TA_READ	0x02	ストリーム属性。受信を可能にする。

2.3 全体規則(Java)

2.3.1 Java パッケージ構成

2.3.1.1 JTRON 標準パッケージ構成

JTRON 標準 Java パッケージ名は、表 2.1 で示すように定義する。

表 2.1 JTRON 標準 Java パッケージ名

識別	JTRON 標準パッケージ名
共通仕様で使用されるパッケージ	org.jtron
タイプ 1 で使用されるパッケージ	org.jtron.attach
タイプ 2 で使用されるパッケージ	org.jtron.shared
タイプ 3 で使用されるパッケージ	org.jtron.stream

但し、タイプ 1（アタッチクラス）にて使用されるクラス実装は、下層の μITRON 仕様 OS の実装定義部分の定義により変化するため、実装毎に違ったクラス定義となる場合もある。本仕様上で明確に実装依存または実装定義と定義されたもの以外の変更を行う場合は、標準パッケージ名を利用できないものとする。

2.3.1.2 ベンダ実装のパッケージ命名規則

Java の言語仕様から、パッケージ名は Internet ドメイン名(*XXX*)で始まり、管理識別用の名前(*YYY*)が続く(表 2.2)。

表 2.2 ベンダ実装の Java パッケージ命名規則

識別	JTRON 標準パッケージ名
共通仕様で使用されるパッケージ	*XXX*.jtron.*YYY*
タイプ 1 で使用されるパッケージ	*XXX*.jtron.attach.*YYY*
タイプ 2 で使用されるパッケージ	*XXX*.jtron.shared.*YYY*
タイプ 3 で使用されるパッケージ	*XXX*.jtron.stream.*YYY*

ベンダがパッケージを拡張する場合は、ベンダが JTRON 標準パッケージ名に準じた名前を付与するものとする。すなわち、*XXX* にベンダのドメイン名がくる。このようにすることにより、パッケージ構成を同じにし、利用者の便を図る。ベンダにおいて同じ名前で中身が違うのは許されないものとする。この場合は名前を変える、あるいは、ベンダで独自のパッケージ名にする。

2.3.2 JTRON 標準 Java クラス構成

クラス定義において、プログラマに見せるメソッド名や変数名は、public とし、ベンダ依存は public にしない。
なお、本仕様書では、スーパークラスで定義されたメソッドのうち、オーバライドしているメソッドは記述していない。ベンダは、必要に応じて適切にオーバライドしなければならない。

例 ・ Object#toString()
・ Throwable#getMessage()

2.3.3 Java システムプロパティ

以下のシステムプロパティを標準で提供する。

jtron.version: Ver.*X.YY.XX*[.*WW*]
提供する JTRON の仕様バージョン番号(μITRON のバージョン番号の表記に準じる)

jtron.type:
タイプ番号。以下の英数字１文字以上を組み合わせで表記する。
0: アタッチクラス
1: 共有オブジェクトインタフェース
2: ストリームインタフェース
3〜9, A〜Z: 将来の拡張のため予約

jtron.vendor:
ベンダ名(ベンダで自由に定めてよい)

これらのプロパティの値は、org.jtron.JtronSystem クラスの getProperty メソッドで取得できる。

2.3.4 μITRON 仕様と Java のデータ型について

μITRON 仕様ではプラットフォーム毎に実際の型を決めており実装定義となっている。
本仕様では Java のデータ型との対応を以下のように規定する。

μITRON 仕様での型	意　　味	Java での型
B	符号付き 8 ビット整数	byte
H	符号付き 16 ビット整数	short
W	符号付き 32 ビット整数	int
D	符号付き 64 ビット整数	long
UB	符号無し 8 ビット整数	short
UH	符号無し 16 ビット整数	int
UW	符号無し 32 ビット整数	long
UD	符号無し 64 ビット整数	long(*1)
VB	データタイプが定まらない 8 ビットの値	byte
VH	データタイプが定まらない 16 ビットの値	short
VW	データタイプが定まらない 32 ビットの値	int
VD	データタイプが定まらない 64 ビットの値	long

μITRON 仕様での型	意　　味	Java での型
VP	データタイプが定まらないものへのポインタ	int(*2)
FP	プログラムの起動番地	int(*2)
INT	プロセッサに自然なサイズの符号付き整数	int(*2)
UINT	プロセッサに自然なサイズの符号無し整数	int(*2)
BOOL	真偽値(TRUE または FALSE)	boolean
FN	機能コード(符号付き整数)	int(*2)
ER	エラーコード(符号付き整数)	int(*2)
ID	オブジェクトの ID 番号(符号付き整数)	int(*2)
ATR	オブジェクト属性(符号無し整数)	int(*2)
STAT	オブジェクトの状態(符号無し整数)	int(*2)
MODE	サービスコールの動作モード(符号無し整数)	int(*2)
PRI	優先度(符号付き整数)	int(*2)
SIZE	メモリ領域のサイズ(符号無し整数)	int(*2)
TMO	タイムアウト指定(符号付き整数、時間単位は実装定義)	int(*3)
RELTIM	相対時間(符号無し整数、時間単位は実装定義)	int(*3)
SYSTIM	システム時刻(符号無し整数､時間単位は実装定義)	int(*3)
VP_INT	データタイプが定まらないものへのポインタまたはプロセッサに自然なサイズの符号付き整数	int(*2)
ER_BOOL	エラーコードまたは真偽値	int(*2)
ER_ID	エラーコードまたは ID 番号(負の ID 番号は表現できない)	int(*2)
ER_UINT	エラーコードまたは符号無し整数(符号無し整数の有効ビット数は UINT より 1 ビット短い)	int(*2)
TEXPTN	タスク例外要因のビットパターン(符号無し整数)	int(*2)
FLGPTN	イベントフラグのビットパターン(符号無し整数)	int(*2)
T_MSG	メールボックスのメッセージヘッダ	(*4)
T_MSG_PRI	メールボックスの優先度付きメッセージヘッダ	(*4)

μITRON 仕様での型	意　　味	Java での型
RDVPTN	ランデブ条件のビットパターン(符号無し整数)	int(*2)
RDVNO	ランデブ番号	int(*2)
OVRTIM	プロセッサ時間(符号無し整数、時間単位は実装定義)	int(*2)
INHNO	割込みハンドラ番号	int(*2)
INTNO	割込み番号	int(*2)
EXCNO	CPU 例外ハンドラ番号	int(*2)

(*1)符号無しの値が扱えるように一つ大きな型を利用するが、long より大きい型がないため D と同様に long 型とする。

(*2)本仕様書中での Java での型の記述は、int 型で統一して記述されているが実装定義とする。本仕様書中の実装定義にて型変更できる項目には"(★)"をパラメータ説明につける。

(*3)時間単位はスタンダードプロファイルに沿って 1 ミリ秒とする。Java での型は(*2)同様の取り扱いとする。

(*4)本仕様中では、該当する Java のデータ型を規定しない。

2.3.5 Java の例外の取り扱いについて

μITRON 仕様 OS が返すエラーコードに対応する例外は、メソッドの【例外】説明には個別には記述しておらず、関連する例外処理が必要な場合は、スーパークラス(ItronCauseException)にて一括して記述する。これにより、ベンダは適切なエラーコードに対応する例外を発生させる処理が必要となる。また、Java 実行環境の実装依存に伴い、JtronCauseException を実装依存にて適宜発生させても良い。

実装依存により JtronCauseException または ItronCauseException が発生すると思われる個所は双方のスーパークラスである JtronException にて記述する。

2.4 準拠性

タイプ１、タイプ２、タイプ３のいずれかの仕様(拡張仕様を除く)を実装していれば「JTRON 仕様に準拠している」といってよいものとする。

第 3 章 共通仕様

3.1 概要

Java スレッドとリアルタイムタスクのマッピングを規定する ITRON API と、システムクラス、例外クラスについての定義を行う。
Java スレッドとリアルタイムタスクに関連する ITRON API には、Java スレッドとリアルタイムタスクの対応の ITRON API を定義する。

3.2 ITRON API

3.2.1 Java スレッドとリアルタイムタスクの対応

Java スレッドとリアルタイムタスクの対応方法には、Java スレッドの実装により大きく 2 つある。

1. 1 つの Java スレッドを 1 つのリアルタイムタスクで実装させる。したがって、1 つの Java スレッドを 1 つのリアルタイムタスクが対応する。
2. Java 実行環境を 1 つのリアルタイムタスクで実装する(グリーンスレッド)。この場合、全ての Java スレッドが 1 つのリアルタイムタスクに対応することになる。

JTRON では、1 による対応を前提とし、以下の API により対応関係を設定、参照することができる。

API 名	概 要	種 別
jti_set_hpr	Java 実行環境を実装するリアルタイムタスクの最高優先度を設定する。	標準仕様
jti_get_hpr	Java 実行環境を実装するリアルタイムタスクの最高優先度を取得する。	標準仕様
jti_cnv_jpr	Java スレッドの優先度からリアルタイムタスクの優先度を求める。	標準仕様
jti_cnv_lpr	Java 実行環境を実装するリアルタイムタスクの最低優先度を求める。	標準仕様

(1)Java スレッドからリアルタイムタスクの優先度への対応を定義する。すなわち、Java 実行環境を実装するリアルタイムタスクの優先度のうち最高優先度を定義する。

【仕様決定の理由】

Java スレッドとリアルタイムタスクの間の優先度マッピングを定めるにあたって以下のような方法が考えられた。

- Java スレッドの優先度とリアルタイムタスクの優先度のマッピングを表の形式で設定、参照できるものとする。
- Java 実行環境を実装するリアルタイムタスクの最高優先度、最低優先度(あるいはリアルタイムタスクで利用可能なリアルタイムタスクの最高優先度、最低優先度)を定義可能とする。

しかし、この方法は、以下の理由から採用しなかった。

- Java スレッドよりも高い優先度でリアルタイムタスクを実行するため、リアルタイムタスクプログラマは、Java 実行環境を実現するタスクの最高優先度のみに関心があり最低優先度の方には関心がないと考えられる。

JTI_SET_HPR 標準仕様

jti_set_hpr 標準仕様

Java 実行環境を実装するリアルタイムタスクの最高優先度を設定する。

【C 言語 API】

```
ER ercd = jti_set_hpr(PRI hijpr);
```

【静的 API】

```
JTI_SET_HPR(PRI hijpr);
```

【パラメータ】

PRI	hijpr	Java 実行環境を実装するリアルタイムタスクの最高優先度

【リターンパラメータ】

ER	ercd	終了(E_OK)またはエラーコード

【エラーコード】

E_PAR	パラメータエラー(hijpr が不正)

【機能】

Java 実行環境を実装するリアルタイムタスクの最高優先度の値を設定する。本 API が実行されない場合、JTI_DFL_HPR が初期値として設定される。 動的に最高優先度の値を変更した場合、既に動作している Java スレッドの優先度への反映については実装定義とする。

【JTRON2.0 仕様との相違】

エラーコードを返す。

【補足説明】

動的 API **は、**Java **実行環境が起動される前に呼び出される事を想定している。**Java **実行環境が起動後の動作は、未定義とする。**

jti_get_hpr 標準仕様

Java 実行環境を実装するリアルタイムタスクの最高優先度を取得する。

【C 言語 API】

```
ER ercd = jti_get_hpr(PRI *p_hijpr);
```

【リターンパラメータ】

ER	ercd	正常終了(E_OK)またはエラーコード
PRI	hijpr	Java 実行環境を実装するリアルタイムタスクの最高優先度

【エラーコード】

E_OBJ	オブジェクト状態エラー

【機能】

Java 実行環境を実装するリアルタイムタスクの最高優先度を取得する。

【JTRON2.0 仕様との相違】

新規追加の API である。

【補足説明】

動的 API は、Java 実行環境が起動される前に呼び出される事を想定している。Java 実行環境が起動後の動作は、未定義とする。

JTI_CNV_JPR 標準仕様
jti_cnv_jpr 標準仕様

Java スレッドの優先度からリアルタイムタスクの優先度を求める。

【C 言語 API】

```
ER ercd = jti_cnv_jpr(INT jpr, PRI *p_pri);
```

【静的 API】

```
PRI pri = JTI_CNV_JPR(PRI hijpr, INT jpr);
```

【パラメータ】

PRI	hijpr	Java 実行環境を実装するリアルタイムタスクの最高優先度
INT	jpr	Java スレッドの優先度

【リターンパラメータ】

ER	ercd	正常終了(E_OK)またはエラーコード
PRI	pri	リアルタイムタスクの優先度

【エラーコード】

E_PAR	パラメータエラー(jpr が不正)

【機能】

Java スレッドの優先度からリアルタイムタスクの優先度を求める。

【JTRON2.0 仕様との相違】

意味合いを明確にするため jti_get_hpr から名称変更した。
静的 API に必要な hijpr は、動的 API に必要ないので削除した。

【補足説明】

逆変換(リアルタイムタスクの優先度から Java スレッドの優先度を求める、jti_cnv_hpr)は、検討したが使用方法が不明なため定義しない。動的 API は、Java 実行環境が起動される前に呼び出される事を想定している。Java 実行環境が起動後の動作は、未定義とする。

JTI_CNV_LPR 標準仕様

jti_cnv_lpr 標準仕様

Java 実行環境を実装するリアルタイムタスクの最低優先度を求める。

【C 言語 API】

```
ER ercd = jti_cnv_lpr(PRI *p_lwjpr);
```

【静的 API】

```
PRI lwjpr = JTI_CNV_LPR(PRI hijpr);
```

【パラメータ】

PRI	hijpr	Java 実行環境を実装するリアルタイムタスクの最高優先度

【リターンパラメータ】

ER	ercd	正常終了(E_OK)またはエラーコード
PRI	lwjpr	Java 実行環境を実装するリアルタイムタスクの最低優先度

【機能】

Java 実行環境を実装するリアルタイムタスクの最低優先度を求める。

【JTRON2.0 仕様との相違】

新規追加の API である。

【補足説明】

動的 API は、Java 実行環境が起動される前に呼び出される事を想定している。Java 実行環境が起動後の動作は、未定義とする。

3.3 Java API

3.3.1 パッケージ構成

JTRON システムの全体を管理や制御するためのクラスは、org.jtron パッケージにまとめられている。以下のクラスから構成される。

パッケージ名：org.jtron

クラス：JtronSystem, JtronException

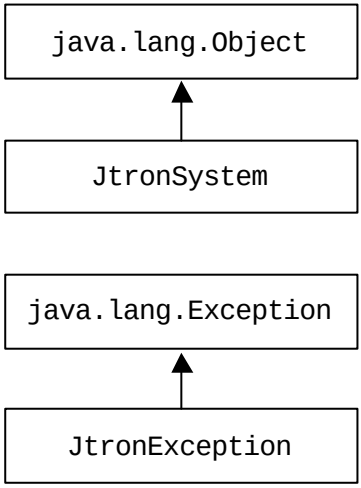


図 3.3: org.jtron パッケージのクラス構成

3.3.2 JTRON システムクラス(JtronSystem)

```
java.lang.Object
    |
    └── org.jtron.JtronSystem
```

public class JtronSystem **標準仕様**

JTRON に関するプロパティの情報などを管理する。

■クラス定義

```
package org.jtron;

public class JtronSystem {
        public static String getProperty(String key);
        public static String getProperty(String key,
                                  String defaultValue);
        public static Properties getProperties();
}
```

■static **メソッド**

public static String getProperty(String key);

【パラメータ】

String key **キーとなる文字列**

【戻り値】

String **キーに対応した文字列**

【機能】

指定されたキーで示される JTRON **システムプロパティを得る。**

```
public static String getProperty(String key,
                                 String defaultValue);
```

【パラメータ】

String	key	キーとなる文字列
String	defaultValue	デフォルトの文字列

【戻り値】

String	キーに対応した文字列

【機能】

指定されたキーで示される JTRON システムプロパティを得る。key で指定したプロパティがみつからない場合は、defaultValue の文字列を返す。

```
public static Properties getProperties();
```

【戻り値】

Properties	Properties クラスのオブジェクト

【機能】

JTRON システムプロパティを得る。以下のプロパティは、標準で定義されている。

jtron.version: Ver.*X*.*YY*.*XX*[.*WW*]
提供する JTRON の仕様バージョン番号(μITRON のバージョン番号の表記に準じる)。

jtron.type:
タイプ番号。以下の英数字 1 文字以上を組み合わせで表記する。

- 0: アタッチクラス
- 1: 共有オブジェクトインタフェース
- 2: ストリームインタフェース
- 3〜9, A〜Z: 将来の拡張のため予約

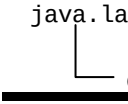
jtron.vendor:
ベンダ名(ベンダで自由に定めてよい)

【JTRON2.0 仕様との相違】

・JtiSystem クラスの名称を略称を使わない JtronSystem に変更した。

・getJtiSystem メソッドを削除した。これを呼び出すことにより Java 側の JTRON 機構の動作が保証されることになっていたが、初期化処理が必要ならば事前に行なうよう実装することが可能であること、また他のインスタンスメソッドを static メソッドに変更することでインスタンスを得る必要がなくなったことによる。

3.3.3 JTRON 例外クラス(JtronException)



public class JtronException **標準仕様**

JTRON クラスで取り扱うすべての例外クラスの基底クラス。

■クラス定義

```
package org.jtron;

public class JtronException extends Exception {
        public JtronException();
        public JtronException(String msg);
}
```

■コンストラクタ

public JtronException();

【機能】

詳細なメッセージ無しで JtronException **を生成する。**

public JtronException(String msg);

【パラメータ】

String msg **詳細メッセージ文字列**

【機能】

指定された詳細メッセージ msg **をもつ** JtronException **を生成する。**

【補足説明】

階層化された例外クラスの使い方については、「4.2.1 ITRON **による例外クラス(**`ItronCauseException`**)」の【補足説明】を参照せよ。**

【JTRON1.0 **仕様との相違】**

アタッチクラス(タイプ 1)**から共通仕様に移動する。そして他のタイプの例外クラスはすべて本クラスを継承するものとする。そのため本クラスの中身は空とし、従来のメンバを** ITRON **による例外クラス(**`ItronCauseException`**)に移動する。**

第 4 章 タイプ 1 インタフェース

(アタッチクラス／メモリ操作クラス／例外クラス)

4.1 概要

アタッチクラスは Java プログラムから ITRON の機能を利用する手段を提供する。タスク、セマフォ、メールボックスといった ITRON 機能毎にそれに対応するクラスを設け、そのクラスのインスタンスを ITRON カーネルオブジェクトに対応させる。ITRON カーネルオブジェクトの操作は生成したインスタンスのメソッドを呼び出すことで行なうので、いちいちオブジェクトの ID を指定する必要はない。これは Java のオブジェクト指向プログラミングになじむものである。
アタッチという名称はクラスのインスタンスが ITRON カーネルオブジェクトに１対１対応するところから来ている。対応の方法として、インスタンス生成時に以下のどちらかを選択することができる。

(1) ITRON カーネルオブジェクトを生成し、それを生成インスタンスに対応させる。
(2) 既に存在する ITRON カーネルオブジェクトに生成インスタンスを対応させる。

メモリ操作クラスはアドレスを意識しないでメモリを扱う手段を提供する。Java にはメモリアドレスの概念がないので、ITRON の機能を利用する場合に必要なアドレスを指定することができない。またその内容を操作することもできない。メモリ操作クラスを用いるとストリーム I/O のような形式でメモリを扱うことができる。

例外クラスはアタッチクラスとメモリ操作クラスで発生するエラーを Java の例外処理機構で扱うための手段を提供する。ITRON 由来のエラーとそれ以外のエラーに分類されており、ITRON 由来のエラーはさらに ITRON のエラーコード毎に別々のクラスに派生定義されているため、例外処理機構の利用を容易にする。

パッケージ名：org.jtron.attach

例外クラス：

ITRON による例外クラス

- ItronCauseException

各 ITRON サービスコール例外クラス

- ItronSYSException
- ItronNOSPTException
- ItronRSFNException
- ItronRSATRException
- ItronPARException
- ItronIDException
- ItronCTXException
- ItronMACVException
- ItronOACVException
- ItronILUSEException
- ItronNOMEMException
- ItronNOIDException
- ItronOBJException
- ItronNOEXSException
- ItronQOVRException
- ItronRLWAIException
- ItronTMOUTException
- ItronDLTException
- ItronCLSException
- ItronWBLKException
- ItronBOVRException

JTRON による例外クラス

- JtronCauseException

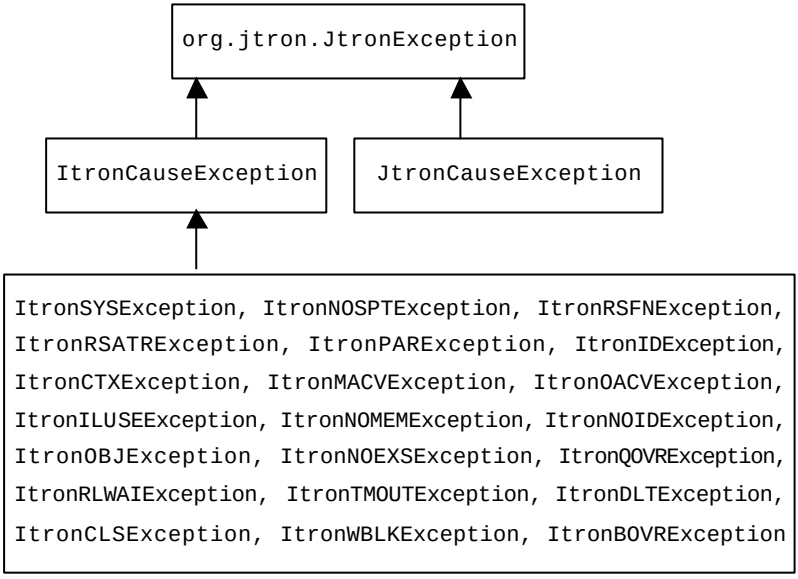


図 4.1: org.jtron.attach パッケージの例外クラス構成

アタッチクラスは下層のμITRON 仕様 OS の実装に大きく影響される。そのため実装されていないサービスコールが必要なクラス・メソッドの扱いをどうするかを以下のどちらかの実装定義とする。

(1)該当クラス・メソッドを実装しない。この場合、実行時これを呼び出すと ClassNotFoundException、NoSuchMethodException のような例外が発生する。

(2)すべてのクラス・メソッドを実装し、実行時これを呼び出した時 ItronNOSPTException 例外を起こす。

クラス：

メモリ操作クラス

ItronMemory

タスク

Task, T_CTSK, T_RTST, T_RTSK, T_DTEX, T_RTEX, T_ROVR

セマフォ

Semaphore, T_CSEM, T_RSEM

イベントフラグ

EventFlag, T_CFLG, T_RFLG

データキュー

DataQueue, T_CDTQ, T_RDTQ

メールボックス

MailBox, T_CMBX, T_RMBX

ミューテックス

Mutex, T_CMTX, T_RMTX

メッセージバッファ

MessageBuffer, T_CMBF, T_RMBF

ランデブ

RendezvousPort, T_CPOR, T_RPOR, Rendezvous, T_RRDV

固定長メモリプール

FixedMemoryPool, T_CMPF, T_RMPF, ItronFixedMemory

可変長メモリプール

VariableMemoryPool, T_CMPL, T_RMPL, ItronVariableMemory

周期ハンドラ

CyclicHandler, T_CCYC, T_RCYC

アラームハンドラ

AlarmHandler, T_CALM, T_RALM

割込みサービスルーチン

InterruptServiceRoutine, T_CISR, T_RISR

その他

Kernel, T_DOVR, T_RSYS, T_DINH, T_DSVC, T_DEXC, T_RCFG, T_RVER

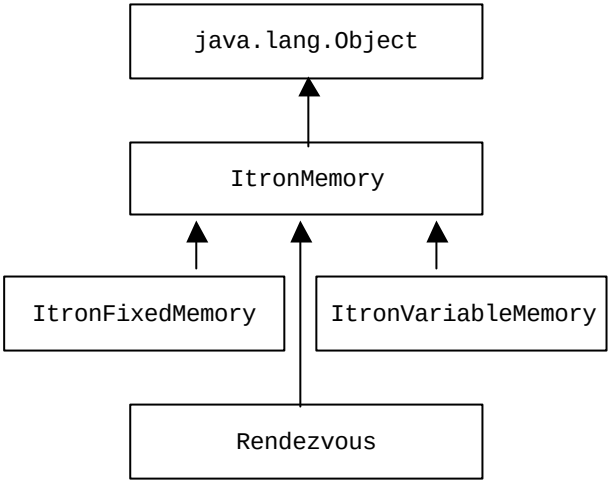


図 4.2: org.jtron.attach パッケージのメモリ操作クラス構成

4.2 Java API

4.2.1 ITRON による例外クラス(ItronCauseException)

```
org.jtron.JtronException
    |
    └── org.jtron.attach.ItronCauseException
```

public class ItronCauseException 標準仕様

各 ITRON サービスコール例外クラスの基底クラス

ITRON サービスコールエラー時投げられる例外である。java.lang.Throwable クラスから継承した toString() により得られる文字列には、エラーを起こしたサービスコール名とそのメインエラーコード名、サブエラーコードの値が含まれる。

■クラス定義

```
package org.jtron.attach;

public class ItronCauseException
                        extends org.jtron.JtronException {

        public ItronCauseException(String message,
           int resourceId, int functionCode, int errorCode);

        public int resourceId;
        public int functionCode;
        public int errorCode;

        public final int MERCD();
        public final int SERCD();

        public static final int  // ITRON 機能コード一覧
          // -0x01～-0x04 までは予約
          TFN_CRE_TSK    = -0x05, TFN_DEL_TSK   = -0x06,
```

```
TFN_ACT_TSK    = -0x07, TFN_CAN_ACT  = -0x08,
TFN_STA_TSK    = -0x09, TFN_EXT_TSK  = -0x0a,
TFN_EXD_TSK    = -0x0b, TFN_TER_TSK  = -0x0c,
TFN_CHG_PRI    = -0x0d, TFN_GET_PRI  = -0x0e,
TFN_REF_TSK    = -0x0f, TFN_REF_TST  = -0x10,
TFN_SLP_TSK    = -0x11, TFN_TSLP_TSK = -0x12,
TFN_WUP_TSK    = -0x13, TFN_CAN_WUP  = -0x14,
TFN_REL_WAI    = -0x15, TFN_SUS_TSK  = -0x16,
TFN_RSM_TSK    = -0x17, TFN_FRSM_TSK = -0x18,
TFN_DLY_TSK    = -0x19, // -0x1a は予約
TFN_DEF_TEX    = -0x1b, TFN_RAS_TEX  = -0x1c,
TFN_DIS_TEX    = -0x1d, TFN_ENA_TEX  = -0x1e,
TFN_SNS_TEX    = -0x1f, TFN_REF_TEX  = -0x20,
TFN_CRE_SEM    = -0x21, TFN_DEL_SEM  = -0x22,
TFN_SIG_SEM    = -0x23, // -0x24 は予約
TFN_WAI_SEM    = -0x25, TFN_POL_SEM  = -0x26,
TFN_TWAI_SEM   = -0x27, TFN_REF_SEM  = -0x28,
TFN_CRE_FLG    = -0x29, TFN_DEL_FLG  = -0x2a,
TFN_SET_FLG    = -0x2b, TFN_CLR_FLG  = -0x2c,
TFN_WAI_FLG    = -0x2d, TFN_POL_FLG  = -0x2e,
TFN_TWAI_FLG   = -0x2f, TFN_REF_FLG  = -0x30,
TFN_CRE_DTQ    = -0x31, TFN_DEL_DTQ  = -0x32,
// -0x33～-0x34 までは予約
TFN_SND_DTQ    = -0x35, TFN_PSND_DTQ = -0x36,
TFN_TSND_DTQ   = -0x37, TFN_FSND_DTQ = -0x38,
TFN_RCV_DTQ    = -0x39, TFN_PRCV_DTQ = -0x3a,
TFN_TRCV_DTQ   = -0x3b, TFN_REF_DTQ  = -0x3c,
TFN_CRE_MBX    = -0x3d, TFN_DEL_MBX  = -0x3e,
TFN_SND_MBX    = -0x3f, // -0x40 は予約
TFN_RCV_MBX    = -0x41, TFN_PRCV_MBX = -0x42,
TFN_TRCV_MBX   = -0x43, TFN_REF_MBX  = -0x44,
TFN_CRE_MPF    = -0x45, TFN_DEL_MPF  = -0x46,
TFN_REL_MPF    = -0x47, // -0x48 は予約
TFN_GET_MPF    = -0x49, TFN_PGET_MPF = -0x4a,
TFN_TGET_MPF   = -0x4b, TFN_REF_MPF  = -0x4c,
TFN_SET_TIM    = -0x4d, TFN_GET_TIM  = -0x4e,
TFN_CRE_CYC    = -0x4f, TFN_DEL_CYC  = -0x50,
```

```
TFN_STA_CYC    = -0x51, TFN_STP_CYC  = -0x52,
TFN_REF_CYC    = -0x53, // -0x54 は予約
TFN_ROT_RDQ    = -0x55, TFN_GET_TID  = -0x56,
// -0x57～-0x58 までは予約
TFN_LOC_CPU    = -0x59, TFN_UNL_CPU  = -0x5a,
TFN_DIS_DSP    = -0x5b, TFN_ENA_DSP  = -0x5c,
TFN_SNS_CTX    = -0x5d, TFN_SNS_LOC  = -0x5e,
TFN_SNS_DSP    = -0x5f, TFN_SNS_DPN  = -0x60,
TFN_REF_SYS    = -0x61, // -0x62 は予約
// -0x63～-0x64 までは予約
TFN_DEF_INH    = -0x65, TFN_CRE_ISR  = -0x66,
TFN_DEL_ISR    = -0x67, TFN_REF_ISR  = -0x68,
TFN_DIS_INT    = -0x69, TFN_ENA_INT  = -0x6a,
TFN_CHG_IXX    = -0x6b, TFN_GET_IXX  = -0x6c,
TFN_DEF_SVC    = -0x6d, TFN_DEF_EXC  = -0x6e,
TFN_REF_CFG    = -0x6f, TFN_REF_VER  = -0x70,
TFN_IACT_TSK   = -0x71, TFN_IWUP_TSK = -0x72,
TFN_IREL_WAI   = -0x73, TFN_IRAS_TEX = -0x74,
TFN_ISIG_SEM   = -0x75, TFN_ISET_FLG = -0x76,
TFN_IPSND_DTQ  = -0x77, TFN_IFSND_DTQ= -0x78,
TFN_IROT_RDQ   = -0x79, TFN_IGET_TID = -0x7a,
TFN_ILOC_CPU   = -0x7b, TFN_IUNL_CPU = -0x7c,
TFN_ISIG_TIM   = -0x7d, // -0x7e は予約
// -0x7f～-0x80 までは予約
TFN_CRE_MTX    = -0x81, TFN_DEL_MTX  = -0x82,
TFN_UNL_MTX    = -0x83, // -0x84 は予約
TFN_LOC_MTX    = -0x85, TFN_PLOC_MTX = -0x86,
TFN_TLOC_MTX   = -0x87, TFN_REF_MTX  = -0x88,
TFN_CRE_MBF    = -0x89, TFN_DEL_MBF  = -0x8a,
// -0x8b～-0x8c までは予約
TFN_SND_MBF    = -0x8d, TFN_PSND_MBF = -0x8e,
TFN_TSND_MBF   = -0x8f, // -0x90 は予約
TFN_RCV_MBF    = -0x91, TFN_PRCV_MBF = -0x92,
TFN_TRCV_MBF   = -0x93, TFN_REF_MBF  = -0x94,
TFN_CRE_POR    = -0x95, TFN_DEL_POR  = -0x96,
TFN_CAL_POR    = -0x97, TFN_TCAL_POR = -0x98,
TFN_ACP_POR    = -0x99, TFN_PACP_POR = -0x9a,
```

```
 TFN_TACP_POR   = -0x9b, TFN_FWD_POR  = -0x9c,
 TFN_RPL_RDV    = -0x9d, TFN_REF_POR  = -0x9e,
 TFN_REF_RDV    = -0x9f, // -0xa0 は予約
 TFN_CRE_MPL    = -0xa1, TFN_DEL_MPL  = -0xa2,
 TFN_REL_MPL    = -0xa3, // -0xa4 は予約
 TFN_GET_MPL    = -0xa5, TFN_PGET_MPL = -0xa6,
 TFN_TGET_MPL   = -0xa7, TFN_REF_MPL  = -0xa8,
 TFN_CRE_ALM    = -0xa9, TFN_DEL_ALM  = -0xaa,
 TFN_STA_ALM    = -0xab, TFN_STP_ALM  = -0xac,
 TFN_REF_ALM    = -0xad, // -0xae は予約
 // -0xaf～-0xb0 までは予約
 TFN_DEF_OVR    = -0xb1, TFN_STA_OVR  = -0xb2,
 TFN_STP_OVR    = -0xb3, TFN_REF_OVR  = -0xb4,
 // -0xb5～-0xc0 までは予約
 TFN_ACRE_TSK   = -0xc1, TFN_ACRE_SEM = -0xc2,
 TFN_ACRE_FLG   = -0xc3, TFN_ACRE_DTQ = -0xc4,
 TFN_ACRE_MBX   = -0xc5, TFN_ACRE_MTX = -0xc6,
 TFN_ACRE_MBF   = -0xc7, TFN_ACRE_POR = -0xc8,
 TFN_ACRE_MPF   = -0xc9, TFN_ACRE_MPL = -0xca,
 TFN_ACRE_CYC   = -0xcb, TFN_ACRE_ALM = -0xcc,
 TFN_ACRE_ISR   = -0xcd; // -0xce は予約
 // -0xcf～-0xe0 までは予約
 // -0xe1～-0xff までは実装独自のサービスコール

public static final int // エラーコード一覧
 E_SYS     = -5,   // システムエラー
 E_NOSPT   = -9,   // 未サポート機能
 E_RSFN    = -10,  // 予約機能コード
 E_RSTAR   = -11,  // 予約属性
 E_PAR     = -17,  // パラメータエラー
 E_ID      = -18,  // 不正 ID 番号
 E_CTX     = -25,  // コンテキストエラー
 E_MACV    = -26,  // メモリアクセス違反
 E_OACV    = -27,  // オブジェクトアクセス違反
 E_ILUSE   = -28,  // サービスコール不正使用
 E_NOMEM   = -33,  // メモリ不足
 E_NOID    = -34,  // ID 番号不足
```

```
        E_OBJ     = -41, // オブジェクト状態エラー
        E_NOEXS   = -42, // オブジェクト未生成
        E_QOVR    = -43, // キューイングオーバーフロー
        E_RLWAI   = -49, // 待ち状態の強制解除
        E_TMOUT   = -50, // ポーリング失敗またはタイムアウト
        E_DLT     = -51, // 待ちオブジェクトの削除
        E_CLS     = -52, // 待ちオブジェクトの状態変化
        E_WBLK    = -57, // ノンブロッキング受付け
        E_BOVR    = -58; // バッファオーバーフロー
}
```

■定数

ITRON 機能コード一覧
エラーコード一覧

■変数

`int resourceId`	異常発生時に操作したオブジェクト ID(★)
`int functionCode`	異常発生時の ITRON 機能コード(★)
`int errorCode`	異常発生時のエラーコード(★)

■コンストラクタ

```
public ItronCauseException(String message,
        int resourceId, int functionCode, int errorCode);
```

【パラメータ】

`String message`	詳細メッセージ
`int resourceId`	オブジェクト ID(★)
`int functionCode`	ITRON 機能コード(★)
`int errorCode`	エラーコード(★)

【機能】

詳細メッセージを持つインスタンスを生成する。

■メソッド

```
public final int MERCD();
```

【戻り値】

int メインエラーコード(★)

【機能】

errorCode からメインエラーコードを取得する。

```
public final int SERCD();
```

【戻り値】

int サブエラーコード(★)

【機能】

errorCode からサブエラーコードを取得する。

【補足説明】

本クラスは各 ITRON サービスコール例外クラスと組み合わせてつぎのように使うことを想定している。
これはメールボックスからメッセージを受信し、受信待ちの間 0.5 秒ごとに LED(ランプ)を点滅させる処理の例である。

```
ItronMemory msg = null;
MailBox mbx = new MailBox(10);// ID=10 のメールボックスに接続
for (;;) {
   try{
      // 0.5 秒タイムアウト付きで受信待ち
      msg = mbx.receive(1024, 500);
      break;  // 正常受信完了；本来の処理へ
```

```
    }
    catch (ItronRLWAIException e1) { // 待ちの強制解除;無視
    }
    catch (ItronTMOUTException e2) { // タイムアウト
       boolean led = SystemLED.getStatus();//LED の状態を得る
       SystemLED.putStatus(!led); // 状態を反転
    }
    catch (ItronCauseException e_ic) { // その他の ITRON エラー
       handleITRONerror(e_ic); // →エラー処理
    }
    catch (JtronException e_j) { // その他のエラー
       handleError(e_j);         // →エラー処理
    }
 }
 // ～受信メッセージの処理～
```

try 節内の MailBox#receive(int, int)でメールボックスからメッセージを受信する。メッセージが正常に受信できれば break 文が実行されメッセージが処理される。
受信メソッド呼び出しが異常終了すると catch節のいずれかが実行される。catch 節は４つあり、そこに指定されている例外クラスは継承のレベルにおいて下位→上位という順に配置されている。
まず最初の２つはITRON サービスコールの個別エラーコードに対応した処理であり、待ちの強制解除とタイムアウトに対応している。タイムアウト時には正常処理として LED(ランプ)の状態を反転する。
３つめは最初の２つの上位クラスであり、すべての ITRON サービスコールエラーを含んでいる。最初の２つで捕らえられなかった ITRON サービスコールエラーをここで捕らえて処理できる。
４つめは３つめのさらに上位のクラスであり、ITRON サービスコールエラー以外のエラーを一括処理することができる。

つまり、最初の方で当然起こると想定される個別のエラーを捕らえて処理し、その後の上位の例外クラスを配置して catch 節を記述することでより広範囲のエラーを処理する。ここでは注目すべきエラーだけ処理して、その他は呼び出し元に処理をまかせるようにもできる。

catch 節の順が重要であることに注意されたい。下位クラスから上位

クラスへという配置でなければ期待するように動作しない。

【JTRON1.0 仕様との相違】

・JtronException が共通仕様に移動するので、その意味合いを受け継いだ新クラスである。

・大文字で始まるフィールドを Java コアクラスの習慣に合わせ小文字から始まる名前に変更する。

・各フィールドの意味を明確にする。継承元の java.lang.Throwable クラスに toString メソッド があるので errorName は廃止する。

・サービスコール名はメッセージ文字列に含まれるので apiName フィールドを廃止する。

・メインエラーコードとサブエラーコードを取り出すメソッド、ITRON サービスコール番号の定数、メインエラーコードの定数を追加する。

4.2.2 各 ITRON サービスコール例外クラス

ITRON サービスコールのエラーコード毎に対応する以下の複数のクラスから構成される。
各 ITRON サービスコールの例外クラスの命名規則は、メインエラーコード名の"E_"の部分を"Itron"に置き換えてあとに"Exception"を続けたクラス名となっている。

各 ITRON サービスコール例外クラス：

システムエラー	ItronSYSException
未サポート機能	ItronNOSPTException
予約機能コード	ItronRSFNException
予約属性	ItronRSATRException
パラメータエラー	ItronPARException
不正 ID 番号	ItronIDException
コンテキストエラー	ItronCTXException
メモリアクセス違反	ItronMACVException
オブジェクトアクセス違反	ItronOACVException
サービスコール不正使用	ItronILUSEException
メモリ不足	ItronNOMEMException
ID 番号不足	ItronNOIDException
オブジェクト状態エラー	ItronOBJException
オブジェクト未生成	ItronNOEXSException
キューイングオーバーフロー	ItronQOVRException
待ち状態の強制解除	ItronRLWAIException
ポーリング失敗またはタイムアウト	ItronTMOUTException
待ちオブジェクトの削除	ItronDLTException
待ちオブジェクトの状態変化	ItronCLSException
ノンブロッキング受付け	ItronWBLKException
バッファオーバーフロー	ItronBOVRException

4.2.2.1 システムエラー(ItronSYSException)

```
org.jtron.attach.ItronCauseException
    |
    └── org.jtron.attach.ItronSYSException
```

public class ItronSYSException 標準仕様

システムエラー

■クラス定義

```
package org.jtron.attach;

public class ItronSYSException extends ItronCauseException {
        public ItronSYSException(String message,
          int resourceId, int functionCode, int errorCode);
}
```

■コンストラクタ

```
public ItronSYSException(String message,
        int resourceId, int functionCode, int errorCode);
```

【パラメータ】

String	message	詳細メッセージ
int	resourceId	オブジェクト ID(★)
int	functionCode	ITRON 機能コード(★)
int	errorCode	エラーコード(★)

【機能】

詳細メッセージを持つインスタンスを生成する。

【JTRON1.0 仕様との相違】

ITRON サービスコールエラーに対応した新クラスである。

4.2.2.2 未サポート機能(ItronNOSPTException)

```
org.jtron.attach.ItronCauseException
    |
    └── org.jtron.attach.ItronNOSPTException
```

public class ItronNOSPTException 標準仕様

未サポート機能

■クラス定義

```
package org.jtron.attach;

public class ItronNOSPTException extends ItronCauseException {
        public ItronNOSPTException(String message,
           int resourceId, int functionCode, int errorCode);
}
```

■コンストラクタ

```
public ItronNOSPTException(String message,
        int resourceId, int functionCode, int errorCode);
```

【パラメータ】

String message	詳細メッセージ
int resourceId	オブジェクト ID(★)
int functionCode	ITRON 機能コード(★)
int errorCode	エラーコード(★)

【機能】

詳細メッセージを持つインスタンスを生成する。

【JTRON1.0 仕様との相違】

ITRON サービスコールエラーに対応した新クラスである。

4.2.2.3 予約機能コード(ItronRSFNException)

```
org.jtron.attach.ItronCauseException
    |
    └── org.jtron.attach.ItronRSFNException
```

public class ItronRSFNException 標準仕様

予約機能コード

■クラス定義

```
package org.jtron.attach;

public class ItronRSFNException extends ItronCauseException {
        public ItronRSFNException(String message,
           int resourceId, int functionCode, int errorCode);
}
```

■コンストラクタ

```
public ItronRSFNException(String message,
        int resourceId, int functionCode, int errorCode);
```

【パラメータ】

String message	詳細メッセージ
int resourceId	オブジェクト ID(★)
int functionCode	ITRON 機能コード(★)
int errorCode	エラーコード(★)

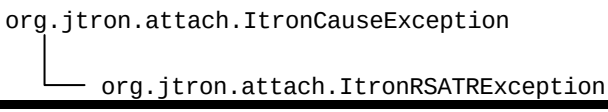
【機能】

詳細メッセージを持つインスタンスを生成する。

【JTRON1.0 仕様との相違】

ITRON サービスコールエラーに対応した新クラスである。

4.2.2.4 予約属性(ItronRSATRException)



public class ItronRSATRException 標準仕様

予約属性

■クラス定義

```
package org.jtron.attach;

public class ItronRSATRException extends ItronCauseException {
        public ItronRSATRException(String message,
            int resourceId, int functionCode, int errorCode);
}
```

■コンストラクタ

```
public ItronRSATRException(String message,
        int resourceId, int functionCode, int errorCode);
```

【パラメータ】

String	message	詳細メッセージ
int	resourceId	オブジェクト ID(★)
int	functionCode	ITRON 機能コード(★)
int	errorCode	エラーコード(★)

【機能】

詳細メッセージを持つインスタンスを生成する。

【JTRON1.0 仕様との相違】

ITRON サービスコールエラーに対応した新クラスである。

4.2.2.5 パラメータエラー(ItronPARException)

```
org.jtron.attach.ItronCauseException
    |
    └── org.jtron.attach.ItronPARException
```

public class ItronPARException **標準仕様**

パラメータエラー

■クラス定義

```
package org.jtron.attach;

public class ItronPARException extends ItronCauseException {
        public ItronPARException(String message,
           int resourceId, int functionCode, int errorCode);
}
```

■コンストラクタ

```
public ItronPARException(String message,
        int resourceId, int functionCode, int errorCode);
```

【パラメータ】

String message	詳細メッセージ
int resourceId	オブジェクト ID(★)
int functionCode	ITRON 機能コード(★)
int errorCode	エラーコード(★)

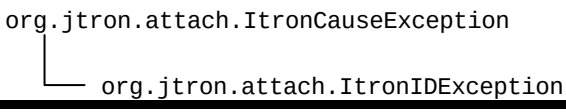
【機能】

詳細メッセージを持つインスタンスを生成する。

【JTRON1.0 仕様との相違】

ITRON サービスコールエラーに対応した新クラスである。

4.2.2.6 不正 ID 番号(ItronIDException)



public class ItronIDException **標準仕様**

不正 ID 番号

■クラス定義

```
package org.jtron.attach;

public class ItronIDException extends ItronCauseException {
        public ItronIDException(String message,
          int resourceId, int functionCode, int errorCode);
}
```

■コンストラクタ

```
public ItronIDException(String message,
        int resourceId, int functionCode, int errorCode);
```

【パラメータ】

String message	詳細メッセージ
int resourceId	オブジェクト ID(★)
int functionCode	ITRON 機能コード(★)
int errorCode	エラーコード(★)

【機能】

詳細メッセージを持つインスタンスを生成する。

【JTRON1.0 仕様との相違】

ITRON サービスコールエラーに対応した新クラスである。

4.2.2.7 コンテキストエラー(ItronCTXException)

```
org.jtron.attach.ItronCauseException
    |
    └── org.jtron.attach.ItronCTXException
```

public class ItronCTXException **標準仕様**

コンテキストエラー

■クラス定義

```
package org.jtron.attach;

public class ItronCTXException extends ItronCauseException {
        public ItronCTXException(String message,
          int resourceId, int functionCode, int errorCode);
}
```

■コンストラクタ

```
public ItronCTXException(String message,
        int resourceId, int functionCode, int errorCode);
```

【パラメータ】

String message	詳細メッセージ
int resourceId	オブジェクト ID(★)
int functionCode	ITRON 機能コード(★)
int errorCode	エラーコード(★)

【機能】

詳細メッセージを持つインスタンスを生成する。

【JTRON1.0 仕様との相違】

ITRON サービスコールエラーに対応した新クラスである。

4.2.2.8 メモリアクセス違反(ItronMACVException)

```
org.jtron.attach.ItronCauseException
    |
    └── org.jtron.attach.ItronMACVException
```

public class ItronMACVException 標準仕様

メモリアクセス違反

■クラス定義

```
package org.jtron.attach;

public class ItronMACVException extends ItronCauseException {
        public ItronMACVException(String message,
           int resourceId, int functionCode, int errorCode);
}
```

■コンストラクタ

```
public ItronMACVException(String message,
        int resourceId, int functionCode, int errorCode);
```

【パラメータ】

String	message	詳細メッセージ
int	resourceId	オブジェクト ID(★)
int	functionCode	ITRON 機能コード(★)
int	errorCode	エラーコード(★)

【機能】

詳細メッセージを持つインスタンスを生成する。

【JTRON1.0 仕様との相違】

ITRON サービスコールエラーに対応した新クラスである。

4.2.2.9 オブジェクトアクセス違反 (ItronOACVException)

```
org.jtron.attach.ItronCauseException
    |
    └── org.jtron.attach.ItronOACVException
```

public class ItronOACVException 標準仕様

オブジェクトアクセス違反

■クラス定義

```
package org.jtron.attach;

public class ItronOACVException extends ItronCauseException {
        public ItronOACVException(String message,
           int resourceId, int functionCode, int errorCode);
}
```

■コンストラクタ

```
public ItronOACVException(String message,
        int resourceId, int functionCode, int errorCode);
```

【パラメータ】

String message	詳細メッセージ
int resourceId	オブジェクト ID(★)
int functionCode	ITRON 機能コード(★)
int errorCode	エラーコード(★)

【機能】

詳細メッセージを持つインスタンスを生成する。

【JTRON1.0 仕様との相違】

ITRON サービスコールエラーに対応した新クラスである。

4.2.2.10 サービスコール不正使用(ItronILUSEException)

```
org.jtron.attach.ItronCauseException
    |
    └──org.jtron.attach.ItronILUSEException
```

public class ItronILUSEException 標準仕様

サービスコール不正使用

■クラス定義

```
package org.jtron.attach;

public class ItronILUSEException extends ItronCauseException {
        public ItronILUSEException(String message,
           int resourceId, int functionCode, int errorCode);
}
```

■コンストラクタ

```
public ItronILUSEException(String message,
        int resourceId, int functionCode, int errorCode);
```

【パラメータ】

String message	詳細メッセージ
int resourceId	オブジェクト ID(★)
int functionCode	ITRON 機能コード(★)
int errorCode	エラーコード(★)

【機能】

詳細メッセージを持つインスタンスを生成する。

【JTRON1.0 仕様との相違】

ITRON サービスコールエラーに対応した新クラスである。

4.2.2.11 メモリ不足(ItronNOMEMException)

```
org.jtron.attach.ItronCauseException
    |
    └── org.jtron.attach.ItronNOMEMException
```

public class ItronNOMEMException 標準仕様

メモリ不足

■クラス定義

```
package org.jtron.attach;

public class ItronNOMEMException extends ItronCauseException {
        public ItronNOMEMException(String message,
           int resourceId, int functionCode, int errorCode);
}
```

■コンストラクタ

```
public ItronNOMEMException(String message,
        int resourceId, int functionCode, int errorCode);
```

【パラメータ】

String message	詳細メッセージ
int resourceId	オブジェクト ID(★)
int functionCode	ITRON 機能コード(★)
int errorCode	エラーコード(★)

【機能】

詳細メッセージを持つインスタンスを生成する。

【JTRON1.0 仕様との相違】

ITRON サービスコールエラーに対応した新クラスである。

4.2.2.12 ID 番号不足(ItronNOIDException)

```
org.jtron.attach.ItronCauseException
    |
    └── org.jtron.attach.ItronNOIDException
```

public class ItronNOIDException 標準仕様

ID 番号不足

■クラス定義

```
package org.jtron.attach;

public class ItronNOIDException extends ItronCauseException {
        public ItronNOIDException(String message,
           int resourceId, int functionCode, int errorCode);
}
```

■コンストラクタ

```
public ItronNOIDException(String message,
        int resourceId, int functionCode, int errorCode);
```

【パラメータ】

String message	詳細メッセージ
int resourceId	オブジェクト ID(★)
int functionCode	ITRON 機能コード(★)
int errorCode	エラーコード(★)

【機能】

詳細メッセージを持つインスタンスを生成する。

【JTRON1.0 仕様との相違】

ITRON サービスコールエラーに対応した新クラスである。

4.2.2.13 オブジェクト状態エラー(ItronOBJException)

```
org.jtron.attach.ItronCauseException
    |
    └── org.jtron.attach.ItronOBJException
```

public class ItronOBJException **標準仕様**

オブジェクト状態エラー

■クラス定義

```
package org.jtron.attach;

public class ItronOBJException extends ItronCauseException {
        public ItronOBJException(String message,
          int resourceId, int functionCode, int errorCode);
}
```

■コンストラクタ

```
public ItronOBJException(String message,
        int resourceId, int functionCode, int errorCode);
```

【パラメータ】

String	message	詳細メッセージ
int	resourceId	オブジェクト ID(★)
int	functionCode	ITRON 機能コード(★)
int	errorCode	エラーコード(★)

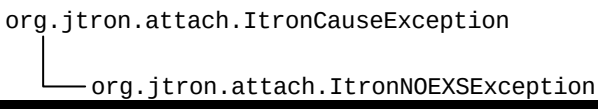
【機能】

詳細メッセージを持つインスタンスを生成する。

【JTRON1.0 仕様との相違】

ITRON サービスコールエラーに対応した新クラスである。

4.2.2.14 オブジェクト未生成(ItronNOEXSException)



public class ItronNOEXSException 標準仕様

オブジェクト未生成

■クラス定義

```
package org.jtron.attach;

public class ItronNOEXSException extends ItronCauseException {
        public ItronNOEXSException(String message,
           int resourceId, int functionCode, int errorCode);
}
```

■コンストラクタ

```
public ItronNOEXSException(String message,
        int resourceId, int functionCode, int errorCode);
```

【パラメータ】

String message	詳細メッセージ
int resourceId	オブジェクト ID(★)
int functionCode	ITRON 機能コード(★)
int errorCode	エラーコード(★)

【機能】

詳細メッセージを持つインスタンスを生成する。

【JTRON1.0 仕様との相違】

ITRON サービスコールエラーに対応した新クラスである。

4.2.2.15 キューイングオーバーフロー (ItronQOVRException)

```
org.jtron.attach.ItronCauseException
    |
    └── org.jtron.attach.ItronQOVRException
```

public class ItronQOVRException 標準仕様

キューイングオーバーフロー

■クラス定義

```
package org.jtron.attach;

public class ItronQOVRException extends ItronCauseException {
        public ItronQOVRException(String message,
           int resourceId, int functionCode, int errorCode);
}
```

■コンストラクタ

```
public ItronQOVRException(String message,
        int resourceId, int functionCode, int errorCode);
```

【パラメータ】

String message	詳細メッセージ
int resourceId	オブジェクト ID(★)
int functionCode	ITRON 機能コード(★)
int errorCode	エラーコード(★)

【機能】

詳細メッセージを持つインスタンスを生成する。

【JTRON1.0 仕様との相違】

ITRON サービスコールエラーに対応した新クラスである。

4.2.2.16 待ち状態の強制解除(ItronRLWAIException)

```
org.jtron.attach.ItronCauseException
    |
    └── org.jtron.attach.ItronRLWAIException
```

public class ItronRLWAIException 標準仕様

待ち状態の強制解除

■クラス定義

```
package org.jtron.attach;

public class ItronRLWAIException extends ItronCauseException {
        public ItronRLWAIException(String message,
           int resourceId, int functionCode, int errorCode);
}
```

■コンストラクタ

```
public ItronRLWAIException(String message,
        int resourceId, int functionCode, int errorCode);
```

【パラメータ】

String	message	詳細メッセージ
int	resourceId	オブジェクト ID(★)
int	functionCode	ITRON 機能コード(★)
int	errorCode	エラーコード(★)

【機能】

詳細メッセージを持つインスタンスを生成する。

【JTRON1.0 仕様との相違】

ITRON サービスコールエラーに対応した新クラスである。

4.2.2.17 ポーリング失敗またはタイムアウト (ItronTMOUTException)

```
org.jtron.attach.ItronCauseException
    │
    └── org.jtron.attach.ItronTMOUTException
```

public class ItronTMOUTException 標準仕様

ポーリング失敗またはタイムアウト

■クラス定義

```
package org.jtron.attach;

public class ItronTMOUTException extends ItronCauseException {
        public ItronTMOUTException(String message,
           int resourceId, int functionCode, int errorCode);
}
```

■コンストラクタ

```
public ItronTMOUTException(String message,
        int resourceId, int functionCode, int errorCode);
```

【パラメータ】

String	message	詳細メッセージ
int	resourceId	オブジェクト ID(★)
int	functionCode	ITRON 機能コード(★)
int	errorCode	エラーコード(★)

【機能】

詳細メッセージを持つインスタンスを生成する。

【JTRON1.0 仕様との相違】

ITRON サービスコールエラーに対応した新クラスである。

4.2.2.18 待ちオブジェクトの削除(ItronDLTException)

```
org.jtron.attach.ItronCauseException
    |
    └──org.jtron.attach.ItronDLTException
```

public class ItronDLTException **標準仕様**

待ちオブジェクトの削除

■クラス定義

```
package org.jtron.attach;

public class ItronDLTException extends ItronCauseException {
        public ItronDLTException(String message,
          int resourceId, int functionCode, int errorCode);
}
```

■コンストラクタ

```
public ItronDLTException(String message,
        int resourceId, int functionCode, int errorCode);
```

【パラメータ】

String	message	詳細メッセージ
int	resourceId	オブジェクト ID(★)
int	functionCode	ITRON 機能コード(★)
int	errorCode	エラーコード(★)

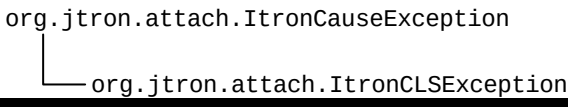
【機能】

詳細メッセージを持つインスタンスを生成する。

【JTRON1.0 仕様との相違】

ITRON サービスコールエラーに対応した新クラスである。

4.2.2.19 待ちオブジェクトの状態変化(ItronCLSException)



public class ItronCLSException　標準仕様

待ちオブジェクトの状態変化

■クラス定義

```
package org.jtron.attach;

public class ItronCLSException extends ItronCauseException {
        public ItronCLSException(String message,
          int resourceId, int functionCode, int errorCode);
}
```

■コンストラクタ

```
public ItronCLSException(String message,
        int resourceId, int functionCode, int errorCode);
```

【パラメータ】

String message	詳細メッセージ
int resourceId	オブジェクト ID(★)
int functionCode	ITRON 機能コード(★)
int errorCode	エラーコード(★)

【機能】

詳細メッセージを持つインスタンスを生成する。

【JTRON1.0 仕様との相違】

ITRON サービスコールエラーに対応した新クラスである。

4.2.2.20 ノンブロッキング受付け(ItronWBLKException)

```
org.jtron.attach.ItronCauseException
    |
    └──org.jtron.attach.ItronWBLKException
```

public class ItronWBLKException 標準仕様

ノンブロッキング受付け

■クラス定義

```
package org.jtron.attach;

public class ItronWBLKException extends ItronCauseException {
        public ItronWBLKException(String message,
           int resourceId, int functionCode, int errorCode);
}
```

■コンストラクタ

```
public ItronWBLKException(String message,
        int resourceId, int functionCode, int errorCode);
```

【パラメータ】

String message	詳細メッセージ
int resourceId	オブジェクト ID(★)
int functionCode	ITRON 機能コード(★)
int errorCode	エラーコード(★)

【機能】

詳細メッセージを持つインスタンスを生成する。

【JTRON1.0 仕様との相違】

ITRON サービスコールエラーに対応した新クラスである。

4.2.2.21 バッファオーバーフロー(ItronBOVRException)

```
org.jtron.attach.ItronCauseException
    |
    └── org.jtron.attach.ItronBOVRException
```

public class ItronBOVRException 標準仕様

バッファオーバーフロー

■クラス定義

```
package org.jtron.attach;

public class ItronBOVRException extends ItronCauseException {
        public ItronBOVRException(String message,
           int resourceId, int functionCode, int errorCode);
}
```

■コンストラクタ

```
public ItronBOVRException(String message,
        int resourceId, int functionCode, int errorCode);
```

【パラメータ】

String message	詳細メッセージ
int resourceId	オブジェクト ID(★)
int functionCode	ITRON 機能コード(★)
int errorCode	エラーコード(★)

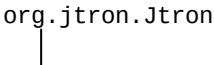
【機能】

詳細メッセージを持つインスタンスを生成する。

【JTRON1.0 仕様との相違】

ITRON サービスコールエラーに対応した新クラスである。

4.2.3 JTRON による例外クラス(JtronCauseException)



public class JtronCauseException 標準仕様

JTRON による例外クラス

ITRON サービスコールエラー以外の場合に例外が発行される。
一般的なエラーの他、Java 実行環境の動作に必要なタスク、セマフォなどをユーザが操作しようとした場合に起きるエラーも含む。

■クラス定義

```
package org.jtron.attach;

public class JtronCauseException
                        extends org.jtron.JtronException {
        // 実装定義
}
```

【JTRON1.0 仕様との相違】

μITRON 仕様 OS のサービスコールエラー以外の JTRON に関連する新クラスである。

【補足説明】

Java 実行環境利用のカーネル資源を判別できない実装も考えられ、実装の有無はベンダ依存とする。

4.2.4 メモリ操作クラス(ItronMemory)

```
java.lang.Object
    |
    +--- org.jtron.ItronMemory
```

public class ItronMemory **標準仕様**

Java プログラムから、メモリを操作するための基底クラス。

ITRON にはメモリを直接操作するサービスコールが多数存在するが、Java では特定のメモリ領域の内容を操作することはもちろん、そのアドレス値を得ることも不可能である。

そこで Java プログラムからこれらのサービスコール機能を実現するためには、Java でメモリ内容を操作するための仕組みが必要になる。JTRON ではメモリ操作クラスでこれを実現する。

メモリ操作クラスは、Java プログラムでメモリアドレスを意識することなくメモリ内容を操作することができるように設計されている。メモリ操作クラスを使用することで、メモリ内容の書き換えがストリームないしはランダムアクセスファイルに対する入出力と同様な方法で行うことが可能になる。

メモリ操作クラスではメモリ領域の範囲を管理している。この領域外のアクセスを行なった場合には例外が発生する。これにより決められた範囲のメモリにしかアクセスできないので安全性が維持できる。また、より安全性を高められるようメモリ領域を書き込み禁止に設定することも可能である。

メモリ内容の読み書きは、ITRON で定義している B や H、W 型などのデータを読み書きするメソッドを呼び出すことで行う。読み書きする位置（アクセス位置）を領域先頭からのオフセットとして持っており、インスタンス生成直後はこれが先頭になっている。すべての読み書きするメソッドではそのメソッド呼び出し後、読み書きしたサイズ分だけアクセス位置の値が増加する。これを利用して先頭から順に読み書きすることもできるし、アクセス位置を直接指定して読み書きすることもできる。また単にアクセス位置だけを変更することもできる。

読み書き中にエラー（例外）が発生した場合、アクセス位置や領域の値がどうなるかは実装依存である。

扱うデータ型のサイズ、エンディアン、バイト並びなどはすべてITRON でのそれに従う。従って先頭から順に読み書きせず途中のメンバをオフセット指定でアクセスする場合、そのオフセットを定数にすると移植性を失う。

メモリ操作クラスのインスタンスを生成するには、以下の２つのいずれかの方法を選択する。

(1) 新規にメモリ領域を確保する。
(2) すでに確保されているメモリ領域に対応させる。

(1)は Java プログラムで直接行うことができ、生成されたインスタンスに必要な値を書き込んでメールボックスやデータキューなどにより送信するのに用いることができる。この場合、領域の解放はこのインスタンスが消滅するときに自動的に行なわれるが、プログラムが明示的に行うこともできる。

(2)は Java プログラムで直接行うことができない。これはメールボックスやデータキューなどからメッセージを受信する場合にメソッドのリターン値として返されるものを使うことができるだけである。この場合、実際のメモリ領域は送信側で確保される。送信側は Java とは無関係の ITRON タスクでもよい。メモリ操作クラスではメモリ領域の確保・解放には一切タッチしない。

メモリ操作クラスで管理するメモリ領域は、(1)の場合には Java のガーベジコレクションの対象にならないことを保証する。そのため受信側が ITRON タスクであっても特別な処理を必要としない。(2)の場合は送信側がどのような領域を送信するかに依存しており、受信側で何もしないことを保証するのみである。

メモリ操作クラスはこのまま使うことができるが、以下のクラスがこれを継承しているので同様の方法で内容を操作することができる。

・ランデブアタッチクラス
・固定長メモリブロックアタッチクラス
・可変長メモリブロックアタッチクラス

■クラス定義

```
package org.jtron.attach;

public class ItronMemory {
        public ItronMemory(int length);

        public void release();
        public int getLength();

        public void disableWrite();
        public void enableWrite();
        public boolean isWriteable();

        public void seek(int offset);
        public void skipBytes(int n);
        public int getOffset();

        public byte readB();
        public byte readB(int offset);
        public short readH();
        public short readH(int offset);
        public int readW();
        public int readW(int offset);
        public long readD();
        public long readD(int offset);
        public short readUB();
        public short readUB(int offset);
        public int readUH();
        public int readUH(int offset);
        public long readUW();
        public long readUW(int offset);

        public int read(byte[] b);
        public int read(int offset, byte[] b);
        public int read(byte[] b, int bOff, int len);
```

```
        public int read(int offset, byte[] b, int bOff, int len);

        public void writeB(byte value);
        public void writeB(int offset, byte value);
        public void writeH(short value);
        public void writeH(int offset, short value);
        public void writeW(int value);
        public void writeW(int offset, int value);
        public void writeD(long value);
        public void writeD(int offset, long value);

        public int write(byte[] b);
        public int write(int offset, byte[] b);
        public int write(byte[] b, int bOff, int len);
        public int write(int offset,
                         byte[] b, int bOff, int len);
}
```

■コンストラクタ

public ItronMemory(int length) throws JtronException;

【パラメータ】

int	length	実領域のサイズ(★)

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたはJTRON による例外クラス)

【機能】

指定したサイズ(length)の実領域を持ったインスタンスを生成する。生成直後の状態は以下の通り。

- アクセス位置：0（先頭）
- 書き込み可否：true（可能）

・実領域：length バイト確保ずみ（解放可能）
実領域がどこからどのようにして確保されるかは実装定義である。

■メソッド

```
public void release() throws JtronException;
```

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたは JTRON による例外クラス)

【機能】

実領域を解放する。領域をすでに解放済みの場合は何もしない。

```
public int getLength() throws JtronCauseException;
```

【戻り値】

int	実領域の長さ(★)

【例外】

JtronCauseException	JTRON による例外クラス(実領域が無効)

【機能】

実領域の長さを返す。

```
public void disableWrite() throws JtronCauseException;
```

【例外】

JtronCauseException	JTRON による例外クラス(実領域が無効)

【機能】

領域に対する書き込みを禁止する。

```
public void enableWrite() throws JtronCauseException;
```

【例外】

JtronCauseException	JTRON による例外クラス(実領域が無効)

【機能】

領域に対する書き込みを許可する。

```
public boolean isWriteable() throws JtronCauseException;
```

【戻り値】

boolean	領域に対する書き込み許可状態

【例外】

JtronCauseException	JTRON による例外クラス(実領域が無効)

【機能】

領域が書き込み許可になっているかどうかを返す。

```
public void seek(int offset) throws JtronCauseException;
```

【パラメータ】

int	offset	先頭からの開始位置(★)

【例外】

JtronCauseException	JTRON による例外クラス(実領域が無効またはアクセス位置が終端を越えて指定された)

【機能】

メモリアクセス位置を先頭から offset バイトに設定する。

```
public void skipBytes(int n) throws JtronCauseException;
```

【パラメータ】

int	n	スキップバイト数(★)

【例外】

JtronCauseException	JTRON による例外クラス(実領域が無効またはアクセス位置が終端を越えて指定された)

【機能】

メモリアクセス位置を現在位置＋n バイトに設定する。

```
public int getOffset() throws JtronCauseException;
```

【戻り値】

int	現在のメモリアクセス位置(★)

【例外】

JtronCauseException	JTRON による例外クラス(実領域が無効)

【機能】

現在のメモリアクセス位置を返す。

【補足説明】

アクセス位置がメモリ領域の終端に達している場合は、実領域の長さを返す。

```
public byte readB() throws JtronCauseException;
```

【戻り値】

byte 符号付き 8 ビット整数データ

【例外】

JtronCauseException JTRON による例外クラス(実領域が無効またはアクセス位置が実領域の終端を越えた)

【機能】

現在位置のデータを、μITRON 仕様のデータ型 B(符号付き 8 ビット整数)にて取得する。

```
public byte readB(int offset) throws JtronCauseException;
```

【パラメータ】

int offset 先頭からの開始位置(★)

【戻り値】

byte 符号付き 8 ビット整数データ

【例外】

JtronCauseException JTRON による例外クラス(実領域が無効またはアクセス位置が実領域の終端を越えた)

【機能】

指定された offset 位置のデータを、μITRON 仕様のデータ型 B(符号付き 8 ビット整数)にて取得する。

```
public short readH() throws JtronCauseException;
```

【戻り値】

short 符号付き 16 ビット整数データ

【例外】

JtronCauseException JTRON による例外クラス(実領域

が無効またはアクセス位置が実領域の終端を越えた)

【機能】

現在位置のデータを、μITRON 仕様のデータ型 H(符号付き 16 ビット整数)にて取得する。

```
public short readH(int offset)
                        throws JtronCauseException;
```

【パラメータ】

int	offset	先頭からの開始位置(★)

【戻り値】

short	符号付き 16 ビット整数データ

【例外】

JtronCauseException	JTRON による例外クラス(実領域が無効またはアクセス位置が実領域の終端を越えた)

【機能】

指定された offset 位置のデータを、μITRON 仕様のデータ型 H(符号付き 16 ビット整数)にて取得する。

```
public int readW() throws JtronCauseException;
```

【戻り値】

int	符号付き 32 ビット整数データ

【例外】

JtronCauseException	JTRON による例外クラス(実領域が無効またはアクセス位置が実領域の終端を越えた)

【機能】

現在位置のデータを、μITRON 仕様のデータ型 W(符号付き 32 ビット整数)にて取得する。

```
public int readW(int offset) throws JtronCauseException;
```

【パラメータ】

int	offset	先頭からの開始位置(★)

【戻り値】

int	符号付き 32 ビット整数データ

【例外】

JtronCauseException	JTRON による例外クラス(実領域が無効またはアクセス位置が実領域の終端を越えた)

【機能】

指定された offset 位置のデータを、μITRON 仕様のデータ型 W(符号付き 32 ビット整数)にて取得する。

```
public long readD() throws JtronCauseException;
```

【戻り値】

long	符号付き 64 ビット整数データ

【例外】

JtronCauseException	JTRON による例外クラス(実領域が無効またはアクセス位置が実領域の終端を越えた)

【機能】

現在位置のデータを、μITRON 仕様のデータ型 D(符号付き 64 ビット整数)にて取得する。

```
public long readD(int offset) throws JtronCauseException;
```

【パラメータ】

int	offset	先頭からの開始位置(★)

【戻り値】

long	符号付き 64 ビット整数データ

【例外】

JtronCauseException	JTRON による例外クラス(実領域が無効またはアクセス位置が実領域の終端を越えた)

【機能】

指定された offset 位置のデータを、μITRON 仕様のデータ型 D(符号付き 64 ビット整数)にて取得する。

```
public short readUB() throws JtronCauseException;
```

【戻り値】

short	符号無し 8 ビット整数データ

【例外】

JtronCauseException	JTRON による例外クラス(実領域が無効またはアクセス位置が実領域の終端を越えた)

【機能】

現在位置のデータを、μITRON 仕様のデータ型 UB(符号無し 8 ビット整数)にて取得する。

```
public short readUB(int offset)
                        throws JtronCauseException;
```

【パラメータ】

int	offset	先頭からの開始位置(★)

【戻り値】

short	符号無し 8 ビット整数データ

【例外】

JtronCauseException	JTRON による例外クラス(実領域が無効またはアクセス位置が実領域の終端を越えた)

【機能】

指定された offset 位置のデータを、μITRON 仕様のデータ型 UB(符号無し 8 ビット整数)にて取得する。

```
public int readUH() throws JtronCauseException;
```

【戻り値】

int	符号無し 16 ビット整数データ

【例外】

JtronCauseException	JTRON による例外クラス(実領域が無効またはアクセス位置が実領域の終端を越えた)

【機能】

現在位置のデータを、μITRON 仕様のデータ型 UH(符号無し 16 ビット整数)にて取得する。

```
public int readUH(int offset) throws JtronCauseException;
```

【パラメータ】

int	offset	先頭からの開始位置(★)

【戻り値】

int	符号無し 16 ビット整数データ

【例外】

JtronCauseException	JTRON による例外クラス(実領域が無効またはアクセス位置が実領域の終端を越えた)

【機能】

指定された offset 位置のデータを、μITRON 仕様のデータ型 UH(符号無し 16 ビット整数)にて取得する。

```
public long readUW() throws JtronCauseException;
```

【戻り値】

long	符号無し 32 ビット整数データ

【例外】

JtronCauseException	JTRON による例外クラス(実領域が無効またはアクセス位置が実領域の終端を越えた)

【機能】

現在位置のデータを、μITRON 仕様のデータ型 UW(符号無し 32 ビット整数)にて取得する。

```
public long readUW(int offset)
                        throws JtronCauseException;
```

【パラメータ】

int	offset	先頭からの開始位置(★)

【戻り値】

long	符号無し 32 ビット整数データ

【例外】

JtronCauseException	JTRON による例外クラス(実領域

が無効またはアクセス位置が実領域の終端を越えた)

【機能】

指定された offset 位置のデータを、μ ITRON 仕様のデータ型 UW(符号無し 32 ビット整数)にて取得する。

```
public int read(byte[] b) throws JtronCauseException;
```

【パラメータ】

byte[]	b	データを読む込むバッファ

【戻り値】

int	バッファに読み込んだバイト数を返す。実領域の終端に達し、データが無い場合は-1 を返す。(★)

【例外】

JtronCauseException	JTRON による例外クラス(実領域が無効)

【機能】

現在位置からバイト配列に最大 b.length バイトのデータを読み込む。

```
public int read(int offset, byte[] b)
                        throws JtronCauseException;
```

【パラメータ】

int	offset	先頭からの開始位置(★)
byte[]	b	データを読む込むバッファ

【戻り値】

int	バッファに読み込んだバイト数を返す。実領域の終端に達し、データが無い場合は-1 を返す。(★)

【例外】

JtronCauseException	JTRON による例外クラス(実領域が無効)

【機能】

指定された offset 位置からバイト配列に最大 b.length バイトのデータを読み込む。

```
public int read(byte[] b, int bOff, int len)
                        throws JtronCauseException;
```

【パラメータ】

byte[]	b	データを読む込むバッファ
int	bOff	格納開始オフセット(★)
int	len	読む込む最大バイト数(★)

【戻り値】

int	バッファに読み込んだバイト数を返す。ファイルの終端に達し、データが無い場合は-1 を返す。(★)

【例外】

JtronCauseException	JTRON による例外クラス(実領域が無効)

【機能】

現在位置からバイト配列の指定された格納開始位置 bOff へ最大 len バイトのデータを読み込む。

```
public int read(int offset, byte[] b, int bOff, int len)
                        throws JtronCauseException;
```

【パラメータ】

int	offset	先頭からの開始位置(★)

byte[]	b	データを読む込むバッファ
int	bOff	格納開始オフセット(★)
int	len	読む込む最大バイト数(★)

【戻り値】

int	バッファに読み込んだバイト数を返す。ファイルの終端に達し、データが無い場合は-1 を返す。(★)

【例外】

JtronCauseException	JTRON による例外クラス(実領域が無効)

【機能】

指定された offset 位置からバイト配列の指定された格納開始位置 bOff へ最大 len バイトのデータを読み込む。

```
public void writeB(byte value)
                        throws JtronCauseException;
```

【パラメータ】

byte	value	符号付き 8 ビット整数データ

【例外】

JtronCauseException	JTRON による例外クラス(書き込み禁止の場合、実領域が無効またはアクセス位置が実領域の終端を越えた)

【機能】

現在位置へ、μITRON 仕様のデータ型 B(符号付き 8 ビット整数)にて指定されたデータを格納する。

```
public void writeB(int offset, byte value)
                        throws JtronCauseException;
```

【パラメータ】

int	offset	先頭からの開始位置(★)
byte	value	符号付き 8 ビット整数データ

【例外】

JtronCauseException	JTRON による例外クラス(書き込み禁止の場合、実領域が無効またはアクセス位置が実領域の終端を越えた)

【機能】

指定された offset 位置へ、μITRON 仕様のデータ型 B(符号付き 8 ビット整数)にて指定されたデータを格納する。

```
public void writeH(short value)
                        throws JtronCauseException;
```

【パラメータ】

short	value	符号付き 16 ビット整数データ

【例外】

JtronCauseException	JTRON による例外クラス(書き込み禁止の場合、実領域が無効またはアクセス位置が実領域の終端を越えた)

【機能】

現在位置へ、μITRON 仕様のデータ型 H(符号付き 16 ビット整数)にて指定されたデータを格納する。

```
public void writeH(int offset, short value)
                        throws JtronCauseException;
```

【パラメータ】

int	offset	先頭からの開始位置(★)
short	value	符号付き 16 ビット整数データ

【例外】

JtronCauseException	JTRON による例外クラス(書き込み禁止の場合、実領域が無効またはアクセス位置が実領域の終端を越えた)

【機能】

指定された offset 位置へ、μITRON 仕様のデータ型 H(符号付き 16 ビット整数)にて指定されたデータを格納する。

```
public void writeW( int value) throws JtronCauseException;
```

【パラメータ】

int	value	符号付き 32 ビット整数データ

【例外】

JtronCauseException	JTRON による例外クラス(書き込み禁止の場合、実領域が無効またはアクセス位置が実領域の終端を越えた)

【機能】

現在位置へ、μ ITRON 仕様のデータ型 W(符号付き 32 ビット整数)にて指定されたデータを格納する。

```
public void writeW(int offset, int value)
                        throws JtronCauseException;
```

【パラメータ】

int	offset	先頭からの開始位置(★)
int	value	符号付き 32 ビット整数データ

【例外】

JtronCauseException	JTRON による例外クラス(実領域が無効またはアクセス位置が実領域の終端を越えた)

【機能】

指定された offset 位置へ、μITRON 仕様のデータ型 W(符号付き 32 ビット整数)にて指定されたデータを格納する。

```
public void writeD(long value)
                        throws JtronCauseException;
```

【パラメータ】

long	value	符号付き 64 ビット整数データ

【例外】

JtronCauseException	JTRON による例外クラス(実領域が無効またはアクセス位置が実領域の終端を越えた)

【機能】

現在位置へ、μITRON 仕様のデータ型 D(符号付き 64 ビット整数)にて指定されたデータを格納する。

```
public void writeD(int offset, long value)
                        throws JtronCauseException;
```

【パラメータ】

int	offset	先頭からの開始位置(★)
long	value	符号付き 64 ビット整数データ

【例外】

JtronCauseException	JTRON による例外クラス(実領域が無効またはアクセス位置が実領域の終端を越えた)

【機能】

指定された offset 位置へ、μITRON 仕様のデータ型 D(符号付き 64 ビット整数)にて指定されたデータを格納する。

```
public int write(byte[] b) throws JtronCauseException;
```

【パラメータ】

byte[]	b	書き込むバイト配列データ

【戻り値】

int	書き込んだバイト数を返す。既に終端に達し、書き込めない場合は-1 を返す。(★)

【例外】

JtronCauseException	JTRON による例外クラス(実領域が無効)

【機能】

現在位置へ、バイト配列のデータを最大 b.length バイト書き込む。

```
public int write(int offset, byte[] b)
                        throws JtronCauseException;
```

【パラメータ】

int	offset	先頭からの開始位置(★)
byte[]	b	書き込むバイト配列データ

【戻り値】

int	書き込んだバイト数を返す。既に終端に達し、書き込めない場合は-1 を返す。(★)

【例外】

JtronCauseException	JTRON による例外クラス(実領域が無効)

【機能】

指定された offset 位置へ、バイト配列のデータを最大 b.length バイト書き込む。

```
public int write(int byte[], int bOff, int len)
                        throws JtronCauseException;
```

【パラメータ】

byte[]	b	書き込むバイト配列データ
int	bOff	書き込むバイト配列の開始位置(★)
int	len	書き込むデータの長さ(★)

【戻り値】

int	書き込んだバイト数を返す。既に終端に達し、書き込めない場合は-1 を返す。(★)

【例外】

JtronCauseException	JTRON による例外クラス(実領域が無効)

【機能】

現在位置へ、バイト配列の指定された開始位置 bOff から最大 len バイトのデータを書き込む。

```
public int write(int offset, byte[] b, int bOff, int len)
                        throws JtronCauseException;
```

【パラメータ】

int	offset	先頭からの開始位置(★)
byte[]	b	書き込むバイト配列データ
int	bOff	書き込むバイト配列の開始位置(★)
int	len	書き込むデータの長さ(★)

【戻り値】

int	書き込んだバイト数を返す。既に終端に達し、書き込めない場合は-1 を返す。(★)

【例外】

JtronCauseException	JTRON による例外クラス(実領域が無効)

【機能】

指定された offset 位置へ、バイト配列の指定された開始位置 bOff から最大 len バイトのデータを書き込む。

【JTRON1.0 仕様との相違および仕様決定の理由】

・free()を release()と改名した。可変長メモリブロックと固定長メモリブロックはメモリ操作クラスを継承しているのに領域を解放するメソッドの名前が継承元クラスと異なり free()でなく release()になっていた。このままでは本来の解放メソッド release()でなく継承元の解放メソッド free()を呼び出すことができてしまう。しかしこれでは解放できないので名前を合わせることにより誤った呼び出しをできないようにした。

・アクセス位置を返す getOffset()、アクセス位置を相対値で変更する skipBytes()を追加した。これによりライブラリなどで読み書きした後位置を戻しておくことができるようになる。また先頭からのバイト数を計算する必要がなくなることにより変更に強くなる。

・読み書きメソッド名に Java のデータ型名がついていたのをすべてμITRON4.0 仕様のデータ型名に変更した。これはその動作がμITRON4.0 仕様のデータ型に従って行なわれることを示すためであり、その説明も明記した。

*xxx*Byte() → *xxx*B()
*xxx*Short() → *xxx*H()
*xxx*Int() → *xxx*W()
*xxx*Long() → *xxx*D()

・μITRON4.0 仕様の符号無しデータを読み込むメソッドを追加した。

Java にはない符号無しの値を扱うとき、いままではキャストしたりマスクしていたのが不便だったため。符号の影響を避けるため、値域の大きいデータ型の値を返すものとした。但し、値域の大きいデータ型の値を返す事ができない readUD メソッドは用意しない。

・バイト配列読み書きメソッドでアクセスする長さが実領域の終端を越えた時の動作を定義した。この動作は Java コアクラスのストリームアクセスメソッドに合わせたものである。また引数についても Java コアクラスに似せて変更した。

・実領域の取り扱いについて、GC 対象かどうかは実装依存としていたが、GC 対象でないものと変更した。

4.2.5 タスク

ITRON のタスクを操作するために以下のクラス群がある。

- タスクアタッチクラス
- タスク生成情報クラス
- 簡易タスク状態クラス
- タスク状態クラス
- タスク例外処理定義情報クラス
- タスク例外状態クラス
- オーバーランハンドラ状態クラス

タスクアタッチクラスは ITRON のタスクを表現する。

このクラスのインスタンスを生成することによりタスクの生成ができるほか、生成済み ITRON タスクに対応するインスタンス、特定の Java スレッド相当の ITRON タスクに対応するインスタンスを得ることもできる。

このインスタンスを対象にして、ITRON のタスク管理機能、タスク付属同期機能、タスク例外処理機能、オーバーランハンドラの機能を呼び出す。

4.2.5.1 タスクアタッチクラス(Task)

```
java.lang.Object
    |
    +--org.jtron.attach.Task
```

public class Task **標準仕様**

タスクアタッチクラス

ITRON タスクを操作するためのクラスである。

■クラス定義

```
package org.jtron.attach;

public class Task {
        public Task(int tskid);
        public Task(Thread thread);
        public Task(int tskid, T_CTSK pk_ctsk);
        public Task(T_CTSK pk_ctsk);

        public int getId();
        public static Task currentTask();

        public void delete();
        public void activate();
        public int cancelActivate();
        public void start(int stacd);
        public void terminate();
        public void changePriority(int tskpri);
        public int getPriority();
        public T_RTSK refer();
        public T_RTST referSimple();

        public static void sleep();
        public static void sleep(int tmout);
        public void wakeup();
```

```
        public int cancelWakeup();
        public void releaseWait();
        public void suspend();
        public void resume();
        public void forceResume();
        public static void delay(int dlytim);

        public void defineTaskException(T_DTEX pk_dtex);
        public void raiseTaskException(int rasptn);
        public T_RTEX referTaskException();

        public void startOverrunHandler(int ovrtim);
        public void stopOverrunHandler();
        public T_ROVR referOverrunHandler();
}
```

■コンストラクタ

public Task(int tskid) throws JtronException;

【パラメータ】

int	tskid	タスク ID(★)

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたは JTRON による例外クラス)

【機能】

タスク ID を指定して既存のタスクに接続するインスタンスを生成する。

public Task(Thread thread) throws JtronCauseException;

【パラメータ】

Thread thread　　スレッドクラスのインスタンス

【例外】

JtronCauseException JTRON による例外クラス(メモリ不足など)

【機能】

スレッドを指定して既存のタスクに接続するインスタンスを生成する。

```
public Task(int tskid, T_CTSK pk_ctsk)
                        throws ItronCauseException;
```

【パラメータ】

int　tskid　　タスク ID(★)
T_CTSK　pk_ctsk　　タスク生成情報クラス

【例外】

ItronCauseException ITRON による例外クラス

【機能】

cre_tsk サービスコール呼び出しに相当する。
タスクを生成し、接続するインスタンスを生成する。

```
public Task(T_CTSK pk_ctsk)
                        throws ItronCauseException;
```

【パラメータ】

T_CTSK　pk_ctsk　　タスク生成情報クラス

【例外】

ItronCauseException ITRON による例外クラス

【機能】

acre_tsk サービスコール呼び出しに相当する。
タスクを生成し、接続するインスタンスを生成する。

■static メソッド

```
public static Task currentTask()
                        throws JtronCauseException;
```

【戻り値】

Task	タスククラスのインスタンス

【例外】

JtronCauseException	JTRON による例外クラス(メモリ不足など)

【機能】

カレントスレッドに対するタスクに接続するインスタンスを返す。

```
public static void sleep() throws ItronCauseException;
```

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

slp_tsk サービスコール呼び出しに相当する。

```
public static void sleep(int timeout)
                        throws ItronCauseException;
```

【パラメータ】

int	timeout	タイムアウト時間(単位:ms)(★)

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

tslp_tsk サービスコール呼び出しに相当する。

```
public static void delay(int dlytim)
                        throws ItronCauseException;
```

【パラメータ】

int	dlytim	遅延時間(単位:ms)(★)

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

dly_tsk サービスコール呼び出しに相当する。

■メソッド

```
public int getId();
```

【戻り値】

int	タスク ID(★)

【機能】

接続しているタスクのタスク ID を返す。

```
public void delete() throws ItronCauseException;
```

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

del_tsk サービスコール呼び出しに相当する。

```
public void activate() throws ItronCauseException;
```

【例外】

ItronCauseException ITRON による例外クラス

【機能】

act_tsk サービスコール呼び出しに相当する。

```
public int cancelActivate() throws ItronCauseException;
```

【戻り値】

int	キューイングされた起動要求の回数 (正の値または 0)(★)

【例外】

ItronCauseException ITRON による例外クラス

【機能】

can_act サービスコール呼び出しに相当する。

```
public void start(int stacd) throws ItronCauseException;
```

【パラメータ】

int	stacd	スタートコード(★)

【例外】

ItronCauseException ITRON による例外クラス

【機能】

sta_tsk サービスコール呼び出しに相当する。

```
public void terminate() throws ItronCauseException;
```

【例外】

ItronCauseException ITRON による例外クラス

【機能】

ter_tsk サービスコール呼び出しに相当する。

```
public void changePriority(int tskpri)
                        throws ItronCauseException;
```

【パラメータ】

int	tskpri	タスク優先度(★)

【例外】

ItronCauseException ITRON による例外クラス

【機能】

chg_pri サービスコール呼び出しに相当する。

```
public int getPriority() throws ItronCauseException;
```

【戻り値】

int	タスク優先度(★)

【例外】

ItronCauseException ITRON による例外クラス

【機能】

get_pri サービスコール呼び出しに相当する。

```
public T_RTSK refer() throws JtronException;
```

【戻り値】

T_RTSK	タスク状態のパケット形式クラス

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたは JTRON による例外クラス)

【機能】

ref_tsk サービスコール呼び出しに相当する。

```
public T_RTST referSimple() throws JtronException;
```

【戻り値】

T_RTST	タスク状態(簡易版)のパケット形式クラス

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたは JTRON による例外クラス)

【機能】

ref_tst サービスコール呼び出しに相当する。

```
public void wakeup() throws ItronCauseException;
```

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

wup_tsk サービスコール呼び出しに相当する。

```
public int cancelWakeup() throws ItronCauseException;
```

【戻り値】

int	キューイングされていた起床要求の回数(正の値または 0)(★)

【例外】

ItronCauseException ITRON による例外クラス

【機能】

can_wup サービスコール呼び出しに相当する。

```
public void releaseWait() throws ItronCauseException;
```

【例外】

ItronCauseException ITRON による例外クラス

【機能】

rel_wai サービスコール呼び出しに相当する。

```
public void suspend() throws ItronCauseException;
```

【例外】

ItronCauseException ITRON による例外クラス

【機能】

sus_tsk サービスコール呼び出しに相当する。

```
public void resume() throws ItronCauseException;
```

【例外】

ItronCauseException ITRON による例外クラス

【機能】

rsm_tsk サービスコール呼び出しに相当する。

```
public void forceResume() throws ItronCauseException;
```

【例外】

ItronCauseException ITRON による例外クラス

【機能】

frsm_tsk サービスコール呼び出しに相当する。

```
public void defineTaskException(T_DTEX pk_dtex)
                        throws ItronCauseException;
```

【パラメータ】

T_DTEX	pk_dtex	タスク例外処理ルーチンのパケット形式クラス

【例外】

ItronCauseException ITRON による例外クラス

【機能】

def_tex サービスコール呼び出しに相当する。
引数に null を指定すると解除する。

```
public void raiseTaskException(int rasptn)
                        throws ItronCauseException;
```

【パラメータ】

int	rasptn	要求するタスク例外処理のタスク例外要因(★)

【例外】

ItronCauseException ITRON による例外クラス

【機能】

ras_tex サービスコール呼び出しに相当する。

```
public T_RTEX referTaskException()
                        throws JtronException;
```

【戻り値】

T_RTEX	タスク例外処理状態のパケット形式クラス

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたはJTRON による例外クラス)

【機能】

ref_tex サービスコール呼び出しに相当する。

```
public void startOverrunHandler(int ovrtim)
                        throws ItronCauseException;
```

【パラメータ】

int	ovrtim	設定するタスクの上限プロセッサ時間(単位:ms)(★)

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

sta_ovr サービスコール呼び出しに相当する。

```
public void stopOverrunHandler()
                        throws ItronCauseException;
```

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

stp_ovr サービスコール呼び出しに相当する。

```
public T_ROVR referOverrunHandler()
                        throws JtronException;
```

【戻り値】

T_ROVR	オーバーランハンドラ状況を返すパケットクラス

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたはJTRON による例外クラス)

【機能】

ref_ovr サービスコール呼び出しに相当する。

【JTRON1.0 仕様との相違および仕様決定の理由】

・タスクを生成する機能が抜けていたのでそれを行うコンストラクタを追加した。

・start()に起動コード引数が抜けていたので追加した。

・μITRON4.0 仕様の新設サービスコールに対応した activate()、cancelActivate()、getPriority()、referSimple()を追加した。

・referStatus()は名称が冗長なので refer()に変更した。

・２つの sleep()は両方ともインスタンスメソッドだったが、自タスクにしか作用しないことから static メソッドに変更した。

・dly_tsk サービスコール相当のメソッドが抜けていたので delay()として追加した。

・μITRON4.0 仕様の新機能であるタスク例外処理機能、オーバーラン

ハンドラの機能に対応したメソッドを追加した。ただしタスク例外処理機能のうち自タスクにのみ作用する機能は、Java 実行環境に悪影響をもたらす可能性を考慮し盛り込まなかった。

・ext_tsk サービスコール、exd_tsk サービスコールは自タスクにのみ作用しないため、呼び出すと Java 実行環境に悪影響をもたらす可能性がある。そのためそれらに相当するメソッドは追加しないことにした。

・rotateReadyQueue()は特定タスクに作用する機能でないため、新設のカーネルアタッチクラスに移動した。

・getTaskId()はμITRON4.0 仕様で get_tid サービスコールの役割が変更されたため、新設のカーネルアタッチクラスに移動した。

4.2.5.2 タスク生成情報クラス(T_CTSK)

```
java.lang.Object
    |
    └── org.jtron.attach.T_CTSK
```

public class T_CTSK **標準仕様**

タスク生成情報のパケット形式クラス。

タスク生成情報のパケット形式クラスは、タスクアタッチクラスでタスクを生成する際に用いるクラスである。

■クラス定義

```
package org.jtron.attach;

public class T_CTSK {
        public T_CTSK(String task, int itskpri, int stksz);
        public T_CTSK(int tskatr, int exinf,
                        String task, int itskpri, int stksz);

        public int tskatr;
        public int exinf;
        public String task;
        public int itskpri;
        public int stksz;

        public static final int
                TA_HLNG   = 0x00,
                TA_ASM    = 0x01;
        public static final int
                TA_ACT    = 0x02,
                TA_RSTR   = 0x04;

 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```

■コンストラクタ

```
public T_CTSK(String task, int itskpri, int stksz);
```

【パラメータ】

String	task	タスクの起動番地を表わす文字列(シンボル)
int	itskpri	タスクの起動時優先度(★)
int	stksz	タスクのスタック領域のサイズ(バイト数)(★)

【機能】

パラメータで指定される引数と同名の変数を設定する。
引数に無い属性と拡張情報は、属性=TA_HLNG 拡張情報=0 とする。

```
public T_CTSK(int tskatr, int exinf,
              String task, int itskpri, int stksz);
```

【パラメータ】

int	tskatr	タスク属性(★)
int	exinf	タスクの拡張情報(★)
String	task	タスクの起動番地を表わす文字列(シンボル)
int	itskpri	タスクの起動時優先度(★)
int	stksz	タスクのスタック領域のサイズ(バイト数)(★)

【機能】

パラメータで指定される引数と同名の変数を設定する。

■変数

int	tskatr	タスク属性(★)
int	exinf	タスクの拡張情報(★)

String	task	タスクの起動番地を表わす文字列(シンボル)
int	itskpri	タスクの起動時優先度(★)
int	stksz	タスクのスタック領域のサイズ(バイト数)(★)

■定数

TA_HLNG	0x00	高級言語用のインタフェースで処理単位を起動
TA_ASM	0x01	アセンブリ言語用のインタフェースで処理単位を起動
TA_ACT	0x02	タスクを起動された状態で生成
TA_RSTR	0x04	制約タスク

【JTRON1.0 仕様との相違】

タスク生成機能に対応した新クラスである。

【補足説明】

タスクの起動番地（エントリアドレス）は文字列で表わされる。この文字列の解釈は実装定義であるが、例としては関数（サブルーチン、手続き）のシンボル名が想定される。スタックはサイズのみを指定してカーネルの自動確保機能を用いる。

4.2.5.3 簡易タスク状態クラス(T_RTST)

```
java.lang.Object
    |
    └──org.jtron.attach.T_RTST
```

public class T_RTST **標準仕様**

簡易タスク状態のパケット形式クラス。

タスクアタッチクラスの referSimple() で返される簡易タスク状態表わすクラスである。

■クラス定義

```
package org.jtron.attach;

public class T_RTST {
        public int tskstat;
        public int tskwait;

        public static final int
          TTS_RUN    = 0x01,
          TTS_RDY    = 0x02,
          TTS_WAI    = 0x04,
          TTS_SUS    = 0x08,
          TTS_WAS    = 0x0C,
          TTS_DMT    = 0x10;

        public static final int
          TTW_SLP    = 0x0001,
          TTW_DLY    = 0x0002,
          TTW_SEM    = 0x0004,
          TTW_FLG    = 0x0008,
          TTW_SDTQ   = 0x0010,
          TTW_RDTQ   = 0x0020,
          TTW_MBX    = 0x0040,
          TTW_MTX    = 0x0080,
```

```
        TTW_SMBF    = 0x0100,
        TTW_RMBF    = 0x0200,
        TTW_CAL     = 0x0400,
        TTW_ACP     = 0x0800,
        TTW_RDV     = 0x1000,
        TTW_MPF     = 0x2000,
        TTW_MPL     = 0x4000;

 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```

■変数

int	tskstat	タスク状態(★)
int	tskwait	待ち要因(★)

■定数

TTS_RUN	0x01	実行状態
TTS_RDY	0x02	実行可能状態
TTS_WAI	0x04	待ち状態
TTS_SUS	0x08	強制待ち状態
TTS_WAS	0x0c	二重待ち状態
TTS_DMT	0x10	休止状態

TTW_SLP	0x0001	起床待ち状態
TTW_DLY	0x0002	時間経過待ち状態
TTW_SEM	0x0004	セマフォ資源の獲得待ち状態
TTW_FLG	0x0008	イベントフラグ待ち状態
TTW_SDTQ	0x0010	データキューへの送信待ち状態
TTW_RDTQ	0x0020	データキューから受信待ち状態
TTW_MBX	0x0040	メールボックスからの受信待ち状態
TTW_MTX	0x0080	ミューテックスのロック待ち状態
TTW_SMBF	0x0100	メッセージバッファへの送信待ち状態
TTW_RMBF	0x0200	メッセージバッファからの受信待ち状態
TTW_CAL	0x0400	ランデブの成立待ち状態
TTW_ACP	0x0800	ランデブの受付待ち状態

TTW_RDV	0x1000	ランデブの完了待ち状態
TTW_MPF	0x2000	固定長メモリブロックの獲得待ち状態
TTW_MPL	0x4000	可変長メモリブロックの獲得待ち状態

【JTRON1.0 仕様との相違】

μITRON4.0 仕様に対応した新クラスである。

4.2.5.4 タスク状態クラス(T_RTSK)

```
org.jtron.attach.T_RTST
    |
    └── org.jtron.attach.T_RTSK
```

public class T_RTSK **標準仕様**

タスク状態のパケット形式クラス。

タスクアタッチクラスの refer() で返されるタスク状態を表わすクラスである。

■クラス定義

```
package org.jtron.attach;

public class T_RTSK extends T_RTST {
        public int tskpri;
        public int tskbpri;
        public int wobjid;
        public int lefttmo;
        public int actcnt;
        public int wupcnt;
        public int suscnt;

 // 下層のμITRON4.0 仕様 OS に合わせた実装独自のフィールド
}
```

■変数

int	tskpri	タスクの現在優先度(★)
int	tskbpri	タスクのベース優先度(★)
int	wobjid	待ち対象のオブジェクトの ID 番号(★)
int	lefttmo	タイムアウトするまでの時間(★)
int	actcnt	起動要求キューイング数(★)
int	wupcnt	起床要求キューイング数(★)

int	suscnt	強制待ち要求ネスト数(★)

【JTRON1.0 仕様との相違】

μITRON4.0 仕様での変更に対応した。
簡易タスク状態クラス（T_RTST）と内容が重複するため、簡易タスク状態クラスからの派生クラスとする。

4.2.5.5 タスク例外処理定義情報クラス(T_DTEX)

```
java.lang.Object
    |
    └──org.jtron.attach.T_DTEX
```

public class T_DTEX 標準仕様

タスク例外処理定義情報のパケット形式クラス。

■クラス定義

```
package org.jtron.attach;

public class T_DTEX {
        public T_DTEX(String texrtn);
        public T_DTEX(int texatr, String texrtn);

        public int texatr;
        public String texrtn;

        public static final int
                TA_HLNG = 0x00,
                TA_ASM  = 0x01;

 // 下層のμITRON4.0 仕様 OS に合わせた実装独自のフィールド
}
```

■コンストラクタ

public T_DTEX(String texrtn);

【パラメータ】

String	texrtn	タスク例外処理ルーチン起動番地を表わす文字列

【機能】

パラメータで指定される引数と同名の変数を設定する。
引数に無い属性は、属性=TA_HLNG とする。

```
public T_DTEX(int texatr, String texrtn);
```

【パラメータ】

int	texatr	タスク例外処理ルーチン属性(★)
String	texrtn	タスク例外処理ルーチン起動番地を表わす文字列

【機能】

パラメータで指定される引数と同名の変数を設定する。

■変数

int	texatr	タスク例外処理ルーチン属性(★)
String	texrtn	タスク例外処理ルーチン起動番地を表わす文字列

■定数

TA_HLNG	0x00	高級言語用のインタフェースで処理単位を起動
TA_ASM	0x01	アセンブリ言語用のインタフェースで処理単位を起動

【JTRON1.0 仕様との相違】

μITRON4.0 仕様に対応した新クラスである。

4.2.5.6 タスク例外状態クラス(T_RTEX)

```
java.lang.Object
    |
    └──org.jtron.attach.T_RTEX
```

public class T_RTEX **標準仕様**

タスク例外状態のパケット形式クラス。

タスクアタッチクラスの referTaskException() で返されるタスク例外状態を表わすクラスである。

■クラス定義

```
package org.jtron.attach;

public class T_RTEX {
        public int texstat;
        public int pndptn;

        public static final int
                TTEX_ENA = 0x00,
                TTEX_DIS = 0x01;

 // 下層のμITRON4.0 仕様 OS に合わせた実装独自のフィールド
}
```

■変数

int	texstat	タスク例外処理の状態(★)
int	pndptn	保留例外要因(★)

■定数

TTEX_ENA	0x00	タスク例外処理許可状態
TTEX_DIS	0x01	タスク例外処理禁止状態

【JTRON1.0 仕様との相違】

μITRON4.0 仕様に対応した新クラスである。

4.2.5.7 オーバーランハンドラ状態クラス(T_ROVR)

```
java.lang.Object
    |
    └──org.jtron.attach.T_ROVR
```

public class T_ROVR **標準仕様**

オーバーランハンドラ状態のパケット形式クラス。

タスクアタッチクラスの referOverrunHandler() で返されるオーバーランハンドラ状態を表わすクラスである。

■クラス定義

```
package org.jtron.attach;

public class T_ROVR {
        public int ovrstat;
        public int leftotm;

        public static final int
                TOVR_STP = 0x00,
                TOVR_STA = 0x01;

 // 下層のμITRON4.0 仕様 OS に合わせた実装独自のフィールド
}
```

■変数

int	ovrstat	オーバーランハンドラの動作状態(★)
int	leftotm	残りのプロセッサ時間(単位:ms)(★)

■定数

TOVR_STP	0x00	上限プロセッサ時間が設定されていない
TOVR_STA	0x01	上限プロセッサ時間が設定されてい

る

【JTRON1.0 仕様との相違】

μITRON4.0 仕様に対応した新クラスである。

4.2.6 セマフォ

ITRON のセマフォを操作するために以下のクラス群がある。

・セマフォアタッチクラス
・セマフォ生成情報クラス
・セマフォ状態クラス

セマフォアタッチクラスは ITRON のセマフォを表現する。

このクラスのインスタンスを生成することによりセマフォの生成ができるほか、生成済みセマフォに対応するインスタンスを得ることもできる。

このインスタンスのメソッドを呼び出すことによってセマフォ資源の獲得や返却などを行う。

4.2.6.1 セマフォアタッチクラス(Semaphore)

```
java.lang.Object
    |
    +--org.jtron.attach.Semaphore
```

public class Semaphore **標準仕様**

セマフォアタッチクラス。

ITRON セマフォを操作するためのクラスである。

■クラス定義

```
package org.jtron.attach;

public class Semaphore {
        public Semaphore(int semid);
        public Semaphore(int semid, T_CSEM pk_csem);
        public Semaphore(T_CSEM pk_csem);

        public int getId();

        public void delete();
        public void signal();
        public void waitSemaphore();
        public void poll();
        public void waitSemaphore(int timeout);
        public T_RSEM refer();
}
```

■コンストラクタ

```
public Semaphore(int semid) throws JtronException;
```

【パラメータ】

int	semid	セマフォ ID(★)

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたは JTRON による例外クラス)

【機能】

セマフォ ID を指定して、既存のセマフォに接続するインスタンスを生成する。

```
public Semaphore(int semid, T_CSEM pk_csem)
                        throws ItronCauseException;
```

【パラメータ】

int	semid	セマフォ ID(★)
T_CSEM	pk_csem	セマフォ生成情報クラス

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

cre_sem サービスコール呼び出しに相当する。
セマフォを生成し、接続するインスタンスを生成する。

```
public Semaphore(T_CSEM pk_csem)
                        throws ItronCauseException;
```

【パラメータ】

T_CSEM	pk_csem	セマフォ生成情報クラス

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

acre_sem サービスコール呼び出しに相当する。
セマフォを生成し、接続するインスタンスを生成する。

■メソッド

```
public int getId();
```

【戻り値】

int セマフォ ID(★)

【機能】

接続しているセマフォのセマフォ ID を返す。

```
public void delete() throws ItronCauseException;
```

【例外】

ItronCauseException ITRON による例外クラス

【機能】

del_sem サービスコール呼び出しに相当する。

```
public void signal() throws ItronCauseException;
```

【例外】

ItronCauseException ITRON による例外クラス

【機能】

sig_sem サービスコール呼び出しに相当する。

```
public void waitSemaphore() throws ItronCauseException;
```

【例外】

ItronCauseException ITRON による例外クラス

【機能】

wai_sem サービスコール呼び出しに相当する。

```
public void poll() throws ItronCauseException;
```

【例外】

ItronCauseException ITRON による例外クラス

【機能】

pol_sem サービスコール呼び出しに相当する。

```
public void waitSemaphore(int timeout)
                        throws ItronCauseException;
```

【パラメータ】

int	timeout	タイムアウト時間(単位:ms)(★)

【例外】

ItronCauseException ITRON による例外クラス

【機能】

twai_sem サービスコール呼び出しに相当する。

```
public T_RSEM refer() throws JtronException;
```

【戻り値】

T_RSEM	セマフォ状態クラス

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたは JTRON による例外ク

ラス)

【機能】

ref_sem サービスコール呼び出しに相当する。

【JTRON1.0 仕様との相違および仕様決定の理由】

・delete()の動作が、イベントフラグやメールボックスではそれぞれ del_flg、del_mbx サービスコール相当なのに対して、セマフォだけ del_sem サービスコール相当の動作になっていなかったので、これを del_sem サービスコール相当に変更した。

・referStatus()は名称が冗長なので refer()に変更した。

4.2.6.2 セマフォ生成情報クラス(T_CSEM)

```
java.lang.Object
   |
   └── org.jtron.attach.T_CSEM
```

public class T_CSEM **標準仕様**

セマフォ生成情報のパケット形式クラス。

セマフォ生成情報のパケット形式クラスは、セマフォアタッチクラスでセマフォを生成する際に用いるクラスである。

■クラス定義

```
package org.jtron.attach;

public class T_CSEM {
        public T_CSEM(int cnt);
        public T_CSEM(int sematr, int semcnt, int maxsem);

        public int sematr;
        public int isemcnt;
        public int maxsem;

        public static final int
                TA_TFIFO = 0x00,
                TA_TPRI  = 0x01;

 // 下層のμITRON4.0 仕様 OS に合わせた実装独自のフィールド
}
```

■コンストラクタ

public T_CSEM(int cnt);

【パラメータ】

int	cnt	セマフォの資源数(★)

【機能】

セマフォの資源数(cnt)で指定される引数をセマフォの資源数の初期値(isemcnt)とセマフォの最大資源数(maxsem)に設定する。
引数に無い属性は、属性=TA_TFIFO とする。

```
public T_CSEM(int sematr, int semcnt, int maxsem);
```

【パラメータ】

int	sematr	セマフォ属性(★)
int	isemcnt	セマフォの資源数の初期値(★)
int	maxsem	セマフォの最大資源数(★)

【機能】

パラメータで指定される引数と同名の変数を設定する。

■変数

int	sematr	セマフォ属性(★)
int	isemcnt	セマフォの資源数の初期値(★)
int	maxsem	セマフォの最大資源数(★)

■定数

TA_TFIFO	0x00 タスクの待ち行列を FIFO 順に
TA_TPRI	0x01 タスクの待ち行列を優先度順に

【JTRON1.0 仕様との相違】

フィールドを初期設定できるコンストラクタを追加する。

4.2.6.3 セマフォ状態クラス(T_RSEM)

```
java.lang.Object
    |
    └──org.jtron.attach.T_RSEM
```

public class T_RSEM **標準仕様**

セマフォ状態のパケット形式クラス。

セマフォアタッチクラスの refer() で返されるセマフォ状態を表わすクラスである。

■クラス定義

```
package org.jtron.attach;

public class T_RSEM {
        public int wtskid;
        public int semcnt;

 // 下層のμITRON4.0 仕様 OS に合わせた実装独自のフィールド
}
```

■変数

int	wtskid	セマフォの待ち行列の先頭タスクの ID 番号(★)
int	semcnt	セマフォの現在の資源数(★)

【JTRON1.0 仕様との相違】

μITRON4.0 仕様での変数名変更に対応した。

4.2.7 イベントフラグ

ITRONのイベントフラグを操作するために以下のクラス群がある。

・イベントフラグアタッチクラス
・イベントフラグ生成情報クラス
・イベントフラグ状態クラス

イベントフラグアタッチクラスはITRONのイベントフラグを表現する。

このクラスのインスタンスを生成することによりイベントフラグの生成ができるほか、生成済みイベントフラグに対応するインスタンスを得ることもできる。

このインスタンスのメソッドを呼び出すことによってイベントフラグのセット、クリア、待ちなどを行う。

4.2.7.1 イベントフラグアタッチクラス(EventFlag)

```
java.lang.Object
    |
    +--org.jtron.attach.EventFlag
```

public class EventFlag **標準仕様**

イベントフラグアタッチクラス。

ITRON イベントフラグを操作するためのクラスである。

■クラス定義

```
package org.jtron.attach;

public class EventFlag {
        public EventFlag(int flgid);
        public EventFlag(int flgid, T_CFLG pk_cflg);
        public EventFlag(T_CFLG pk_cflg);

        public int getId();

        public void delete();
        public void set(int setptn);
        public void clear(int clrptn);
        public int waitFlag(int waiptn, int wfmode);
        public int poll(int waiptn, int wfmode);
        public int waitFlag(int waiptn, int wfmode, int tmout);
        public T_RFLG refer();

        public static final int
                TWF_ANDW = 0x00,
                TWF_ORW  = 0x01;
}
```

■コンストラクタ

```
public EventFlag(int flgid) throws JtronException;
```

【パラメータ】

int	flgid	既存の接続対象のイベントフラグ ID 番号(★)

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたは JTRON による例外クラス)

【機能】

イベントフラグ ID を指定して、既存のイベントフラグに接続するインスタンスを生成する。

```
public EventFlag(int flgid, T_CFLG pk_cflg)
                        throws ItronCauseException;
```

【パラメータ】

int	flgid	生成対象のイベントフラグの ID 番号(★)
T_CFLG	pk_cflg	イベントフラグ生成情報クラス

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

cre_flg サービスコール呼び出しに相当する。
イベントフラグを生成し、接続するインスタンスを生成する。

```
public EventFlag(T_CFLG pk_cflg)
                        throws ItronCauseException;
```

【パラメータ】

T_CFLG pk_cflg イベントフラグ生成情報クラス

【例外】

ItronCauseException ITRON による例外クラス

【機能】

acre_flg サービスコール呼び出しに相当する。
イベントフラグを生成し、接続するインスタンスを生成する。

■メソッド

```
public int getId();
```

【戻り値】

int イベントフラグ ID(★)

【機能】

接続しているイベントフラグのイベントフラグ ID を返す。

```
public void delete() throws ItronCauseException;
```

【例外】

ItronCauseException ITRON による例外クラス

【機能】

del_flg サービスコール呼び出しに相当する。

```
public void set(int setptn) throws ItronCauseException;
```

【パラメータ】

int setptn セットするビットパターン(★)

【例外】

ItronCauseException ITRON による例外クラス

【機能】

set_flg サービスコール呼び出しに相当する。

```
public void clear(int clrptn) throws ItronCauseException;
```

【パラメータ】

int	clrptn	クリアするビットパターン(ビット毎の反転値)(★)

【例外】

ItronCauseException ITRON による例外クラス

【機能】

clr_flg サービスコール呼び出しに相当する。

```
public int waitFlag(int waiptn, int wfmode)
                        throws ItronCauseException;
```

【パラメータ】

int	waiptn	待ちビットパターン(★)
int	wfmode	待ちモード(★)

【戻り値】

int	待ち解除時のビットパターン(★)

【例外】

ItronCauseException ITRON による例外クラス

【機能】

wai_flg サービスコール呼び出しに相当する。

```
public int poll(int waiptn, int wfmode)
                        throws ItronCauseException;
```

【パラメータ】

int	waiptn	待ちビットパターン(★)
int	wfmode	待ちモード(★)

【戻り値】

int	待ち解除時のビットパターン(★)

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

pol_flg サービスコール呼び出しに相当する。

```
public int waitFlag(int waiptn, int wfmode, int tmout)
                        throws ItronCauseException;
```

【パラメータ】

int	waiptn	待ちビットパターン(★)
int	wfmode	待ちモード(★)
int	tmout	タイムアウト指定(単位:ms)(★)

【戻り値】

int	待ち解除時のビットパターン(★)

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

twai_flg サービスコール呼び出しに相当する。

```
public int T_RFLG refer() throws JtronException;
```

【戻り値】

T_RFLG	イベントフラグ状態のパケット形式クラス

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたはJTRON による例外クラス)

【機能】

ref_flg サービスコール呼び出しに相当する。

■定数

TWF_ANDW	0x00	イベントフラグの AND 待ち
TWF_ORW	0x01	イベントフラグの OR 待ち

【JTRON1.0 仕様との相違および仕様決定の理由】

・referStatus()は名称が冗長なので refer()に変更した。

4.2.7.2 イベントフラグ生成情報クラス(T_CFLG)

```
java.lang.Object
    |
    └──org.jtron.attach.T_CFLG
```

public class T_CFLG **標準仕様**

イベントフラグ生成情報のパケット形式クラス。

イベントフラグ生成情報のパケット形式クラスは、イベントフラグアタッチクラスでイベントフラグを生成する際に用いるクラスである。

■クラス定義

```
package org.jtron.attach;

public class T_CFLG {
        public T_CFLG();
        public T_CFLG(int flgatr, int iflgptn);

        public int flgatr;
        public int iflgptn;

        public static final int
                TA_TFIFO  = 0x00,
                TA_TPRI   = 0x01;

        public static final int
                TA_WSGL   = 0x00,
                TA_WMUL   = 0x02;

        public static final int
                TA_CLR    = 0x04;

 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```

■コンストラクタ

```
public T_CFLG();
```

【機能】

イベントフラグ属性 flgatr=(TA_FIFO|TA_WSGL)、イベントフラグのビットパターンの初期値 iflgptn=0 とする。

```
public T_CFLG(int flgatr, int iflgptn);
```

【パラメータ】

int	flgatr	イベントフラグ属性(★)
int	iflgptn	イベントフラグのビットパターンの初期値(★)

【機能】
パラメータで指定される引数と同名の変数を設定する。

■変数

int	flgatr	イベントフラグ属性(★)
int	iflgptn	イベントフラグのビットパターンの初期値(★)

■定数

TA_TFIFO	0x00	タスクの待ち行列を FIFO 順に
TA_TPRI	0x01	タスクの待ち行列を優先度順に
TA_WSGL	0x00	イベントフラグを複数のタスクが持つことを許さない
TA_WMUL	0x02	イベントフラグを複数のタスクが持つことを許す

TA_CLR	0x04	待ち解除時にイベントフラグをクリア

【JTRON1.0 仕様との相違】

μITRON4.0 仕様での変更に対応した。
フィールドを初期設定できるコンストラクタを追加する。

4.2.7.3 イベントフラグ状態クラス(T_RFLG)

```
java.lang.Object
    |
    +--org.jtron.attach.T_RFLG
```

public class T_RFLG **標準仕様**

イベントフラグ状態のパケット形式クラス。

イベントフラグアタッチクラスの refer() で返されるイベントフラグ状態を表わすクラスである。

■クラス定義

```
package org.jtron.attach;

public class T_RFLG {
        public int wtskid;
        public int flgptn;

 // 下層のμITRON4.0 仕様 OS に合わせた実装独自のフィールド
}
```

■変数

int	wtskid	イベントフラグの待ち行列の先頭のタスクの ID 番号(★)
int	flgptn	イベントフラグの現在のビットパターン(★)

【JTRON1.0 仕様との相違】

μITRON4.0 仕様での変更に対応した。

4.2.8 データキュー

ITRON のデータキューを操作するために以下のクラス群がある。

・データキューアタッチクラス
・データキュー生成情報クラス
・データキュー状態クラス

データキューアタッチクラスは ITRON のデータキューを表現する。

このクラスのインスタンスを生成することによりデータキューの生成ができるほか、生成済みデータキューに対応するインスタンスを得ることもできる。

このインスタンスのメソッドを呼び出すことによってデータキューに対してメッセージの送信、受信などを行う。送受信メッセージはメモリ操作クラスによって表現される。

ITRON のデータキュー機能は１つのメモリアドレス値を送受信するだけで、その値も正常なメモリアドレスである必要性がないことから、それを単純な整数値と見なす応用があり得る。そこでメモリ操作クラスによるメッセージの他に整数値の送受信もできるようになっている。しかしその区別はすべてユーザに任されている。整数値として送信されたものをメッセージとして受信し、受け取ったメモリ操作クラスで内容をアクセスすると予期しないメモリ破壊につながる恐れがあるので注意が必要である。

4.2.8.1 データキューアタッチクラス(DataQueue)

```
java.lang.Object
    |
    +--org.jtron.attach.DataQueue
```

public class DataQueue 標準仕様

データキューアタッチクラス。

ITRON データキューを操作するためのクラスである。

■クラス定義

```
package org.jtron.attach;

public class DataQueue {
        public DataQueue(int dtqid);
        public DataQueue(int dtqid, T_CDTQ pk_cdtq);
        public DataQueue(T_CDTQ pk_cdtq);

        public int getId();

        public void delete();

    // メッセージの送信
        public void send(ItronMemory data);
        public void pollSend(ItronMemory data);
        public void send(ItronMemory data, int tmout);
        public void forceSend(ItronMemory data);

    // 整数値の送信
        public void sendValue(int data);
        public void pollSendValue(int data);
        public void sendValue(int data, int tmout);
        public void forceSendValue(int data);

    // メッセージの受信
```

```
        public ItronMemory receive(int length);
        public ItronMemory pollReceive(int length);
        public ItronMemory receive(int length, int tmout);

    // 整数値の受信
        public int receiveValue();
        public int pollReceiveValue();
        public int receiveValue(int tmout);

        public T_RDTQ refer();
}
```

■コンストラクタ

```
public DataQueue(int dtqid) throws JtronException;
```

【パラメータ】

int	dtqid	既存の接続対象のデータキューの ID 番号(★)

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたはJTRON による例外クラス)

【機能】

データキューID を指定して、既存のデータキューに接続するインスタンスを生成する。

```
public DataQueue(int dtqid, T_CDTQ pk_cdtq)
                        throws ItronCauseException;
```

【パラメータ】

int	dtqid	生成対象のデータキューの ID 番号(★)

T_CDTQ pk_cdtq データキュー生成情報クラス

【例外】

ItronCauseException ITRON による例外クラス

【機能】

cre_dtq サービスコール呼び出しに相当する。
データキューを生成し、接続するインスタンスを生成する。

```
public DataQueue(T_CDTQ pk_cdtq)
                        throws ItronCauseException;
```

【パラメータ】

T_CDTQ pk_cdtq データキュー生成情報クラス

【例外】

ItronCauseException ITRON による例外クラス

【機能】

acre_dtq サービスコール呼び出しに相当する。
データキューを生成し、接続するインスタンスを生成する。

■メソッド

```
public int getId();
```

【戻り値】

int データキューID(★)

【機能】

接続しているデータキューのデータキューID を返す。

```
public void delete() throws ItronCauseException;
```

【例外】

ItronCauseException ITRON による例外クラス

【機能】

del_dtq サービスコール呼び出しに相当する。

```
public void send(ItronMemory data)
                        throws ItronCauseException;
```

【パラメータ】

ItronMemory data データキューへ送信するデータ

【例外】

ItronCauseException ITRON による例外クラス

【機能】

snd_dtq サービスコール呼び出しに相当する。

```
public void pollSend(ItronMemory data)
                        throws ItronCauseException;
```

【パラメータ】

ItronMemory data データキューへ送信するデータ

【例外】

ItronCauseException ITRON による例外クラス

【機能】

psnd_dtq サービスコール呼び出しに相当する。

```
public void send(ItronMemory data, int tmout)
                        throws ItronCauseException;
```

【パラメータ】

```
        ItronMemory  data   データキューへ送信するデータ
        int          tmout  タイムアウト指定(単位:ms)(★)
```

【例外】

```
        ItronCauseException  ITRON による例外クラス
```

【機能】

tsnd_dtq サービスコール呼び出しに相当する。

```
public void forceSend(ItronMemory data)
                        throws ItronCauseException;
```

【パラメータ】

```
        ItronMemory   data  データキューへ送信するデータ
```

【例外】

```
        ItronCauseException  ITRON による例外クラス
```

【機能】

fsnd_dtq サービスコール呼び出しに相当する。

```
public void sendValue(int data)
                        throws ItronCauseException;
```

【パラメータ】

```
        int     data        データキューへ送信するデータ(★)
```

【例外】

```
        ItronCauseException   ITRON による例外クラス
```

【機能】

snd_dtq サービスコール呼び出しに相当する。

```
public void pollSendValue(int data)
                        throws ItronCauseException;
```

【パラメータ】

int	data	データキューへ送信するデータ(★)

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

psnd_dtq サービスコール呼び出しに相当する。

```
public void sendValue(int data, int tmout)
                        throws ItronCauseException;
```

【パラメータ】

int	data	データキューへ送信するデータ(★)
int	tmout	タイムアウト指定(単位:ms)(★)

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

tsnd_dtq サービスコール呼び出しに相当する。

```
public void forceSendValue(int data)
                        throws ItronCauseException;
```

【パラメータ】

int	data	データキューへ送信するデータ(★)

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

fsnd_dtq サービスコール呼び出しに相当する。

```
public ItronMemory receive(int length)
                        throws JtronException;
```

【パラメータ】

int	length	受信データの長さ(★)

【戻り値】

ItronMemory	受信データの ItronMemory クラス

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたは JTRON による例外クラス)

【機能】

rcv_dtq サービスコール呼び出しに相当する。
指定した length は、返される ItronMemory の長さになる。

```
public ItronMemory pollReceive(int length)
                        throws JtronException;
```

【パラメータ】

int	length	受信データの長さ(★)

【戻り値】

ItronMemory	受信データの ItronMemory クラス

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたは JTRON による例外クラス)

【機能】

prcv_dtq サービスコール呼び出しに相当する。

指定した length は、返される ItronMemory の長さになる。

```
public ItronMemory receive(int length, int tmout)
                        throws JtronException;
```

【パラメータ】

int	length	受信データの長さ(★)
int	tmout	タイムアウト指定(単位:ms)(★)

【戻り値】

ItronMemory	受信データの ItronMemory クラス

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたは JTRON による例外クラス)

【機能】

trcv_dtq サービスコール呼び出しに相当する。
指定した length は、返される ItronMemory の長さになる。

```
public int receiveValue() throws ItronCauseException;
```

【戻り値】

int	データキューから受信したデータ(★)

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

rcv_dtq サービスコール呼び出しに相当する。

```
public int pollReceiveValue()
                        throws ItronCauseException;
```

【戻り値】

int	データキューから受信したデータ(★)

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

prcv_dtq サービスコール呼び出しに相当する。

```
public int receiveValue(int tmout)
                        throws ItronCauseException;
```

【パラメータ】

int	tmout	タイムアウト指定(単位:ms)(★)

【戻り値】

int	データキューから受信したデータ(★)

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

trcv_dtq サービスコール呼び出しに相当する。

```
public T_RDTQ refer() throws JtronException;
```

【戻り値】

T_RDTQ	データキュー状態を返すパケットクラス

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたは JTRON による例外クラス)

【機能】

ref_dtq サービスコール呼び出しに相当する。

【JTRON1.0 仕様との相違】

μITRON4.0 仕様に対応した新クラスである。
μITRON の送受信データ型 VP_INT を取り扱うため、ItronMemory と int 型を送受信するメソッドをそれぞれ用意した。

4.2.8.2 データキュー生成情報クラス(T_CDTQ)

```
java.lang.Object
    |
    └──org.jtron.attach.T_CDTQ
```

public class T_CDTQ **標準仕様**

データキュー生成情報のパケット形式クラス。

データキュー生成情報のパケット形式クラスは、データキューアタッチクラスで、データキューを生成する際に用いるクラスである。

■クラス定義

```
package org.jtron.attach;

public class T_CDTQ {
        public T_CDTQ(int dtqcnt);
        public T_CDTQ(int dtqatr, int dtqcnt);

        public int dtqatr;
        public int dtqcnt;

        public static final int
                TA_TFIFO = 0x00,
                TA_TPRI  = 0x01;

        public static final int TSZ_DTQ(int dtqcnt);

 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```

■コンストラクタ

public T_CDTQ(int dtqcnt);

【パラメータ】

int	dtqcnt	データキュー領域の容量(データの個数)(★)

【機能】

パラメータで指定される引数と同名の変数を設定する。
データキュー属性は、dtqatr=TA_TFIFOとする。

```
public T_CDTQ(int dtqatr, int dtqcnt);
```

【パラメータ】

int	dtqatr	データキュー属性(★)
int	dtqcnt	データキュー領域の容量(データの個数)(★)

【機能】

パラメータで指定される引数と同名の変数を設定する。

■メソッド

```
public static final int TSZ_DTQ(int dtqcnt);
```

【パラメータ】

int	dtqcnt	データキュー領域の領域(データの個数)(★)

【戻り値】

int	データキュー領域のサイズ(バイト数)(★)

【機能】

dtqcnt 個のデータを格納するのに必要なデータキュー領域のサイズ(バイト数)を返す。

■変数

int	dtqatr	データキュー属性(★)
int	dtqcnt	データキュー領域の容量(データの個数)(★)

■定数

TA_TFIFO	0x00	タスクの待ち行列を FIFO 順に
TA_TPRI	0x01	タスクの待ち行列をタスクの優先度順に

【JTRON1.0 仕様との相違】

μITRON4.0 仕様に対応した新クラスである。

【補足説明】

データキュー領域は容量（データの個数）のみを指定してカーネルの自動確保機能を用いる。

4.2.8.3 データキュー状態クラス(T_RDTQ)

```
java.lang.Object
    |
    +--org.jtron.attach.T_RDTQ
```

public class T_RDTQ **標準仕様**

データキュー状態のパケット形式クラス。

データキューアタッチクラスの refer() で返されるデータキュー状態を表わすクラスである。

■クラス定義

```
package org.jtron.attach;

public class T_RDTQ {
        public int stskid;
        public int rtskid;
        public int sdtqcnt;

 // 下層のμITRON4.0 仕様 OS に合わせた実装独自のフィールド
}
```

■変数

int	stskid	データキューの送信待ち行列の先頭のタスクの ID 番号(★)
int	rtskid	データキューの受信待ち行列の先頭のタスクの ID 番号(★)
int	sdtqcnt	データキューに入っているデータの数(★)

【JTRON1.0 仕様との相違】

μITRON4.0 仕様に対応した新クラスである。

4.2.9 メールボックス

ITRONのメールボックスを操作するために以下のクラス群がある。

・メールボックスアタッチクラス
・メールボックス生成情報クラス
・メールボックス状態クラス

メールボックスアタッチクラスはITRONのメールボックスを表現する。

このクラスのインスタンスを生成することによりメールボックスの生成ができるほか、生成済みメールボックスに対応するインスタンスを得ることもできる。

このインスタンスのメソッドを呼び出すことによってメールボックスに対してメッセージの送信、受信などを行う。送受信メッセージはメモリ操作クラスによって表現される。

4.2.9.1 メールボックスアタッチクラス(MailBox)

```
java.lang.Object
    |
    +--org.jtron.attach.MailBox
```

public class MailBox 標準仕様

メールボックスアタッチクラス。

ITRON メールボックスを操作するためのクラスである。

■クラス定義

```
package org.jtron.attach;

public class MailBox {
        public MailBox(int mbxid);
        public MailBox(int mbxid, T_CMBX pk_cmbx);
        public MailBox(T_CMBX pk_cmbx);

        public int getId();

        public void delete();
        public void send(ItronMemory pk_msg);
        public ItronMemory receive(int length);
        public ItronMemory pollReceive(int length);
        public ItronMemory receive(int length, int tmout);
        public T_RMBX refer(int length);

        public static void seekNextToHeader(ItronMemory msg);
        public static int readPriority(ItronMemory msg);
        public static void writePriority(ItronMemory msg,
                                         int msgpri);
}
```

■コンストラクタ

```
public MailBox(int mbxid) throws JtronException;
```

【パラメータ】

int	mbxid	既存の接続対象のメールボックス ID 番号(★)

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたは JTRON による例外クラス)

【機能】

メールボックス ID を指定して、既存のメールボックスに接続するインスタンスを生成する。

```
public MailBox(int mbxid, T_CMBX pk_cmbx)
                        throws ItronCauseException;
```

【パラメータ】

int	mbxid	生成対象のメールボックスの ID 番号(★)
T_CMBX	pk_cmbx	メールボックス生成情報クラス

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

cre_mbx サービスコール呼び出しに相当する。
メールボックスを生成し、接続するインスタンスを生成する。

```
public MailBox(T_CMBX pk_cmbx)
                        throws ItronCauseException;
```

【パラメータ】

T_CMBX pk_cmbx メールボックス生成情報クラス

【例外】

ItronCauseException ITRON による例外クラス

【機能】

acre_mbx サービスコール呼び出しに相当する。
メールボックスを生成し、接続するインスタンスを生成する。

■メソッド

```
public int getId();
```

【戻り値】

int メールボックス ID(★)

【機能】

接続しているメールボックスのメールボックス ID を返す。

```
public void delete() throws ItronCauseException;
```

【例外】

ItronCauseException ITRON による例外クラス

【機能】

del_mbx サービスコール呼び出しに相当する。

```
public void send(ItronMemory pk_msg)
                        throws ItronCauseException;
```

【パラメータ】

ItronMemory pk_msg メールボックスへ送信するメッセージデータ

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

snd_mbx サービスコール呼び出しに相当する。

```
public ItronMemory receive(int length)
                                throws JtronException;
```

【パラメータ】

int	length	受信メッセージの長さ(★)

【戻り値】

ItronMemory	受信メッセージの ItronMemory クラス

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたは JTRON による例外クラス)

【機能】

rcv_mbx サービスコール呼び出しに相当する。
指定した length は、返される ItronMemory の長さになる。

```
public ItronMemory pollReceive(int length)
                                throws JtronException;
```

【パラメータ】

int	length	受信メッセージの長さ(★)

【戻り値】

ItronMemory	受信メッセージの ItronMemory クラス

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたは JTRON による例外クラス)

【機能】

prcv_mbx サービスコール呼び出しに相当する。
指定した length は、返される ItronMemory の長さになる。

```
public ItronMemory receive(int length, int tmout)
                                    throws JtronException;
```

【パラメータ】

int	length	受信メッセージの長さ(★)
int	tmout	タイムアウト指定(単位:ms)(★)

【戻り値】

ItronMemory	受信メッセージの ItronMemory クラス

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたは JTRON による例外クラス)

【機能】

trcv_mbx サービスコール呼び出しに相当する。
指定した length は、返される ItronMemory の長さになる。

```
public T_RMBX refer(int length) throws JtronException;
```

【パラメータ】

int	length	戻り値にて取得されるメッセージキューの先頭のメッセージデータの長

さ(★)

【戻り値】

T_RMBX	メールボックス状態クラス

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたはJTRONによる例外クラス)

【機能】

ref_mbx サービスコール呼び出しに相当する。
指定した length は、返される T_RMBX のインスタンス内メンバ pk_msg の長さになる。

■static メソッド

```
public static void seekNextToHeader(ItronMemory msg);
```

【パラメータ】

ItronMemory	msg	メッセージの ItronMemory クラス

【機能】

メッセージのアクセス位置をメッセージヘッダの直後に移動する。この位置は優先度付きメッセージの場合優先度値、そうでない場合メッセージ本体の位置である。つまりこのメソッドは、優先度無しメッセージのメッセージ本体をアクセスする前に呼び出すためのものである。

```
public static int readPriority(ItronMemory msg);
```

【パラメータ】

ItronMemory	msg	メッセージの ItronMemory クラス

【戻り値】

```
int                    メッセージ優先度(★)
```

【機能】

優先度付きメッセージの優先度値を返す。このメソッドが呼び出されると、アクセス位置を優先度値の直後に移動する。これはメッセージ本体の位置である。つまり、このメソッドは優先度付きメッセージのメッセージ本体を読み込む前に呼び出すためのものである。

```
public static void writePriority(ItronMemory msg,
                                 int msgpri);
```

【パラメータ】

```
ItronMemory  msg    メッセージの ItronMemory クラス
int          msgpri メッセージ優先度(★)
```

【機能】

優先度付きメッセージの優先度値を設定する。このメソッドが呼び出されると、アクセス位置を優先度値の直後に移動する。これはメッセージ本体の位置である。つまり、このメソッドは優先度付きメッセージのメッセージ本体に書き込む前に呼び出すためのものである。

【JTRON1.0 仕様との相違および仕様決定の理由】

・referStatus()は名称が冗長なので refer()に変更した。また返される情報で必要なためメッセージ長を示す引数を追加した。詳しくはメールボックス状態クラスの項を参照。

・メッセージヘッダの多様性を考慮してユーティリティメソッドを３つ追加した。これらを使うことによりメッセージヘッダの形式／サイズに依存しなくなり移植性が増す。seekNextToHeader()の名称を seekToBody()にしなかったのは、メッセージが優先度付きかどうかが自動では判定できないためである。メールボックスを Java から生成した場合なら判定は可能だが、既存メールボックスにアタッチした場合これは不可能である。

4.2.9.2 メールボックス生成情報クラス(T_CMBX)

```
java.lang.Object
    |
    └── org.jtron.attach.T_CMBX
```

public class T_CMBX 標準仕様

メールボックス生成情報のパケット形式クラス。

メールボックス生成情報のパケット形式クラスは、メールボックスアタッチクラスでメールボックスを生成する際に用いるクラスである。

■クラス定義

```
package org.jtron.attach;

public class T_CMBX {
        public T_CMBX();
        public T_CMBX(int mbxatr, int maxmpri);

        public int mbxatr;
        public int maxmpri;

        public static final int
                TA_TFIFO = 0x00,
                TA_TPRI  = 0x01;
        public static final int
                TA_MFIFO = 0x00,
                TA_MPRI  = 0x02;

        public static final int TSZ_MPRIHD(int maxmpri);

 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```

■コンストラクタ

```
public T_CMBX();
```

【機能】

メールボックス属性 mbxatr=(TA_TFIFO|TA_MFIFO)とする。
この時の優先度の最大値指定は不要となる。

```
public T_CMBX(int mbxatr, int maxmpri);
```

【パラメータ】

int	mbxatr	メールボックス属性(★)
int	maxmpri	メッセージ優先度の最大値(属性 TA_MPRI 指定で有効) (★)

【機能】

パラメータで指定される引数と同名の変数を設定する。

■メソッド

```
public static final int TSZ_MPRIHD(int maxmpri);
```

【パラメータ】

int	maxmpri	メッセージ優先度の最大値(★)

【戻り値】

int	メッセージキューヘッダ領域のサイズ(★)

【機能】

メッセージ優先度の最大値が maxmpri のメールボックスに必要な優先度別メッセージキューヘッダ領域のサイズ(バイト数)を返す。

■変数

int	mbxatr	メールボックス属性(★)
int	maxmpri	メッセージ優先度の最大値(属性 TA_MPRI 指定で有効)(★)

■定数

TA_TFIFO	0x00	タスクの待ち行列を FIFO 順に
TA_TPRI	0x01	タスクの待ち行列をタスクの優先度順に
TA_MFIFO	0x00	メッセージのキューを FIFO 順に
TA_MPRI	0x02	メッセージのキューをメッセージの優先度順に

【JTRON1.0 仕様との相違】

μITRON4.0 仕様での変更に対応した。
フィールドを初期設定できるコンストラクタを追加する。

【補足説明】

優先度別のメッセージキューヘッダ領域はカーネルの自動生成機能を利用する。

4.2.9.3 メールボックス状態クラス(T_RMBX)

```
java.lang.Object
    |
    └── org.jtron.attach.T_RMBX
```

public class T_RMBX **標準仕様**

メールボックス状態のパケット形式クラス。

メールボックスアタッチクラスの refer() で返されるメールボックス状態を表わすクラスである。

■クラス定義

```
package org.jtron.attach;

public class T_RMBX {
        public int wtskid;
        public ItronMemory pk_msg;

 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```

■変数

int	wtskid	メールボックスの待ち行列の先頭のタスクの ID 番号(★)
ItronMemory	pk_msg	メッセージキューの先頭のメッセージデータ。無い場合は null となる。

【JTRON1.0 仕様との相違】

μITRON4.0 仕様での変更に対応した。

【補足説明】

pk_msg フィールドに返されるメッセージの長さは不明である。この

ため、MailBox クラスの refer メソッドに長さをユーザに指定してもらう。

4.2.10 ミューテックス

ITRONのミューテックスを操作するために以下のクラス群がある。

・ミューテックスアタッチクラス
・ミューテックス生成情報クラス
・ミューテックス状態クラス

ミューテックスアタッチクラスはITRONのミューテックスを表現する。

このクラスのインスタンスを生成することによりミューテックスの生成ができるほか、生成済みミューテックスに対応するインスタンスを得ることもできる。

このインスタンスのメソッドを呼び出すことによってミューテックスに対してロック、ロック解除などを行う。

4.2.10.1 ミューテックスアタッチクラス(Mutex)

```
java.lang.Object
    |
    +--org.jtron.attach.Mutex
```

public class Mutex **標準仕様**

ミューテックスアタッチクラス。

ITRON ミューテックスを操作するためのクラスである。

■クラス定義

```
package org.jtron.attach;

public class Mutex {
        public Mutex(int mtxid);
        public Mutex(int mtxid, T_CMTX pk_cmtx);
        public Mutex(T_CMTX pk_cmtx);

        public int getId();

        public void delete();
        public void lock();
        public void pollLock();
        public void lock(int tmout);
        public void unlock();
        public T_RMTX refer();
}
```

■コンストラクタ

public Mutex(int mtxid) throws JtronException;

【パラメータ】

int	mtxid	既存の接続対象のミューテックスの ID 番号(★)

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたは JTRON による例外クラス)

【機能】

ミューテックス ID を指定して、既存のミューテックスに接続するインスタンスを生成する。

```
public Mutex(int mtxid, T_CMTX pk_cmtx)
                        throws ItronCauseException;
```

【パラメータ】

int	mtxid	生成対象のミューテックスの ID 番号(★)
T_CMTX	pk_cmtx	ミューテックス生成情報クラス

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

cre_mtx サービスコール呼び出しに相当する。
ミューテックスを生成し、接続するインスタンスを生成する。

```
public Mutex(T_CMTX pk_cmtx) throws ItronCauseException;
```

【パラメータ】

T_CMTX	pk_cmtx	ミューテックス生成情報クラス

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

acre_mtx サービスコール呼び出しに相当する。
ミューテックスを生成し、接続するインスタンスを生成する。

■メソッド

```
public int getId();
```

【戻り値】

int ミューテックス ID(★)

【機能】

接続しているミューテックスのミューテックス ID を返す。

```
public void delete() throws ItronCauseException;
```

【例外】

ItronCauseException ITRON による例外クラス

【機能】

del_mtx サービスコール呼び出しに相当する。

```
public void lock() throws ItronCauseException;
```

【例外】

ItronCauseException ITRON による例外クラス

【機能】

loc_mtx サービスコール呼び出しに相当する。

```
public void pollLock() throws ItronCauseException;
```

【例外】

ItronCauseException ITRON による例外クラス

【機能】

ploc_mtx サービスコール呼び出しに相当する。

```
public void lock(int tmout) throws ItronCauseException;
```

【パラメータ】

int tmout タイムアウト指定(単位:ms)(★)

【例外】

ItronCauseException ITRON による例外クラス

【機能】

tloc_mtx サービスコール呼び出しに相当する。

```
public void unlock() throws ItronCauseException;
```

【例外】

ItronCauseException ITRON による例外クラス

【機能】

unl_mtx サービスコール呼び出しに相当する。

```
public T_RMTX refer() throws JtronException;
```

【戻り値】

T_RMTX ミューテックス状態クラス

【例外】

JtronException JTRON 例外クラス(ITRON による例外クラスまたは JTRON による例外クラス)

【機能】

ref_mtx サービスコール呼び出しに相当する。

【JTRON1.0 仕様との相違】

μITRON4.0 仕様に対応した新クラスである。

4.2.10.2 ミューテックス生成情報クラス(T_CMTX)

```
java.lang.Object
    |
    +--org.jtron.attach.T_CMTX
```

public class T_CMTX 標準仕様

ミューテックス生成情報のパケット形式クラス。

ミューテックス生成情報のパケット形式クラスは、ミューテックスアタッチクラスでミューテックスを生成する際に用いるクラスである。

■クラス定義

```
package org.jtron.attach;

public class T_CMTX {
        public T_CMTX();
        public T_CMTX(int mtxatr, int ceilpri);

        public int mtxatr;
        public int ceilpri;

        public static final int
                TA_TFIFO   = 0x00,
                TA_TPRI    = 0x01,
                TA_INHERIT = 0x02,
                TA_CEILING = 0x03;

 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```

■コンストラクタ

public T_CMTX();

【機能】

ミューテックス属性は、mtxatr=TA_TFIFO とする。
ミューテックスの上限優先度(ceilpri)は設定されない。

```
public T_CMTX(int mtxatr, int ceilpri);
```

【パラメータ】

int	mtxatr	ミューテックス属性(★)
int	ceilpri	ミューテックスの上限優先度(★)

【機能】

パラメータで指定される引数と同名の変数を設定する。

■変数

int	mtxatr	ミューテックス属性(★)
int	ceilpri	ミューテックスの上限優先度(★)

■定数

TA_TFIFO	0x00	タスクの待ち行列を FIFO 順に
TA_TPRI	0x01	タスクの待ち行列をタスクの優先度順に
TA_INHERIT	0x02	ミューテックスが優先度継承プロトコルをサポート
TA_CEILING	0x03	ミューテックスが優先度上限プロトコルをサポート

【JTRON1.0 仕様との相違】

μITRON4.0 仕様に対応した新クラスである。

4.2.10.3 ミューテックス状態クラス(T_RMTX)

```
java.lang.Object
    |
    └──org.jtron.attach.T_RMTX
```

public class T_RMTX **標準仕様**

ミューテックス状態のパケット形式クラス。

ミューテックスアタッチクラスの refer() で返されるミューテックス状態を表わすクラスである。

■クラス定義

```
package org.jtron.attach;

public class T_RMTX {
        public int htskid;
        public int wtskid;

 // 下層のμITRON4.0 仕様 OS に合わせた実装独自のフィールド
}
```

■変数

int	htskid	ミューテックスのロックしているタスクの ID 番号(★)
int	wtskid	ミューテックスの待ち行列の先頭のタスクの ID 番号(★)

【JTRON1.0 仕様との相違】

μITRON4.0 仕様に対応した新クラスである。

4.2.11 メッセージバッファ

ITRON のメッセージバッファを操作するために以下のクラス群がある。

・メッセージバッファアタッチクラス
・メッセージバッファ生成情報クラス
・メッセージバッファ状態クラス

メッセージバッファアタッチクラスは ITRON のメッセージバッファを表現する。

このクラスのインスタンスを生成することによりメッセージバッファの生成ができるほか、生成済みメッセージバッファに対応するインスタンスを得ることもできる。

このインスタンスのメソッドを呼び出すことによってメッセージバッファに対してメッセージの送信、受信などを行う。送受信メッセージはメモリ操作クラスによって表現される。

4.2.11.1 メッセージバッファアタッチクラス(MessageBuffer)

```
java.lang.Object
    |
    +--org.jtron.attach.MessageBuffer
```

public class MessageBuffer 標準仕様

メッセージバッファアタッチクラス。

ITRON メッセージバッファを操作するためのクラスである。

■クラス定義

```
package org.jtron.attach;

public class MessageBuffer {
        public MessageBuffer(int mbfid);
        public MessageBuffer(int mbfid, T_CMBF pk_cmbf);
        public MessageBuffer(T_CMBF pk_cmbf);

        public int getId();

        public void delete();

        public void send(ItronMemory msg);
        public void send(ItronMemory data, int off, int len);
        public void pollSend(ItronMemory msg);
        public void pollSend(ItronMemory data,
                                int off, int len);
        public void send(ItronMemory msg, int tmout);
        public void send(ItronMemory data,
                        int off, int len, int tmout);

        public int receive(ItronMemory msg);
        public int pollReceive(ItronMemory msg);
        public int receive(ItronMemory msg, int tmout);
```

```
        public T_RMBF refer();
}
```

■コンストラクタ

```
public MessageBuffer(int mbfid) throws JtronException;
```

【パラメータ】

int	dtqid	既存の接続対象のメッセージバッファの ID 番号(★)

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたは JTRON による例外クラス)

【機能】

メッセージバッファ ID を指定して、既存のメッセージバッファに接続するインスタンスを生成する。

```
public MessageBuffer(int mbfid, T_CMBF pk_cmbf)
                        throws ItronCauseException;
```

【パラメータ】

int	mbfid	生成対象のメッセージバッファの ID 番号(★)
T_CMBF	pk_cmbf	メッセージバッファ生成情報クラス

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

cre_mbf サービスコール呼び出しに相当する。
メッセージバッファを生成し、接続するインスタンスを生成する。

```
public MessageBuffer(T_CMBF pk_cmbf)
                        throws ItronCauseException;
```

【パラメータ】

T_CMBF	pk_cmbf	メッセージバッファ生成情報クラス

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

acre_mbf サービスコール呼び出しに相当する。
メッセージバッファを生成し、接続するインスタンスを生成する。

■メソッド

```
public int getId();
```

【戻り値】

int	メッセージバッファ ID(★)

【機能】

接続しているメッセージバッファのメッセージバッファ ID を返す。

```
public void delete() throws ItronCauseException;
```

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

del_mbf サービスコール呼び出しに相当する。

```
public void send(ItronMemory msg)
                        throws ItronCauseException;
```

【パラメータ】

ItronMemory	msg	送信メッセージデータ

【例外】

ItronCauseException ITRON による例外クラス

【機能】

snd_mbf サービスコール呼び出しに相当する。

```
public void send(ItronMemory data, int off, int len)
                        throws ItronCauseException;
```

【パラメータ】

ItronMemory	data	書き込むデータ
int	off	書き込むデータの送信先頭オフセット値(★)
int	len	送信メッセージデータの長さ(★)

【例外】

ItronCauseException ITRON による例外クラス

【機能】

snd_mbf サービスコール呼び出しに相当する。

```
public void pollSend(ItronMemory msg)
                        throws ItronCauseException;
```

【パラメータ】

ItronMemory	msg	送信メッセージデータ

【例外】

ItronCauseException ITRON による例外クラス

【機能】

psnd_mbf サービスコール呼び出しに相当する。

```
public void pollSend(ItronMemory data, int off, int len)
                        throws ItronCauseException;
```

【パラメータ】

ItronMemory	data	書き込むデータ
int	off	書き込むデータの送信先頭オフセット値(★)
int	len	送信メッセージデータの長さ(★)

【例外】

ItronCauseException ITRON による例外クラス

【機能】

psnd_mbf サービスコール呼び出しに相当する。

```
public void send(ItronMemory msg, int tmout)
                        throws ItronCauseException;
```

【パラメータ】

ItronMemory	msg	送信メッセージデータ
int	tmout	タイムアウト指定(単位:ms)(★)

【例外】

ItronCauseException ITRON による例外クラス

【機能】

tsnd_mbf サービスコール呼び出しに相当する。

```
public void send(ItronMemory data, int off, int len,
                int tmout) throws ItronCauseException;
```

【パラメータ】

ItronMemory	data	書き込むデータ
int	off	書き込むデータの送信先頭オフセット値(★)
int	len	送信メッセージデータの長さ(★)
int	tmout	タイムアウト指定(単位:ms)(★)

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

tsnd_mbf サービスコール呼び出しに相当する。

```
public int receive(ItronMemory msg)
                        throws ItronCauseException;
```

【パラメータ】

ItronMemory	msg	受信メッセージを格納する領域

【戻り値】

int	受信メッセージのサイズ(★)

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

rcv_mbf サービスコール呼び出しに相当する。受信すべきメッセージのバイト数が msg.getLength()より大きいときの動作は実装定義である。但し、次のようになることを期待する。
受信すべきメッセージのバイト数が msg.getLength()より大きいときは、msg.getLength()バイトだけ msg に格納し、残りは捨てられる。この場合のリターン値は msg.getLength()と同じ値になる。

```
public int pollReceive(ItronMemory msg)
                        throws ItronCauseException;
```

【パラメータ】

ItronMemory msg 受信メッセージを格納する領域

【戻り値】

int 受信メッセージのサイズ(★)

【例外】

ItronCauseException ITRON による例外クラス

【機能】

prcv_mbf サービスコール呼び出しに相当する。受信すべきメッセージのバイト数が msg.getLength()より大きいときの動作は実装定義である。但し、次のようになることを期待する。
受信すべきメッセージのバイト数が msg.getLength()より大きいときは、msg.getLength()バイトだけ msg に格納し、残りは捨てられる。この場合のリターン値は msg.getLength()と同じ値になる。

```
public int receive(ItronMemory msg, int tmout)
                        throws ItronCauseException;
```

【パラメータ】

ItronMemory msg 受信メッセージを格納する領域
int tmout タイムアウト指定(単位:ms)(★)

【戻り値】

int 受信メッセージのサイズ(★)

【例外】

ItronCauseException ITRON による例外クラス

【機能】

trcv_mbf サービスコール呼び出しに相当する。受信すべきメッセージのバイト数が msg.getLength()より大きいときの動作は実装定義である。但し、次のようになることを期待する。
受信すべきメッセージのバイト数が msg.getLength()より大きいときは、msg.getLength()バイトだけ msg に格納し、残りは捨てられる。この場合のリターン値は msg.getLength()と同じ値になる。

```
public T_RMBF refer() throws JtronException;
```

【戻り値】

T_RMBF	メッセージバッファ状態クラス

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたはJTRON による例外クラス)

【機能】

ref_mbf サービスコール呼び出しに相当する。

【JTRON1.0 仕様との相違および仕様決定の理由】

・送信メソッドでメッセージの開始位置と長さを指定できるものがなかったので追加した。

・受信するたびにメッセージ領域を確保するのは無駄なので、受信メソッドのうちリターン値として ItronMemory を返していたものを廃止した。それに伴い「WithMemory」のついたメソッド名からこの部分を取り除いた。

・ItronMemory が引数の受信メソッドで受信したメッセージの長さを得る方法が規定されていなかったので、それをリターン値として返すよう変更した。また受信バッファのサイズが足りないときの動作を明記した。

・referStatus()は名称が冗長なので refer()に変更した。

・送受信データとして byte[]が指定できるメソッドを追加する案があったが、これは次の理由により不採用とした。

1． データバッファとして Java の配列をそのまま使うと、サービスコールを呼び出してスレッドが待ちになったときその配列を固定し

たままその状態になるので、Java のガーベジコレクション処理に悪影響が出る。しかも受信ではこの状態が長く続く可能性が高い。

２． 上記の解決策として、Java の配列をそのまま使わず影響のないメモリ領域にコピーして使う方法もあるが、それは結局 ItronMemory を使う方法と同じである。しかもそれは作ったコピーを毎回使い捨てすることになり、見かけの使いやすさとは裏腹に非常に無駄が多い。

３． Java 実行系の実現方式によっては上記が問題にならない場合もあり得るが、現時点では問題になるものが多い。

４． バイト配列をデータにすると、そこに int や short などのバイナリデータをパックしたりアンパックするのはユーザの責任になる。しかも通信相手が ITRON タスクなのでデータのエンディアンなどを使っている CPU に合わせなければならないが、それを行う標準的な方法は用意されていないし、用意したところで上記の問題は解決しない。

・受信時、受信すべきメッセージのサイズが受信バッファより大きい場合、rcv_mbf サービスコールではバッファの末端より先にデータを読み込んでしまい、これを防止することはできない。またそのメッセージバッファで扱えるメッセージの最大サイズを得ることもできない。これらの理由から、μITRON4.0 仕様 OS のメッセージバッファで安全にメッセージを受信する一般的な方法はない。
但し、ベンダ定義の拡張機能を使うことによって安全に受信できる可能性があることから、期待される動作を記述した。

4.2.11.2 メッセージバッファ生成情報クラス(T_CMBF)

```
java.lang.Object
    |
    +--org.jtron.attach.T_CMBF
```

public class T_CMBF 標準仕様

メッセージバッファ生成情報のパケット形式クラス。

メッセージバッファ生成情報のパケット形式クラスは、メッセージバッファアタッチクラスでメッセージバッファを生成する際に用いるクラスである。

■クラス定義

```
package org.jtron.attach;

public class T_CMBF {
        public T_CMBF(int maxmsz, int mbfsz);
        public T_CMBF(int mbfatr, int maxmsz, int mbfsz);

        public int mbfatr;
        public int maxmsz;
        public int mbfsz;

        public static final int
                TA_TFIFO = 0x00,
                TA_TPRI  = 0x01;

        public static finalint TSZ_MBF(int msgcnt, int msgsz);

 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```

■コンストラクタ

```
public T_CMBF(int maxmsz, int mbfsz);
```

【パラメータ】

int maxmsz	メッセージの最大サイズ(バイト数)(★)
int mbfsz	メッセージバッファ領域のサイズ(バイト数)(★)

【機能】

パラメータで指定される引数と同名の変数を設定する。
メッセージバッファ属性は、mbfatr=TA_TFIFO とする。

```
public T_CMBF(int mbfatr, int maxmsz, int mbfsz);
```

【パラメータ】

int mbfatr	メッセージバッファ属性(★)
int maxmsz	メッセージの最大サイズ(バイト数)(★)
int mbfsz	メッセージバッファ領域のサイズ(バイト数)(★)

【機能】

パラメータで指定される引数と同名の変数を設定する。

■static メソッド

```
public static final int TSZ_MBF(int msgcnt, int msgsz);
```

【パラメータ】

int	msgcnt	メッセージ数(★)
int	msgsz	メッセージサイズ(★)

【戻り値】

int		メッセージバッファ領域のサイズ(目安のバイト数) (★)

【機能】

サイズがmsgszバイトのメッセージをmsgcnt個バッファリングするのに必要なメッセージバッファの領域のサイズ(目安のバイト数)を返す。

■変数

int	mbfatr	メッセージバッファ属性(★)
int	maxmsz	メッセージの最大サイズ(バイト数)(★)
int	mbfsz	メッセージバッファ領域のサイズ(バイト数)(★)

■定数

TA_TFIFO	0x00	タスクの待ち行列を FIFO 順に
TA_TPRI	0x01	タスクの待ち行列をタスクの優先度順に

【JTRON1.0 仕様との相違】

μITRON4.0 仕様での変更に対応した。
フィールドを初期設定できるコンストラクタを追加する。

【補足説明】

メッセージバッファ領域はサイズ（バイト数）のみを指定してカーネルの自動確保機能を用いる。

4.2.11.3 メッセージバッファ状態クラス(T_RMBF)

```
java.lang.Object
    |
    └── org.jtron.attach.T_RMBF
```

public class T_RMBF 標準仕様

メッセージバッファ状態のパケット形式クラス。

メッセージバッファアタッチクラスの refer() で返されるメッセージバッファ状態を表わすクラスである。

■クラス定義

```
package org.jtron.attach;

public class T_RMBF {
        public int stskid;
        public int rtskid;
        public int smsgcnt;
        public int fmbfsz;

 // 下層のμITRON4.0 仕様 OS に合わせた実装独自のフィールド
}
```

■変数

int	stskid	メッセージバッファの送信待ち行列の先頭タスク ID 番号(★)
int	rtskid	メッセージバッファの受信待ち行列の先頭のタスクの ID 番号(★)
int	smsgcnt	メッセージバッファに入っているデータの数(★)
int	fmbfsz	メッセージバッファ領域の空き領域のサイズ（バイト数、最低限の管理領域を除く）(★)

【JTRON1.0 仕様との相違】

μITRON4.0 仕様での変更に対応した。

4.2.12 ランデブ

ITRON のランデブを操作するために以下のクラス群がある。

・ランデブポートアタッチクラス
・ランデブポート生成情報クラス
・ランデブポート状態クラス
・ランデブアタッチクラス
・ランデブ状態クラス

ランデブポートアタッチクラスは ITRON のランデブポートを表現する。

このクラスのインスタンスを生成することによりランデブポートの生成ができるほか、生成済みランデブポートに対応するインスタンスを得ることもできる。

このインスタンスのメソッドを呼び出すことによってランデブの呼出し、受付などを行う。呼出しメッセージはメモリ操作クラスによって表現される。

ランデブを受け付ける場合、呼出しメッセージはランデブアタッチクラスのインスタンスをあらかじめ生成しておき、そこに受け取る。そのインスタンスのメソッドを呼び出すことによってランデブの回送、終了を行う。

クライアント側タスクＣ（**ランデブ呼び出し側スレッド**）、サーバ側タスクＳ（ランデブ受付側スレッド）の典型的な動作は以下のようになる。

S-1) ランデブポートクラスのインスタンスを生成する。
S-2) ランデブクラス（もしくは継承したクラス）のインスタンスを生成する。
S-3) (S-2)のインスタンスを指定して(S-1)のインスタンスの受付メソッドを呼び出す。ランデブ呼出しがあるまでここで待ち状態になる。
C-1) ランデブポートクラスのインスタンスを生成する。
C-2) メモリ操作クラスのインスタンスを生成し、呼出しメッセージを作成する。
C-3) (C-2)のインスタンスを指定して(C-1)のインスタンスの呼出しメソッドを呼び出す。返答があるまでここで待ち状態になる。
S-4) 受付メソッドからリターンしてくると呼出しメッセージが

(S-2)のインスタンスに格納されているので、それを用いて依頼された処理を行う。

S-5) (S-2)のインスタンスの終了メソッドで返答する。

S-6) (S-3) に戻る。

C-4) 呼び出しメソッドからリターンしてくると返答メッセージが(C-2)のインスタンスに格納されている。

4.2.12.1 ランデブポートアタッチクラス(RendezvousPort)

```
java.lang.Object
    |
    +--org.jtron.attach.RendezvousPort
```

public class RendezvousPort 標準仕様

ランデブポートアタッチクラス。

ITRON ランデブポートを操作するためのクラスである。

■クラス定義

```
package org.jtron.attach;

public class RendezvousPort {
        public RendezvousPort(int porid);
        public RendezvousPort(int porid, T_CPOR pk_cpor);
        public RendezvousPort(T_CPOR pk_cpor);

        public int getId();

        public void delete();

        public int call(int calptn, ItronMemory msg,
                                        int cmsgsz);
        public int call(int calptn, ItronMemory msg,
                                int cmsgsz, int tmout);

        public int accept(int acpptn, Rendezvous msg);
        public int pollAccept(int acpptn, Rendezvous msg);
        public int accept(int acpptn, Rendezvous msg,
                                        int tmout);

        public T_RPOR refer();
}
```

■コンストラクタ

```
public RendezvousPort(int porid) throws JtronException;
```

【パラメータ】

int	porid	既存の接続対象のランデブポートの ID 番号(★)

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたは JTRON による例外クラス)

【機能】

ランデブポート ID を指定して、既存のランデブポートに接続するインスタンスを生成する。

```
public RendezvousPort(int porid, T_CPOR pk_cpor)
                        throws ItronCauseException;
```

【パラメータ】

int	porid	生成対象のランデブポートの ID 番号(★)
T_CPOR	pk_cpor	ランデブポート生成情報クラス

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

cre_por サービスコール呼び出しに相当する。
ランデブポートを生成し、接続するインスタンスを生成する。

```
public RendezvousPort(T_CPOR pk_cpor)
                        throws ItronCauseException;
```

【パラメータ】

T_CPOR pk_cpor ランデブポート生成情報クラス

【例外】

ItronCauseException ITRON による例外クラス

【機能】

acre_por サービスコール呼び出しに相当する。
ランデブポートを生成し、接続するインスタンスを生成する。

■メソッド

```
public int getId();
```

【戻り値】

int ランデブポート ID(★)

【機能】

接続しているランデブポートのランデブポート ID を返す。

```
public void delete() throws ItronCauseException;
```

【例外】

ItronCauseException ITRON による例外クラス

【機能】

del_por サービスコール呼び出しに相当する。

```
public int call(int calptn, ItronMemory msg, int cmsgsz)
                        throws ItronCauseException;
```

【パラメータ】

int	calptn	呼出し側のランデブ条件を示すビットパターン(★)
ItronMemory	msg	呼出しメッセージ／返答メッセージを格納する領域
int	cmsgsz	呼出しメッセージのサイズ(バイト数)(★)

【戻り値】

int	返答メッセージのサイズ(バイト数、正の値または 0)(★)

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

cal_por サービスコール呼び出しに相当する。返答メッセージのバイト数が msg.getLength()より大きいときの動作は実装定義である。但し、次のようになることを期待する。
返答メッセージのバイト数が msg.getLength()より大きいときは、msg.getLength()バイトだけ msg に格納され、残りは捨てられる。この場合のリターン値は msg.getLength()と同じ値になる。

```
public int call(int calptn, ItronMemory msg, int cmsgsz,
                int tmout) throws ItronCauseException;
```

【パラメータ】

int	calptn	呼出し側のランデブ条件を示すビットパターン(★)
ItronMemory	msg	呼出しメッセージ／返答メッセージを格納する領域
int	cmsgsz	呼出しメッセージのサイズ(バイト数)(★)
int	tmout	タイムアウト指定(単位:ms)(★)

【戻り値】

int	返答メッセージのサイズ(バイト数、

正の値または 0)(★)

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

tcal_por サービスコール呼び出しに相当する。返答メッセージのバイト数が msg.getLength()より大きいときの動作は実装定義である。但し、次のようになることを期待する。
返答メッセージのバイト数が msg.getLength()より大きいときは、msg.getLength()バイトだけ msg に格納され、残りは捨てられる。この場合のリターン値は msg.getLength()と同じ値になる。

```
public int accept(int acpptn, Rendezvous msg)
                        throws ItronCauseException;
```

【パラメータ】

int	acpptn	受付側のランデブ条件を示すビットパターン(★)
Rendezvous	msg	ランデブアタッチクラス

【戻り値】

int	呼出メッセージのサイズ(★)

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

acp_por サービスコール呼び出しに相当する。呼出メッセージのバイト数が msg.getLength()より大きいときの動作は実装定義である。但し、次のようなことを期待する。
呼出メッセージのバイト数が msg.getLength()より大きいときは、msg.getLength()バイトだけ msg に格納され、残りは捨てられる。この場合のリターン値は msg.getLength()と同じ値になる。

```
public int pollAccept(int acpptn, Rendezvous msg)
```

```
                        throws ItronCauseException;
```

【パラメータ】

int	acpptn	受付側のランデブ条件を示すビットパターン(★)
Rendezvous	msg	ランデブアタッチクラス

【戻り値】

int	呼出メッセージのサイズ(★)

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

pacp_por サービスコール呼び出しに相当する。呼出メッセージのバイト数が msg.getLength()より大きいときの動作は実装定義である。但し、次のようなことを期待する。
呼出メッセージのバイト数が msg.getLength()より大きいときは、msg.getLength()バイトだけ msg に格納され、残りは捨てられる。この場合のリターン値は msg.getLength()と同じ値になる。

```
public int accept(int acpptn, Rendezvous msg, int tmout)
                        throws ItronCauseException;
```

【パラメータ】

int	acpptn	受付側のランデブ条件を示すビットパターン(★)
Rendezvous	msg	ランデブアタッチクラス
int	tmout	タイムアウト指定(単位:ms)(★)

【戻り値】

int	呼出メッセージのサイズ(★)

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

tacp_por サービスコール呼び出しに相当する。呼出メッセージのバイト数が msg.getLength()より大きいときの動作は実装定義である。但し、次のようなことを期待する。
呼出メッセージのバイト数が msg.getLength()より大きいときは、msg.getLength()バイトだけ msg に格納され、残りは捨てられる。この場合のリターン値は msg.getLength()と同じ値になる。

```
public T_RPOR refer() throws JtronException;
```

【戻り値】

T_RPOR	ランデブポート状態クラス

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたは JTRON による例外クラス)

【機能】

ref_por サービスコール呼び出しに相当する。

【JTRON1.0 仕様との相違および仕様決定の理由】

・μITRON4.0 仕様での改称（ポート→ランデブポート）によりクラス名を変更した。

・μITRON4.0 仕様で pcal_por サービスコールがなくなったので pollAndCall()を削除した。

・呼出メソッドで返答をリターン値として返すのではなく指定された呼出メッセージのエリアを流用するよう変更した。これは ITRON の動作に合わせたものだが、無駄なインスタンス生成を抑える効果もある。

・受け付けるたびにメッセージ領域を確保するのは無駄なので、受付メソッドで Rendezvous をリターン値として返すのではなく引数とし

て指定するように変更した。

・forward()は本来ランデブのメソッドと考えられるのでそちらに移動した。

・referStatus()は名称が冗長なので refer()に変更した。

・呼出メッセージとして byte[]が指定できるメソッドを追加するという案があったが不採用とした。理由はメッセージバッファの項参照。

・呼出し時、受信すべき返答メッセージのサイズが呼出メッセージバッファより大きい場合、cal_por サービスコールではバッファの末端より先にデータを読み込んでしまい、これを防止することはできない。またそのランデブポートで扱える返答メッセージの最大サイズを得ることもできない。これらの理由から、μITRON4.0 仕様 OS のランデブポートで安全に返答メッセージを受信する一般的な方法はない。
但し、ベンダ定義の拡張機能を使うことによって安全に受信できる可能性があることから、期待される動作を記述した。またこれは受付時に呼出メッセージを受信する場合にも全く同様のことが言えるため、同様の仕様とした。

4.2.12.2 ランデブポート生成情報クラス(T_CPOR)

```
java.lang.Object
    |
    └── org.jtron.attach.T_CPOR
```

public class T_CPOR 標準仕様

ランデブポート生成情報のパケット形式クラス。

ランデブポート生成情報のパケット形式クラスは、ランデブポートアタッチクラスでランデブポートを生成する際に用いるクラスである。

■クラス定義

```
package org.jtron.attach;

public class T_CPOR {
        public T_CPOR(int maxcmsz, int maxrmsz);
        public T_CPOR(int poratr, int maxcmsz, int maxrmsz);

        public int poratr;
        public int maxcmsz;
        public int maxrmsz;

        public static final int
                TA_TFIFO = 0x00,
                TA_TPRI  = 0x01;

 // 下層のμITRON4.0 仕様 OS に合わせた実装独自のフィールド
}
```

■コンストラクタ

```
public T_CPOR(int maxcmsz, int maxrmsz);
```

【パラメータ】

int	maxcmsz	呼出しメッセージの最大サイズ(バイト数)(★)
int	maxrmsz	返答メッセージの最大サイズ(バイト数)(★)

【機能】

パラメータで指定される引数と同名の変数を設定する。
ランデブポート属性は、poratr=TA_TFIFO とする。

```
public T_CPOR(int poratr, int maxcmsz, int maxrmsz);
```

【パラメータ】

int	poratr	ランデブポート属性(★)
int	maxcmsz	呼出しメッセージの最大サイズ(バイト数)(★)
int	maxrmsz	返答メッセージの最大サイズ(バイト数)(★)

【機能】

パラメータで指定される引数と同名の変数を設定する。

■変数

int	poratr	ランデブポート属性(★)
int	maxcmsz	呼出しメッセージの最大サイズ(バイト数)(★)
int	maxrmsz	返答メッセージの最大サイズ(バイト数)(★)

■定数

TA_TFIFO	0x00	タスクの待ち行列を FIFO 順に
TA_TPRI	0x01	タスクの待ち行列をタスクの優先度順に

【JTRON1.0 仕様との相違】

μITRON4.0 仕様での変更に対応した。
フィールドを初期設定できるコンストラクタを追加する。

4.2.12.3 ランデブポート状態クラス(T_RPOR)

```
java.lang.Object
    |
    └──org.jtron.attach.T_RPOR
```

public class T_RPOR 標準仕様

ランデブポート状態のパケット形式クラス。

ランデブポートアタッチクラスの refer() で返されるランデブポート状態を表わすクラスである。

■クラス定義

```
package org.jtron.attach;

public class T_RPOR {
        public int ctskid;
        public int atskid;

 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```

■変数

int	ctskid	呼出し待ち行列の先頭のタスクのID(★)
int	atskid	受信待ち行列の先頭のタスクのID(★)

【JTRON1.0 仕様との相違】

μITRON4.0 仕様での変更に対応した。

4.2.12.4 ランデブアタッチクラス(Rendezvous)

public class Rendezvous　　標準仕様

ランデブアタッチクラス。

ランデブ受付時に呼出しメッセージが格納されるクラスである。
本クラスはメモリ操作クラスを継承しているので、メッセージの読み書きはメモリ操作クラスと同様の方法で行なえる。

■クラス定義

```
package org.jtron.attach;

public class Rendezvous extends ItronMemory {
        public Rendezvous(int length);

        public int getNo();

        public void forwardTo(int porid, int calptn);
        public void forwardTo(int porid, int calptn,
                        ItronMemory msg, int cmsgsz);
        public void reply(int rmsgsz);
        public void reply(ItronMemory msg, int rmsgsz);

        public T_RRDV refer();
}
```

■コンストラクタ

```
public Rendezvous(int length) throws JtronException;
```

【パラメータ】

int	length	呼出しメッセージを受け取るバッファ(★)

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたはJTRONによる例外クラス)

【機能】

呼出しメッセージを受け取るバッファを生成する。バッファの属性はスーパークラスである ItronMemory クラスのそれに準ずる。返答としても利用できる。

■メソッド

```
public int getNo() throws JtronCauseException;
```

【戻り値】

int	ランデブ番号(★)

【例外】

JtronCauseException	JTRON による例外クラス(受付可能状態)

【機能】

ランデブ番号を返す。受付可能状態の時は JtronCauseException 例外を発生させる。

```
public void forwardTo(int porid, int calptn)
                                    throws JtronException;
```

【パラメータ】

int	porid	回送先のランデブポートの ID 番号(★)
int	calptn	呼出し側のランデブ条件を示すビットパターン(★)

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたは JTRON による例外クラス)

【機能】

fwd_por サービスコール呼び出しに相当する。受付可能状態の時は JtronCauseException 例外を発生させる。

```
public void forwardTo(int porid, int calptn,
                      ItronMemory msg, int cmsgsz)
                              throws JtronException;
```

【パラメータ】

int	porid	回送先のランデブポートの ID 番号(★)
int	calptn	呼出し側のランデブ条件を示すビットパターン(★)
ItronMemory	msg	呼出しメッセージのデータ
int	cmsgsz	呼出しメッセージのサイズ(★)

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたは JTRON による例外クラス)

【機能】

fwd_por サービスコール呼び出しに相当する。受付可能状態の時は JtronCauseException 例外を発生させる。
別の呼出メッセージを回送する。

```
public void reply(int rmsgsz) throws JtronException;
```

【パラメータ】

int	rmsgsz	返答メッセージのサイズ(バイト数)(★)

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたは JTRON による例外クラス)

【機能】

rpl_rdv サービスコール呼び出しに相当する。受付可能状態の時は JtronCauseException 例外を発生させる。
返答メッセージは本インスタンスに格納されたものを利用する。

```
public void reply(ItronMemory msg, int rmsgsz)
                                    throws JtronException;
```

【パラメータ】

ItronMemory	msg	返答メッセージ
int	rmsgsz	返答メッセージのサイズ(バイト数)(★)

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたは JTRON による例外クラス)

【機能】

rpl_rdv サービスコール呼び出しに相当する。受付可能状態の時は JtronCauseException 例外を発生させる。
本インスタンスに格納されている以外のメッセージで返答する。

```
public T_RRDV refer() throws JtronException;
```

【戻り値】

T_RRDV	ランデブの状態参照クラス

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたはJTRONによる例外クラス)

【機能】

ref_rdv サービスコール呼び出しに相当する。受付可能状態の時は JtronCauseException 例外を発生させる。

【JTRON1.0 仕様との相違および仕様決定の理由】

・受付時使うためにユーザがインスタンス生成するようになったので、コンストラクタを追加した。

・μITRON4.0 仕様で追加された ref_rdv サービスコール相当を refer()として追加した。

・返答メッセージ長が呼出メッセージ長と必ず一致するとは言えないので、reply()に返答メッセージ長指定の引数を追加した。また、rpl_rdvサービスコールでは返答メッセージとして任意のものを指定できることから、返答メッセージとして別のものを指定できるメソッドを追加した。

・回送は本来本クラスの機能と考えられるのでランデブポートクラスから forward()を forwardTo()と改名して移動した。その際抜けていた回送先ランデブポートを示す引数を追加した。改名したのはこれがランデブポートクラスのメソッドであったとき、「このランデブポートに回送する」のだと誤解を招く恐れがあったためである。また、fwd_porサービスコールでは呼出メッセージとして任意のものを指定できることから、呼出メッセージとして別のものを指定できるメソッ

ドを追加した。

・呼出メッセージを受け取り、また返答するバッファとして byte[] も使えるようにするという案があったが不採用とした。理由はメッセージバッファの項参照。

【補足説明】

受け付けてからでなければ返答できず、また返答／回送したら初期状態に戻る。そのため受付済みでない状態で返答／回送しようとすると例外が投げられる。ただし、返答／回送しないうちに受付メソッドに渡された場合は無条件で初期状態に戻る。

4.2.12.5 ランデブ状態クラス(T_RRDV)

```
java.lang.Object
    |
    └── org.jtron.attach.T_RRDV
```

public class T_RRDV 標準仕様

ランデブ状態のパケット形式クラス。

ランデブアタッチクラスの refer() で返されるランデブ状態を表わすクラスである。

■クラス定義

```
package org.jtron.attach;

public class T_RRDV {
        public int wtskid;

 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```

■変数

int	wtskid	ランデブ終了待ち状態のタスクのID番号(★)

【JTRON1.0 仕様との相違】

μITRON4.0 仕様に対応した新クラスである。

4.2.13 固定長メモリプール

ITRON の固定長メモリプールを操作するために以下のクラス群がある。

・固定長メモリプールアタッチクラス
・固定長メモリプール生成情報クラス
・固定長メモリプール状態クラス
・固定長メモリブロックアタッチクラス

固定長メモリプールアタッチクラスは ITRON の固定長メモリプールを表現する。

このクラスのインスタンスを生成することにより固定長メモリプールの生成ができるほか、生成済み固定長メモリプールに対応するインスタンスを得ることもできる。

このインスタンスのメソッドを呼び出すことによって固定長メモリブロックの獲得、返却などを行う。

獲得できる固定長メモリブロックは固定長メモリブロックアタッチクラスによって表現される。これはメモリ操作クラスを継承しているため、同様の方法で内容を操作することができる。

4.2.13.1 固定長メモリプールアタッチクラス (FixedMemoryPool)

```
java.lang.Object
   |
   └── org.jtron.attach.FixedMemoryPool
```

public class FixedMemoryPool 標準仕様

固定長メモリプールアタッチクラス。

ITRON 固定長メモリプールを操作するためのクラスである。

■クラス定義

```
package org.jtron.attach;

public class FixedMemoryPool {
        public FixedMemoryPool(int mpfid, int blksz);
        public FixedMemoryPool(int mpfid,
                                T_CMPF pk_cmpf);
        public FixedMemoryPool(T_CMPF pk_cmpf);

        public int getId();

        public void delete();
        public ItronFixedMemory get();
        public ItronFixedMemory poll();
        public ItronFixedMemory get(int timeout);
        public T_RMPF refer();
}
```

■コンストラクタ

```
public FixedMemoryPool(int mpfid, int blksz)
                                throws JtronException;
```

【パラメータ】

int	mpfid	接続対象の既存固定長メモリプールの ID 番号(★)
int	blksz	上記プールから獲得できるメモリブロックのサイズ（バイト数）(★)

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたは JTRON による例外クラス)

【機能】

固定長メモリプール ID を指定して、既存の固定長メモリプールに接続するインスタンスを生成する。blksz 引数は獲得できる固定長メモリブロックのサイズでなければならない。これは get()で返され ItronFixedMemory のサイズになる。

```
public FixedMemoryPool(int mpfid, T_CMPF pk_cmpf)
                        throws ItronCauseException;
```

【パラメータ】

int	mpfid	生成対象の固定長メモリプールの ID 番号(★)
T_CMPF	pk_cmpf	固定長メモリプール生成情報クラス

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

cre_mpf サービスコール呼び出しに相当する。
固定長メモリプールを生成し、接続するインスタンスを生成する。

```
public FixedMemoryPool(T_CMPF pk_cmpf)
                        throws ItronCauseException;
```

【パラメータ】

T_CMPF pk_cmpf 固定長メモリプール生成情報クラス

【例外】

ItronCauseException ITRON による例外クラス

【機能】

acre_mpf サービスコール呼び出しに相当する。
固定長メモリプールを生成し、接続するインスタンスを生成する。

■メソッド

```
public int getId();
```

【戻り値】

int 固定長メモリプール ID(★)

【機能】

接続している固定長メモリプールの固定長メモリプール ID を返す。

```
public void delete() throws ItronCauseException;
```

【例外】

ItronCauseException ITRON による例外クラス

【機能】

del_mpf サービスコール呼び出しに相当する。

```
public ItronFixedMemory get() throws JtronException;
```

【戻り値】

ItronFixedMemory 固定長メモリブロックアタッチクラ

ス

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたはJTRONによる例外クラス)

【機能】

get_mpf サービスコール呼び出しに相当する。

```
public ItronFixedMemory poll() throws JtronException;
```

【戻り値】

ItronFixedMemory	固定長メモリブロックアタッチクラス

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたはJTRONによる例外クラス)

【機能】

pget_mpf サービスコール呼び出しに相当する。

```
public ItronFixedMemory get(int timeout)
                                throws JtronException;
```

【パラメータ】

int	tmout	タイムアウト指定(単位：ms) (★)

【戻り値】

ItronFixedMemory	固定長メモリブロックアタッチクラス

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたは JTRON による例外クラス)

【機能】

tget_mpf サービスコール呼び出しに相当する。

```
public T_RMPF refer() throws JtronException;
```

【戻り値】

T_RMPF	固定長メモリプール状態クラス

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたは JTRON による例外クラス)

【機能】

ref_mpf サービスコール呼び出しに相当する。

【JTRON1.0 仕様との相違および仕様決定の理由】

・FixedMemoryPool(int,int)に２番目の引数用途を明記した。これは既存プールから獲得できるブロックのサイズが ITRON から得られないため必要になる。

・referStatus()は名称が冗長なので refer()に変更した。

4.2.13.2 固定長メモリプール生成情報クラス(T_CMPF)

```
java.lang.Object
    |
    └── org.jtron.attach.T_CMPF
```

public class T_CMPF 標準仕様

固定長メモリプール生成情報のパケット形式クラス。

固定長メモリプール生成情報のパケット形式クラスは、固定長メモリプールアタッチクラスで固定長メモリプールを生成する際に用いるクラスである。

■クラス定義

```
package org.jtron.attach;

public class T_CMPF {
        public T_CMPF(int blkcnt, int blksz);
        public T_CMPF(int mpfatr, int blkcnt, int blksz);

        public int mpfatr;
        public int blkcnt;
        public int blksz;

        public static final int
                TA_TFIFO = 0x00,
                TA_TPRI  = 0x01;

        public static final int TSZ_MPF(int blkcnt,
                                        int blksz);

 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```

■コンストラクタ

```
public T_CMPF(int blkcnt, int blksz);
```

【パラメータ】

int	blkcnt	獲得できるメモリブロック数(個数)(★)
int	blksz	メモリブロックのサイズ(バイト数)(★)

【機能】

パラメータで指定される引数と同名の変数を設定する。
固定長メモリプール属性は、mpfatr=TA_TFIFOとする。

```
public T_CMPF(int mpfatr, int blkcnt, int blksz);
```

【パラメータ】

int	mpfatr	固定長メモリプール属性(★)
int	blkcnt	獲得できるメモリブロック数(個数)(★)
int	blksz	メモリブロックのサイズ(バイト数)(★)

【機能】

パラメータで指定される引数と同名の変数を設定する。

■メソッド

```
public static final int TSZ_MPF(int blkcnt, int blksz);
```

【パラメータ】

int	blkcnt	メモリブロック数(★)
int	blksz	メモリブロックサイズ(★)

【戻り値】

int	メモリプール領域のサイズ(目安のバイト数) (★)

【機能】

サイズがblkszバイトのメモリブロックをblkcnt個獲得できるのに必要な固定長メモリプール領域のサイズ(バイト数)を返す。

■変数

int	mpfatr	固定長メモリプール属性(★)
int	blkcnt	獲得できるメモリブロック数(個数) (★)
int	blksz	メモリブロックのサイズ(バイト数) (★)

■定数

TA_TFIFO	0x00	タスクの待ち行列を FIFO 順に
TA_TPRI	0x01	タスクの待ち行列をタスクの優先度順に

【JTRON1.0 仕様との相違】

μITRON4.0 仕様での変更に対応した。
フィールドを初期設定できるコンストラクタを追加する。

【補足説明】

メモリプール領域はブロック数とサイズのみを指定してカーネルの自動確保機能を用いる。

4.2.13.3 固定長メモリプール状態クラス(T_RMPF)

```
java.lang.Object
    |
    └── org.jtron.attach.T_RMPF
```

public class T_RMPF 標準仕様

固定長メモリプール状態のパケット形式クラス。

固定長メモリプールアタッチクラスの refer() で返される固定長メモリプール状態を表わすクラスである。

■クラス定義

```
package org.jtron.attach;

public class T_RMPF {
        public int wtskid;
        public int fblkcnt;

 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```

■変数

int	wtskid	固定長メモリプールの待ち行列の先頭のタスクの ID 番号(★)
int	fblkcnt	固定長メモリプールの空きメモリブロック数(個数) (★)

【JTRON1.0 仕様との相違】

μITRON4.0 仕様での変更に対応した。

4.2.13.4 固定長メモリブロックアタッチクラス (ItronFixedMemory)

```
org.jtron.ItronMemory
    │
    └──org.jtron.attach.ItronFixedMemory
```

public class ItronFixedMemory 標準仕様

固定長メモリブロックアタッチクラス。

固定長メモリブロックを操作するためのクラスである。

■クラス定義

```
package org.jtron.attach;

public class ItronFixedMemory extends ItronMemory {
        public int getPoolId();
        public void release();
}
```

■メソッド

public int getPoolId();

【戻り値】

int 固定長メモリプール ID(★)

【機能】

固定長メモリプールの ID を返す。

public void release() throws ItronCauseException;

【例外】

ItronCauseException ITRON による例外クラス

【機能】

rel_mpf サービスコール呼び出しに相当する。

【JTRON1.0 仕様との相違】

μITRON4.0 仕様での変更に対応した。
ITRON 互換 API クラスがなくなり、自分でインスタンス生成する必要がなくなったため、コンストラクタを削除した。

4.2.14 可変長メモリプール

ITRON の可変長メモリプールを操作するために以下のクラス群がある。

- 可変長メモリプールアタッチクラス
- 可変長メモリプール生成情報クラス
- 可変長メモリプール状態クラス
- 可変長メモリブロックアタッチクラス

可変長メモリプールアタッチクラスは ITRON の可変長メモリプールを表現する。

このクラスのインスタンスを生成することにより可変長メモリプールの生成ができるほか、生成済み可変長メモリプールに対応するインスタンスを得ることもできる。

このインスタンスのメソッドを呼び出すことによって可変長メモリブロックの獲得、返却などを行う。

獲得できる可変長メモリブロックは可変長メモリブロックアタッチクラスによって表現される。これはメモリ操作クラスを継承しているため、同様の方法で内容を操作することができる。

4.2.14.1 可変長メモリプールアタッチクラス

(VariableMemoryPool)

```
java.lang.Object
    |
    +--org.jtron.attach.VariableMemoryPool
```

public class VariableMemoryPool 標準仕様

可変長メモリプールアタッチクラス。

ITRON 可変長メモリプールを操作するためのクラスである。

■クラス定義

```
package org.jtron.attach;

public class VariableMemoryPool {
        public VariableMemoryPool(int mplid);
        public VariableMemoryPool(int mplid, T_CMPL pk_cmpl);
        public VariableMemoryPool(T_CMPL pk_cmpl);

        public int getId();

        public void delete();
        public ItronVariableMemory get(int blksz);
        public ItronVariableMemory poll(int blksz);
        public ItronVariableMemory get(int blksz, int tmout);
        public T_RMPL refer();
}
```

■コンストラクタ

```
public VariableMemoryPool(int mplid)
                                throws JtronException;
```

【パラメータ】

int	mplid	既存の接続対象の可変長メモリプールの ID 番号(★)

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたは JTRON による例外クラス)

【機能】

可変長メモリプール ID を指定して、既存の可変長メモリプールに接続するインスタンスを生成する。

```
public VariableMemoryPool(int mplid, T_CMPL pk_cmpl)
                        throws ItronCauseException;
```

【パラメータ】

int	mplid	生成対象の可変長メモリプールの ID 番号(★)
T_CMPL	pk_cmpl	可変長メモリプール生成情報クラス

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

cre_mpl サービスコール呼び出しに相当する。
可変長メモリプールを生成し、接続するインスタンスを生成する。

```
public VariableMemoryPool(T_CMPL pk_cmpl)
                        throws ItronCauseException;
```

【パラメータ】

T_CMPL	pk_cmpl	可変長メモリプール生成情報クラス

【例外】

ItronCauseException ITRON による例外クラス

【機能】

acre_mpl サービスコール呼び出しに相当する。
可変長メモリプールを生成し、接続するインスタンスを生成する。

■メソッド

```
public int getId();
```

【戻り値】

int 可変長メモリプール ID(★)

【機能】

接続している可変長メモリプールの可変長メモリプール ID を返す。

```
public void delete() throws ItronCauseException;
```

【例外】

ItronCauseException ITRON による例外クラス

【機能】

del_mpl サービスコール呼び出しに相当する。

```
public ItronVariableMemory get(int blksz)
                                throws JtronException;
```

【パラメータ】

int blksz 獲得するメモリブロックのサイズ(バイト数) (★)

【戻り値】

ItronVariableMemory 可変長メモリブロックアタッチクラス

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたはJTRONによる例外クラス)

【機能】

get_mplサービスコール呼び出しに相当する。

```
public ItronVariableMemory poll(int blksz)
                                    throws JtronException;
```

【パラメータ】

int	blksz	獲得するメモリブロックのサイズ(バイト数) (★)

【戻り値】

ItronVariableMemory	可変長メモリブロックアタッチクラス

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたはJTRONによる例外クラス)

【機能】

pget_mplサービスコール呼び出しに相当する。

```
public ItronVariableMemory get(int blksz, int tmout)
                                    throws JtronException;
```

【パラメータ】

int	blksz	獲得するメモリブロックのサイズ(バイト数) (★)
int	tmout	タイムアウト指定(単位：ms) (★)

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたはJTRON による例外クラス)

【戻り値】

ItronVariableMemory	可変長メモリブロックアタッチクラス

【機能】

tget_mpl サービスコール呼び出しに相当する。

```
public T_RMPL refer() throws JtronException;
```

【戻り値】

T_RMPL	可変長メモリプール状態クラス

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたはJTRON による例外クラス)

【機能】

ref_mpl サービスコール呼び出しに相当する。

【JTRON1.0 仕様との相違】

μITRON4.0 仕様での変更に対応した。

4.2.14.2 可変長メモリプール生成情報クラス(T_CMPL)

```
java.lang.Object
    |
    +--org.jtron.attach.T_CMPL
```

public class T_CMPL 標準仕様

可変長メモリプール生成情報のパケット形式クラス。

可変長メモリプール生成情報のパケット形式クラスは、可変長メモリプールアタッチクラスで可変長メモリプールを生成する際に用いるクラスである。

■クラス定義

```
package org.jtron.attach;

public class T_CMPL {
        public T_CMPL(int mplsz);
        public T_CMPL(int mplatr, int mplsz);

        public int mplatr;
        public int mplsz;

        public static final int
                TA_TFIFO = 0x00,
                TA_TPRI  = 0x01;

        public static finalint TSZ_MPL(int blkcnt, int blksz);

 // 下層のμITRON4.0 仕様 OS に合わせた実装独自のフィールド
}
```

■コンストラクタ

public T_CMPL(int mplsz);

【パラメータ】

int	mplsz	可変長メモリプール領域のサイズ(バイト数) (★)

【機能】

パラメータで指定される引数と同名の変数を設定する。
可変長メモリプール属性は、mplatr=TA_TFIFO とする。

```
public T_CMPL(int mplatr, int mplsz);
```

【パラメータ】

int	mplatr	可変長メモリプール属性(★)
int	mplsz	可変長メモリプール領域のサイズ(バイト数) (★)

【機能】

パラメータで指定される引数と同名の変数を設定する。

■メソッド

```
public static final int TSZ_MPL(int blkcnt, int blksz);
```

【パラメータ】

int	blkcnt	メモリブロック数(★)
int	blksz	メモリブロックサイズ(★)

【戻り値】

int	メモリプール領域のサイズ(目安のバイト数) (★)

【機能】

サイズがblkszバイトのメモリブロックを blkcnt個獲得できるのに

必要な可変長メモリプール領域のサイズ(目安のバイト数)を返す。

■変数

int	mplatr	可変長メモリプール属性(★)
int	mplsz	可変長メモリプール領域のサイズ(バイト数) (★)

■定数

TA_TFIFO	0x00	タスクの待ち行列を FIFO 順に
TA_TPRI	0x01	タスクの待ち行列をタスクの優先度順に

【JTRON1.0 仕様との相違】

μITRON4.0 仕様での変更に対応した。
フィールドを初期設定できるコンストラクタを追加する。

【補足説明】

メモリプール領域はサイズ（バイト数）のみを指定してカーネルの自動確保機能を用いる。

4.2.14.3 可変長メモリプール状態クラス(T_RMPL)

```
java.lang.Object
   |
   └── org.jtron.attach.T_RMPL
```

public class T_RMPL 標準仕様

可変長メモリプール状態のパケット形式クラス。

可変長メモリプールアタッチクラスの refer() で返される可変長メモリプール状態を表わすクラスである。

■クラス定義

```
package org.jtron.attach;

public class T_RMPL {
        public int wtskid;
        public int fmplsz;
        public int fblksz;

 // 下層のμITRON4.0 仕様 OS に合わせた実装独自のフィールド
}
```

■変数

int	wtskid	可変長メモリプールの待ち行列の先頭のタスクの ID 番号(★)
int	fmplsz	可変長メモリプールの空き領域の合計サイズ(バイト数) (★)
int	fblksz	すぐに獲得可能な最大メモリブロックサイズ(バイト数) (★)

【JTRON1.0 仕様との相違】

μITRON4.0 仕様での変更に対応した。

4.2.14.4 可変長メモリブロックアタッチクラス (ItronVariableMemory)

```
org.jtron.ItronMemory
   |
   └──org.jtron.attach.ItronVariableMemory
```

public class ItronVariableMemory 標準仕様

可変長メモリブロックアタッチクラス。

可変長メモリブロックを操作するためのクラスである。

■クラス定義

```
package org.jtron.attach;

public class ItronVariableMemory extends ItronMemory {
        public int getPoolId();
        public void release();
}
```

■メソッド

public int getPoolId();

【戻り値】

int	可変長メモリプール ID(★)

【機能】

可変長メモリプールの ID を返す。

public void release() throws ItronCauseException;

【例外】

ItronCauseException ITRON による例外クラス

【機能】

rel_mpl サービスコール呼び出しに相当する。

【JTRON1.0 仕様との相違】

μITRON4.0 仕様での変更に対応した。
ITRON 互換 API クラスがなくなり、自分でインスタンス生成する必要がなくなったため、コンストラクタを削除した。

4.2.15 周期ハンドラ

ITRON の周期ハンドラを操作するために以下のクラス群がある。

・周期ハンドラアタッチクラス
・周期ハンドラ生成情報クラス
・周期ハンドラ状態クラス

周期ハンドラアタッチクラスは ITRON の周期ハンドラを表現する。

このクラスのインスタンスを生成することにより周期ハンドラの生成ができるほか、生成済み周期ハンドラに対応するインスタンスを得ることもできる。

このインスタンスのメソッドを呼び出すことによって周期ハンドラの動作開始、動作停止などを行う。

4.2.15.1 周期ハンドラアタッチクラス(CyclicHandler)

```
java.lang.Object
    |
    +--org.jtron.attach.CyclicHandler
```

public class CyclicHandler 標準仕様

周期ハンドラアタッチクラス。

ITRON 周期ハンドラを操作するためのクラスである。

■クラス定義

```
package org.jtron.attach;

public class CyclicHandler {
        public CyclicHandler(int cycid);
        public CyclicHandler(int cycid, T_CCYC pk_ccyc);
        public CyclicHandler(T_CCYC pk_ccyc);

        public int getId();

        public void delete();
        public void start();
        public void stop();
        public T_RCYC refer();
}
```

■コンストラクタ

```
public CyclicHandler(int cycid) throws JtronException;
```

【パラメータ】

int	cycid	既存の接続対象の周期ハンドラの ID 番号(★)

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたは JTRON による例外クラス)

【機能】

周期ハンドラ ID を指定して、既存の周期ハンドラに接続するインスタンスを生成する。

```
public CyclicHandler(int cycid, T_CCYC pk_ccyc)
                        throws ItronCauseException;
```

【パラメータ】

int	cycid	生成対象の周期ハンドラの ID 番号(★)
T_CCYC	pk_ccyc	周期ハンドラ生成情報クラス

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

cre_cyc サービスコール呼び出しに相当する。
周期ハンドラを生成し、接続するインスタンスを生成する。

```
public CyclicHandler(T_CCYC pk_ccyc)
                        throws ItronCauseException;
```

【パラメータ】

T_CCYC	pk_ccyc	周期ハンドラ生成情報クラス

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

acre_cyc サービスコール呼び出しに相当する。
周期ハンドラを生成し、接続するインスタンスを生成する。

■メソッド

```
public int getId();
```

【戻り値】

int 周期ハンドラ ID(★)

【機能】

接続している周期ハンドラの周期ハンドラ ID を返す。

```
public void delete() throws ItronCauseException;
```

【例外】

ItronCauseException ITRON による例外クラス

【機能】

del_cyc サービスコール呼び出しに相当する。

```
public void start() throws ItronCauseException;
```

【例外】

ItronCauseException ITRON による例外クラス

【機能】

sta_cyc サービスコール呼び出しに相当する。

```
public void stop() throws ItronCauseException;
```

【例外】

ItronCauseException ITRON による例外クラス

【機能】

stp_cyc サービスコール呼び出しに相当する。

```
public T_RCYC refer() throws JtronException;
```

【戻り値】

T_RCYC 周期ハンドラ状態クラス

【例外】

JtronException JTRON 例外クラス(ITRON による例外クラスまたは JTRON による例外クラス)

【機能】

ref_cyc サービスコール呼び出しに相当する。

【JTRON1.0 仕様との相違】

μITRON4.0 仕様に対応した新クラスである。

4.2.15.2 周期ハンドラ生成情報クラス(T_CCYC)

```
java.lang.Object
    |
    └── org.jtron.attach.T_CCYC
```

public class T_CCYC 標準仕様

周期ハンドラ生成情報のパケット形式クラス。

周期ハンドラ生成情報のパケット形式クラスは、周期ハンドラアタッチクラスで、周期ハンドラを生成する際に用いるクラスである。

■クラス定義

```
package org.jtron.attach;

public class T_CCYC {
        public T_CCYC(String cychdr, int cyctim);
        public T_CCYC(int cycatr, int exinf,
                 String cychdr, int cyctim, int cycphs);

        public int cycatr;
        public int exinf;
        public String cychdr;
        public int cyctim;
        public int cycphs;

        public static final int
                 TA_HLNG   = 0x00,
                 TA_ASM    = 0x01;
        public static final int
                 TA_STA    = 0x02,
                 TA_PHS    = 0x04;

 // 下層のμITRON4.0 仕様 OS に合わせた実装独自のフィールド
}
```

■コンストラクタ

```
public T_CCYC(String cychdr, int cyctim);
```

【パラメータ】

String	cychdr	周期ハンドラの起動番地を表わす文字列(シンボル)
int	cyctim	周期ハンドラの起動周期(★)

【機能】

パラメータで指定される引数と同名の変数を設定する。
周期ハンドラ属性は cycatr=TA_HLNG、周期ハンドラの拡張情報は exinf=0 とする。
この時の周期ハンドラの起動位相は設定不要である。

```
public T_CCYC(int cycatr, int exinf,
              String cychdr, int cyctim, int cycphs);
```

【パラメータ】

int	cycatr	周期ハンドラ属性(★)
int	exinf	周期ハンドラの拡張情報(★)
String	cychdr	周期ハンドラの起動番地を表わす文字列(シンボル)
int	cyctim	周期ハンドラの起動周期(★)
int	cycphs	周期ハンドラの起動位相(★)

【機能】

パラメータで指定される引数と同名の変数を設定する。

■変数

int	cycatr	周期ハンドラ属性(★)
int	exinf	周期ハンドラの拡張情報(★)
String	cychdr	周期ハンドラの起動番地を表わす文

		字列(シンボル)
int	cyctim	周期ハンドラの起動周期(★)
int	cycphs	周期ハンドラの起動位相(★)

■定数

TA_HLNG	0x00	高級言語用のインタフェースで処理単位を起動
TA_ASM	0x01	アセンブリ言語用のインタフェースで処理単位を起動
TA_STA	0x02	周期ハンドラを動作している状態で生成
TA_PHS	0x04	周期ハンドラの位相を保存

【JTRON1.0 仕様との相違】

μITRON4.0 仕様に対応した新クラスである。

【補足説明】

周期ハンドラの起動番地（エントリアドレス）は文字列で表わされる。この文字列の解釈は実装定義であるが、例としては関数（サブルーチン、手続き）のシンボル名が想定される。

4.2.15.3 周期ハンドラ状態クラス(T_RCYC)

```
java.lang.Object
    |
    └──org.jtron.attach.T_RCYC
```

public class T_RCYC 標準仕様

周期ハンドラ状態のパケット形式クラス。

周期ハンドラアタッチクラスの refer() で返される周期ハンドラ状態を表わすクラスである。

■クラス定義

```
package org.jtron.attach;

public class T_RCYC {
        public int cycstat;
        public int lefttim;

        public static final int
                TCYC_STP = 0x00,
                TCYC_STA = 0x01;

 // 下層のμITRON4.0 仕様 OS に合わせた実装独自のフィールド
}
```

■変数

int	cycstat	周期ハンドラの動作状態(★)
int	lefttim	周期ハンドラを次に起動すべき時刻までの時間(単位：ms) (★)

■定数

TCYC_STP	0x00	周期ハンドラが動作していない
TCYC_STA	0x01	周期ハンドラが動作している

【JTRON1.0 仕様との相違】

μITRON4.0 仕様に対応した新クラスである。

4.2.16 アラームハンドラ

ITRON のアラームハンドラを操作するために以下のクラス群がある。

・アラームハンドラアタッチクラス
・アラームハンドラ生成情報クラス
・アラームハンドラ状態クラス

アラームハンドラアタッチクラスは ITRON のアラームハンドラを表現する。

このクラスのインスタンスを生成することによりアラームハンドラの生成ができるほか、生成済みアラームハンドラに対応するインスタンスを得ることもできる。

このインスタンスのメソッドを呼び出すことによってアラームハンドラの動作開始、動作停止などを行う。

4.2.16.1 アラームハンドラアタッチクラス(AlarmHandler)

```
java.lang.Object
    |
    +--org.jtron.attach.AlarmHandler
```

public class AlarmHandler 標準仕様

アラームハンドラアタッチクラス。

ITRON アラームハンドラを操作するためのクラスである。

■クラス定義

```
package org.jtron.attach;

public class AlarmHandler {
        public AlarmHandler(int almid);
        public AlarmHandler(int almid, T_CALM pk_calm);
        public AlarmHandler(T_CALM pk_calm);

        public int getId();

        public void delete();
        public void start(int almtim);
        public void stop();
        public T_RALM refer();
}
```

■コンストラクタ

public AlarmHandler(int almid) throws JtronException;

【パラメータ】

int	almid	既存の接続対象のアラームハンドラの ID 番号(★)

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたはJTRON による例外クラス)

【機能】

アラームハンドラ ID を指定して、既存のアラームハンドラに接続するインスタンスを生成する。

```
public AlarmHandler(int almid, T_CALM pk_calm)
                        throws ItronCauseException;
```

【パラメータ】

int	almid	生成対象のアラームハンドラの ID 番号(★)
T_CALM	pk_calm	アラームハンドラ生成情報クラス

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

cre_alm サービスコール呼び出しに相当する。
アラームハンドラを生成し、接続するインスタンスを生成する。

```
public AlarmHandler(T_CALM pk_calm)
                        throws ItronCauseException;
```

【パラメータ】

T_CALM	pk_calm	アラームハンドラ生成情報クラス

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

acre_alm サービスコール呼び出しに相当する。
アラームハンドラを生成し、接続するインスタンスを生成する。

■メソッド

```
public int getId();
```

【戻り値】

int	アラームハンドラ ID(★)

【機能】

接続しているアラームハンドラのアラームハンドラ ID を返す。

```
public void delete() throws ItronCauseException;
```

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

del_alm サービスコール呼び出しに相当する。

```
public void start(int almtim) throws ItronCauseException;
```

【パラメータ】

int	almtim	アラームハンドラの起動時刻(相対時間、単位:ms)(★)

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

sta_alm サービスコール呼び出しに相当する。

```
public void stop() throws ItronCauseException;
```

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

stp_alm サービスコール呼び出しに相当する。

```
public T_RALM refer() throws JtronException;
```

【戻り値】

T_RALM	アラームハンドラ状態クラス

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたは JTRON による例外クラス)

【機能】

ref_alm サービスコール呼び出しに相当する。

【JTRON1.0 仕様との相違】

μITRON4.0 仕様に対応した新クラスである。

4.2.16.2 アラームハンドラ生成情報クラス(T_CALM)

```
java.lang.Object
    |
    └──org.jtron.attach.T_CALM
```

public class T_CALM 標準仕様

アラームハンドラ生成情報のパケット形式クラス。

アラームハンドラ生成情報のパケット形式クラスは、アラームハンドラアタッチクラスで、アラームハンドラを生成する際に用いるクラスである。

■クラス定義

```
package org.jtron.attach;

public class T_CALM {
        public T_CALM(String almhdr);
        public T_CALM(int almatr, int exinf, String almhdr);

        public int almatr;
        public int exinf;
        public String almhdr;

        public static final int
                TA_HLNG = 0x00,
                TA_ASM  = 0x01;

 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```

■コンストラクタ

public T_CALM(String almhdr);

【パラメータ】

String	almhdr	アラームハンドラの起動番地を表わす文字列

【機能】

パラメータで指定される引数と同名の変数を設定する。
アラームハンドラ属性は almatr=TA_HLNG、アラームハンドラの拡張情報は exinf=0 とする。

```
public T_CALM(int almatr, int exinf, String almhdr);
```

【パラメータ】

int	almatr	アラームハンドラ属性(★)
int	exinf	アラームハンドラの拡張情報(★)
String	almhdr	アラームハンドラの起動番地を表わす文字列

【機能】

パラメータで指定される引数と同名の変数を設定する。

■変数

int	almatr	アラームハンドラ属性(★)
int	exinf	アラームハンドラの拡張情報(★)
String	almhdr	アラームハンドラの起動番地を表わす文字列

■定数

TA_HLNG	0x00	高級言語用のインタフェースで処理単位を起動
TA_ASM	0x01	アセンブリ言語用のインタフェースで処理単位を起動

【JTRON1.0 仕様との相違】

μITRON4.0 仕様に対応した新クラスである。

【補足説明】

アラームハンドラの起動番地（エントリアドレス）は文字列で表わされる。この文字列の解釈は実装定義であるが、例としては関数（サブルーチン、手続き）のシンボル名が想定される。

4.2.16.3 アラームハンドラ状態クラス(T_RALM)

```
java.lang.Object
    |
    └──org.jtron.attach.T_RALM
```

public class T_RALM **標準仕様**

アラームハンドラ状態のパケット形式クラス。

アラームハンドラアタッチクラスの refer( ) で返されるアラームハンドラ状態を表わすクラスである。

■クラス定義

```
package org.jtron.attach;

public class T_RALM {
        public int almstat;
        public int lefttim;

        public static final int
                TALM_STP = 0x00,
                TALM_STA = 0x01;

 // 下層のμITRON4.0 仕様 OS に合わせた実装独自のフィールド
}
```

■変数

int	almstat	アラームハンドラの動作状態(★)
int	lefttim	アラームハンドラの起動時刻までの時間(★)

■定数

TALM_STP	0x00	アラームハンドラが動作していない
TALM_STA	0x01	アラームハンドラが動作している

【JTRON1.0 仕様との相違】

μITRON4.0 仕様に対応した新クラスである。

4.2.17 割込みサービスルーチン

ITRON の割込みサービスルーチン操作するために以下のクラス群がある。

・割込みサービスルーチンアタッチクラス
・割込みサービスルーチン生成情報クラス
・割込みサービスルーチン状態クラス

割込みサービスルーチンアタッチクラスは ITRON の割込みサービスルーチンを表現する。

このクラスのインスタンスを生成することにより割込みサービスルーチンの生成ができるほか、生成済み割込みサービスルーチンに対応するインスタンスを得ることもできる。

このインスタンスのメソッドを呼び出すことによって割込みサービスルーチンの状態参照、削除などを行う。

4.2.17.1 割込みサービスルーチンアタッチクラス (InterruptServiceRoutine)

```
java.lang.Object
    |
    +--org.jtron.attach.InterruptServiceRoutine
```

public class InterruptServiceRoutine 拡張仕様

割込みサービスルーチンアタッチクラス。

ITRON 割込みサービスルーチンを操作するためのクラスである。

■クラス定義

```
package org.jtron.attach;

public class InterruptServiceRoutine {
        public InterruptServiceRoutine(int isrid);
        public InterruptServiceRoutine(int isrid,
                                T_CISR pk_cisr);
        public InterruptServiceRoutine(T_CISR pk_cisr);

        public int getId();

        public void delete();
        public T_RISR refer();
}
```

■コンストラクタ

```
public InterruptServiceRoutine(int isrid)
                                throws JtronException;
```

【パラメータ】

int	isrid	既存の接続対象の割込みサービスル

ーチンの ID 番号(★)

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたは JTRON による例外クラス)

【機能】

割込みサービスルーチン ID を指定して、既存の割込みサービスルーチンに接続するインスタンスを生成する。

```
public InterruptServiceRoutine(int isrid,
        T_CISR pk_cisr) throws ItronCauseException;
```

【パラメータ】

int	isrid	生成対象の割込みサービスルーチンの ID 番号(★)
T_CISR	pk_cisr	割込みサービスルーチン生成情報クラス

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

cre_isr サービスコール呼び出しに相当する。
割込みサービスルーチンを生成し、接続するインスタンスを生成する。

```
public InterruptServiceRoutine(T_CISR pk_cisr)
                        throws ItronCauseException;
```

【パラメータ】

T_CISR	pk_cisr	割込みサービスルーチン生成情報クラス

【例外】

ItronCauseException ITRON による例外クラス

【機能】

acre_isr サービスコール呼び出しに相当する。
割込みサービスルーチンを生成し、接続するインスタンスを生成する。

■メソッド

```
public int getId();
```

【戻り値】

int サービスルーチン ID(★)

【機能】

接続しているサービスルーチンのサービスルーチン ID を返す。

```
public void delete() throws ItronCauseException;
```

【例外】

ItronCauseException ITRON による例外クラス

【機能】

del_isr サービスコール呼び出しに相当する。

```
public T_RISR refer() throws JtronException;
```

【戻り値】

T_RISR 割込みサービスルーチン状態クラス

【例外】

JtronException JTRON 例外クラス(ITRON による例外クラスまたは JTRON による例外クラス)

【機能】

ref_isr サービスコール呼び出しに相当する。

【JTRON1.0 仕様との相違】

μITRON4.0 仕様に対応した新クラスである。

4.2.17.2 割込みサービスルーチン生成情報クラス(T_CISR)

```
java.lang.Object
    |
    └── org.jtron.attach.T_CISR
```

public class T_CISR 拡張仕様

割込みサービスルーチン生成情報のパケット形式クラス。

割込みサービスルーチン生成情報のパケット形式クラスは、割込みサービスルーチンアタッチクラスで、割込みサービスルーチンを生成する際に用いるクラスである。

■クラス定義

```
package org.jtron.attach;

public class T_CISR {
        public T_CISR(int intno, String isr);
        public T_CISR(int isratr, int exinf,
                                  int intno, String isr);

        public int isratr;
        public int exinf;
        public int intno;
        public String isr;

        public static final int
                TA_HLNG = 0x00,
                TA_ASM  = 0x01;

 // 下層のμITRON4.0 仕様 OS に合わせた実装独自のフィールド
}
```

■コンストラクタ

```
public T_CISR(int intno, String isr);
```

【パラメータ】

int	intno	割込みサービスルーチンを付加する割込み番号(★)
String	isr	割込みサービルルーチンの起動番地を表わす文字列

【機能】

パラメータで指定される引数と同名の変数を設定する。
割込みサービスルーチン属性は isratr=TA_HLNG、割込みサービスルーチンの拡張情報は exinf=0 とする。

```
public T_CISR(int isratr, int exinf, int intno, String isr);
```

【パラメータ】

int	isratr	割込みサービスルーチン属性(★)
int	exinf	割込みサービスルーチンの拡張情報(★)
int	intno	割込みサービスルーチンを付加する割込み番号(★)
String	isr	割込みサービルルーチンの起動番地を表わす文字列

【機能】

パラメータで指定される引数と同名の変数を設定する。

■変数

int	isratr	割込みサービスルーチン属性(★)
int	exinf	割込みサービスルーチンの拡張情報(★)
int	intno	割込みサービスルーチンを付加する割込み番号(★)
String	isr	割込みサービルルーチンの起動番地

を表わす文字列

■定数

TA_HLNG	0x00	高級言語用のインタフェースで処理単位を起動
TA_ASM	0x01	アセンブリ言語用のインタフェースで処理単位を起動

【JTRON1.0 仕様との相違】

μITRON4.0 仕様に対応した新クラスである。

【補足説明】

割込みサービスルーチンの起動番地（エントリアドレス）は文字列で表わされる。この文字列の解釈は実装定義であるが、例としては関数（サブルーチン、手続き）のシンボル名が想定される。

4.2.17.3 割込みサービスルーチン状態クラス(T_RISR)

```
java.lang.Object
    |
    └── org.jtron.attach.T_RISR
```

public class T_RISR 拡張仕様

割込みサービスルーチン状態のパケット形式クラス。

割込みサービスルーチンアタッチクラスの refer() で返されるアラームハンドラ状態を表わすクラスである。

■クラス定義

```
package org.jtron.attach;

public class T_RISR {

 // 下層のμITRON4.0 仕様 OS に合わせた実装独自のフィールド
}
```

【JTRON1.0 仕様との相違】

μITRON4.0 仕様に対応した新クラスである。

4.2.18 その他

4.2.18.1 カーネルアタッチクラス(Kernel)

```
java.lang.Object
    |
    └──org.jtron.attach.Kernel
```

public class Kernel **標準仕様**

カーネルアタッチクラス。

ITRON の機能で、操作する対象（インスタンス）がないものをまとめたカーネルアタッチクラスである。

■クラス定義

```
package org.jtron.attach;

public class Kernel {
    // システム時刻管理
        public static void setTime(int systim);
        public static int getTime();

    // オーバーランハンドラの定義
        public static void defineOverrunHandler(
                                T_DOVR pk_dovr);

    // システム状態管理
        public static void rotateReadyQueue(int tskpri);
        public static int getRunningTaskID();
        public static T_RSYS referSystem();

    // 割込み管理                                    拡張仕様
        public static void defineInterruptHandler(
                        int inhno, T_DINH pk_dinh);
        public static void disableInterrupt(int intno);
```

```
        public static void enableInterrupt(int intno);

    // サービスコール管理                                    拡張仕様
        public static void defineServiceCall(int fncd,
                                        T_DSVC pk_dsvc);
        public static int callServiceCall(int fncd,
                                        int[] parm);

    // システム構成管理
        public static void defineCPUExceptionHandler
                (int excno, T_DEXC pk_dexc); //  拡張仕様
        public static T_RCFG referConfiguration();
        public static T_RVER referVersion();

    // 定数
        public static final int
                TSK_SELF= 0,
                TSK_NONE= 0,

                TPRI_SELF  = 0,
                TPRI_INI  = 0;

        public static final int
                TMO_POL  = 0,
                TMO_FEVR= -1,
                TMO_NBLK= -2;

    // 構成定数
        public static final int
                TMIN_TPRI  = 1,
                TMAX_TPRI,

                TMIN_MPRI  = 1,
                TMAX_MPRI,

                TKERNEL_MAKER,
                TKERNEL_PRID,
```

```
                TKERNEL_SPVER,
                TKERNEL_PRVER,

                TMAX_ACTCNT,
                TMAX_WUPCNT,
                TMAX_SUSCNT,

                TBIT_TEXPTN,
                TBIT_FLGPTN,
                TBIT_RDVPTN,

                TIC_NUME,
                TIC_DENO,

                TMAX_MAXSEM;
}
```

■メソッド

システム時刻管理

```
public static void setTime(int systim)
                        throws ItronCauseException;
```

【パラメータ】

int	systim	システム時刻に設定する時刻(単位:ms)(★)

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

set_tim サービスコール呼び出しに相当する。

【補足説明】

Java 実行環境に悪影響を与える恐れがあるため、JtronCauseException 例外を発生させるように実装定義しても良い。

```
public static int getTime();
```

【戻り値】

int	現在のシステム時刻(単位:ms)(★)

【機能】

get_tim サービスコール呼び出しに相当する。

●オーバーランハンドラの定義

```
public static void defineOverrunHandler(T_DOVR pk_dovr)
                        throws ItronCauseException;
```

【パラメータ】

T_DOVR	pk_dovr	オーバーランハンドラ定義情報クラス

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

def_ovr サービスコール呼び出しに相当する。
引数に null を指定すると解除する。

●システム状態管理

```
public static void rotateReadyQueue(int tskpri)
                        throws ItronCauseException;
```

【パラメータ】

int	tskpri	優先順位を回転する対象の優先度(★)

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

rot_rdq サービスコール呼び出しに相当する。

```
public static int getRunningTaskID();
```

【戻り値】

int	実行状態のタスクの ID 番号(★)

【機能】

get_tid サービスコール呼び出しに相当する。

```
public static T_RSYS referSystem() throws JtronException;
```

【戻り値】

T_RSYS	システム状態クラス

【例外】

JtronException	JTRON 例外クラス(ITRON による例外クラスまたは JTRON による例外クラス)

【機能】

ref_sys サービスコール呼び出しに相当する。

割込み管理 **拡張仕様**

```
public static void defineInterruptHandler(int inhno,
        T_DINH pk_dinh) throws ItronCauseException;
```

【パラメータ】

int	inhno	定義対象の割込みハンドラ番号(★)
T_DINH	pk_dinh	割込みハンドラ定義情報クラス

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

def_inh サービスコール呼び出しに相当する。
引数 pk_dinh に null を指定すると解除する。

【補足説明】

拡張仕様なのでこの機能を実現しなくてもよい。実現しない場合の方法は実装定義とするが、「メソッドを実装しない」または「呼び出すと JtronCauseException 例外を発生させる」のどちらかとする。

```
public static void disableInterrupt(int intno)
                        throws ItronCauseException;
```

【パラメータ】

int	intno	割込みを禁止する割込み番号(★)

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

dis_int サービスコール呼び出しに相当する。

【補足説明】

拡張仕様なのでこの機能を実現しなくてもよい。実現しない場合の方法は実装定義とするが、「メソッドを実装しない」または「呼び出すと JtronCauseException 例外を発生させる」のどちらかとする。

```
public static void enableInterrupt(int intno)
                        throws ItronCauseException;
```

【パラメータ】

int	intno	割込み許可する割込み番号(★)

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

ena_int サービスコール呼び出しに相当する。

【補足説明】

拡張仕様なのでこの機能を実現しなくてもよい。実現しない場合の方法は実装定義とするが、「メソッドを実装しない」または「呼び出すと JtronCauseException 例外を発生させる」のどちらかとする。

●サービスコール管理

```
public static void defineServiceCall(int fncd,
        T_DSVC pk_dsvc) throws ItronCauseException;
```

【パラメータ】

int	fncd	定義対象の機能コード(★)
T_DSVC	pk_dsvc	拡張サービスコール定義情報クラス

【例外】

ItronCauseException	ITRON による例外クラス

【機能】

def_svc サービスコール呼び出しに相当する。
引数 pk_dsvc に null を指定すると解除する。

【補足説明】

拡張仕様なのでこの機能を実現しなくてもよい。実現しない場合の方法は実装定義とするが、「メソッドを実装しない」または「呼び出すと JtronCauseException 例外を発生させる」のどちらかとする。

```
public static int callServiceCall(int fncd, int[] parm)
                        throws ItronCauseException;
```

【パラメータ】

int	fncd	呼び出すサービスコールの機能コード(★)
int[]	parm	サービスコールへの引数パラメータ(★)

【例外】

ItronCauseException　ITRON による例外クラス

【機能】

cal_svc サービスコール呼び出しに相当する。

【補足説明】

拡張仕様なのでこの機能を実現しなくてもよい。実現しない場合の方法は実装定義とするが、「メソッドを実装しない」または「呼び出すと JtronCauseException 例外を発生させる」のどちらかとする。例外発生時は、呼び出そうとしたサービスコールの機能コード fncd が例外の変数 functionCode に格納される。

●システム構成管理

```
public static void defineCPUExceptionHandler
              (int excno, T_DEXC pk_dexc)
                      throws ItronCauseException;
                                    // 拡張仕様
```

【パラメータ】

int	excno	定義対象の CPU 例外ハンドラ番号(★)
T_DEXC	pk_dexc	CPU 例外ハンドラ定義情報クラス

【例外】

ItronCauseException ITRON による例外クラス

【機能】

def_exc サービスコール呼び出しに相当する。

【補足説明】

拡張仕様なのでこの機能を実現しなくてもよい。実現しない場合の方法は実装定義とするが、「メソッドを実装しない」または「呼び出すと JtronCauseException 例外を発生させる」のどちらかとする。

```
public static T_RCFG referConfiguration()
                                    throws JtronException;
```

【戻り値】

T_RCFG コンフィギュレーション情報クラス

【例外】

JtronException JTRON 例外クラス(ITRON による例外クラスまたは JTRON による例外クラス)

【機能】

ref_cfg サービスコール呼び出しに相当する。

```
public static T_RVER referVersion()
                                    throws JtronException;
```

【戻り値】

T_RVER バージョン情報クラス

【例外】

JtronException JTRON 例外クラス(ITRON による例外クラスまたは JTRON による例外クラス)

【機能】

ref_ver サービスコール呼び出しに相当する。

■定数

TSK_SELF	0	自タスク指定
TSK_NONE	0	該当するタスクが無い
TPRI_SELF	0	自タスクのベース優先度の指定
TPRI_INI	0	タスクの起動時優先度の指定
TMO_POL	0	ポーリング
TMO_FEVR	-1	永久待ち
TMO_NBLK	-2	ノンブロッキング
TMIN_TPRI	1	タスク優先度の最小値(=1)
TMAX_TPRI		タスク優先度の最大値
TMIN_MPRI	1	メッセージ優先度の最小値(=1)
TMAX_MPRI		メッセージ優先度の最大値
TKERNEL_MAKER		カーネルのメーカーコード
TKERNEL_PRID		カーネルの識別番号
TKERNEL_SPVER		ITRON 仕様のバージョン
TKERNEL_PRVER		カーネルのバージョン番号
TMAX_ACTCNT		タスクの起動要求キューイング数の最大値
TMAX_WUPCNT		タスクの起床要求キューイング数の最大値
TMAX_SUSCNT		タスクの強制待ち要求ネスト数の最大値
TBIT_TEXPTN		タスク例外要求のビット数
TBIT_FLGPTN		イベントフラグのビット数
TBIT_RDVPTN		ランデブ条件のビット数

TIC_NUME	タイムティックの周期の分子
TIC_DENO	タイムティックの周期の分母
TMAX_MAXSEM	セマフォの最大資源数の最大値

【補足説明】

定数値が記載されていないものは、クラスロード時に定義されるものとする。

【JTRON1.0 仕様との相違および仕様決定の理由】

・ほとんど使われることがないようなので、ITRON 互換 API クラス（ItronApi）を廃止した。廃止することにより、従来は長さがゼロの ItronMemory インスタンスを生成して get_blk()に渡し、得られたブロックを ItronVariableMemory インスタンスとして再生成しなければならないなどという、非常に使いにくい設計になっていたことを解消できる。

ITRON 互換 API クラスにのみ存在した機能、つまり他のクラスで相当機能がないもの（対象になるオブジェクトのないサービスコール）を実行するためのものが本クラスである。

・割込み管理機能のうち、chg_ix*x* サービスコールと get_ix*x* サービスコールは CPU による差異が非常に大きいため仕様として規定しない。ただしこれは実装定義機能として用意することを禁止するものではない。

・ディスパッチ制御、CPU ロック関連機能は Java 実行環境に悪影響を及ぼす可能性が高いので仕様に含めないこととした。

4.2.18.2 オーバーランハンドラ定義情報クラス(T_DOVR)

```
java.lang.Object
    |
    └── org.jtron.attach.T_DOVR
```

public class T_DOVR 標準仕様

オーバーランハンドラ定義情報のパケット形式クラス。

カーネルアタッチクラスの defineOverrunHandler() でオーバーランハンドラを定義する際に用いるクラスである。

■クラス定義

```
package org.jtron.attach;

public class T_DOVR {
        public T_DOVR(String ovrhdr);
        public T_DOVR(int ovratr, String ovrhdr);

        public int ovratr;
        public String ovrhdr;

        public static final int
                TA_HLNG = 0x00,
                TA_ASM  = 0x01;

 // 下層のμITRON4.0 仕様 OS に合わせた実装独自のフィールド
}
```

■コンストラクタ

public T_DOVR(String ovrhdr);

【パラメータ】

String	ovrhdr	オーバーランハンドラの起動番地を表わす文字列

【機能】

パラメータで指定される引数と同名の変数を設定する。
引数に無いオーバーランハンドラ属性は、属性=TA_HLNG とする。

```
public T_DOVR(int ovratr, String ovrhdr);
```

【パラメータ】

int	ovratr	オーバーランハンドラ属性(★)
String	ovrhdr	オーバーランハンドラの起動番地を表わす文字列

【機能】

パラメータで指定される引数と同名の変数を設定する。

■変数

int	ovratr	オーバーランハンドラ属性(★)
String	ovrhdr	オーバーランハンドラの起動番地を表わす文字列

■定数

TA_HLNG	0x00	高級言語用のインタフェースで処理単位を起動
TA_ASM	0x01	アセンブリ言語用のインタフェースで処理単位を起動

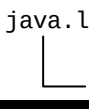
【JTRON1.0 仕様との相違】

μITRON4.0 仕様に対応した新クラスである。

【補足説明】

ハンドラの起動番地（エントリアドレス）は文字列で表わされる。この文字列の解釈は実装定義であるが、例としては関数（サブルーチン、手続き）のシンボル名が想定される。

4.2.18.3 システム状態クラス(T_RSYS)



public class T_RSYS **標準仕様**

システム状態のパケット形式クラス。

カーネルアタッチクラスの `referSystem()` **で返されるシステム状態を表わすクラスである。**

■クラス定義

```
package org.jtron.attach;

public class T_RSYS {

 // 下層のμITRON4.0 仕様 OS に合わせた実装独自のフィールド
}
```

【JTRON1.0 **仕様との相違】**

μITRON4.0 **仕様に対応した新クラスである。**

4.2.18.4 割込みハンドラ定義情報クラス(T_DINH)

```
java.lang.Object
    |
    └── org.jtron.attach.T_DINH
```

public class T_DINH 拡張仕様

割込みハンドラ定義情報のパケット形式クラス。

カーネルアタッチクラスの defineInterruptHandler() で割込みハンドラを定義する際に用いるクラスである。

■クラス定義

```
package org.jtron.attach;

public class T_DINH {
        public T_DINH(int inhatr, String inhhdr);

        public int inhatr;
        public String inhhdr;

 // 下層のμITRON4.0 仕様 OS に合わせた実装独自のフィールド
}
```

■コンストラクタ

public T_DINH(int inhatr, String inhhdr);

【パラメータ】

int	inhatr	割込みハンドラ属性(実装定義) (★)
String	inhhdr	割込みハンドラの起動番地を表わす文字列

【機能】

パラメータで指定される引数と同名の変数を設定する。

■変数

int	inhatr	割込みハンドラ属性(実装定義) (★)
String	inhhdr	割込みハンドラの起動番地を表わす文字列

【JTRON1.0 仕様との相違】

μITRON4.0 仕様に対応した新クラスである。

【補足説明】

ハンドラの起動番地（エントリアドレス）は文字列で表わされる。この文字列の解釈は実装定義であるが、例としては関数（サブルーチン、手続き）のシンボル名が想定される。

4.2.18.5 拡張サービスコール定義情報クラス(T_DSVC)

```
java.lang.Object
    |
    └── org.jtron.attach.T_DSVC
```

public class T_DSVC 拡張仕様

拡張サービスコール定義情報のパケット形式クラス。

カーネルアタッチクラスの defineServiceCall() **で拡張サービスコールを定義する際に用いるクラスである。**

■クラス定義

```
package org.jtron.attach;

public class T_DSVC {
        public T_DSVC(String svcrtn);
        public T_DSVC(int svcatr, String svcrtn);

        public int svcatr;
        public String svcrtn;

        public static final int
                TA_HLNG = 0x00,
                TA_ASM  = 0x01;

 // 下層のμITRON4.0 仕様 OS に合わせた実装独自のフィールド
}
```

■コンストラクタ

public T_DSVC(String svcrtn);

【パラメータ】

String	svcrtn	拡張サービスコールルーチンの起動番地を表わす文字列

【機能】

パラメータで指定される引数と同名の変数を設定する。
引数に無い拡張サービスコール属性は、属性=TA_HLNG とする。

```
public T_DSVC(int svcatr, String svcrtn);
```

【パラメータ】

int	svcatr	拡張サービスコール属性(★)
String	svcrtn	拡張サービスコールルーチンの起動番地を表わす文字列

【機能】

パラメータで指定される引数と同名の変数を設定する。

■変数

int	svcatr	拡張サービスコール属性(★)
String	svcrtn	拡張サービスコールルーチンの起動番地を表わす文字列

■定数

TA_HLNG	0x00	高級言語用のインタフェースで処理単位を起動
TA_ASM	0x01	アセンブリ言語用のインタフェースで処理単位を起動

【JTRON1.0 仕様との相違】

μITRON4.0 仕様に対応した新クラスである。

【補足説明】

ルーチンの起動番地（エントリアドレス）は文字列で表わされる。この文字列の解釈は実装定義であるが、例としては関数（サブルーチン、手続き）のシンボル名が想定される。

4.2.18.6 CPU 例外ハンドラ定義情報クラス(T_DEXC)

```
java.lang.Object
    |
    └── org.jtron.attach.T_DEXC
```

public class T_DEXC 拡張仕様

CPU 例外ハンドラ定義情報のパケット形式クラス。

カーネルアタッチクラスの defineCPUExceptionHandler() で CPU 例外ハンドラを定義する際に用いるクラスである。

■クラス定義

```
package org.jtron.attach;

public class T_DEXC {
        public T_DEXC(int excatr, String exchdr);

        public int excatr;
        public String exchdr;

 // 下層のμITRON4.0 仕様 OS に合わせた実装独自のフィールド
}
```

■コンストラクタ

public T_DEXC(int excno, String exchdr);

【パラメータ】

int	excatr	CPU 例外ハンドラ属性(実装定義)(★)
String	exchdr	CPU 例外ハンドラの起動番地を表わす文字列

【機能】

パラメータで指定される引数と同名の変数を設定する。

■変数

int	excatr	CPU 例外ハンドラ属性(実装定義)(★)
String	exchdr	CPU 例外ハンドラの起動番地を表わす文字列

【JTRON1.0 仕様との相違】

μITRON4.0 仕様に対応した新クラスである。

【補足説明】

ルーチンの起動番地（エントリアドレス）は文字列で表わされる。この文字列の解釈は実装定義であるが、例としては関数（サブルーチン、手続き）のシンボル名が想定される。

4.2.18.7 コンフィグレーション情報クラス(T_RCFG)

```
java.lang.Object
    |
    └──org.jtron.attach.T_RCFG
```

public class T_RCFG 標準仕様

コンフィグレーション情報のパケット形式クラス。

カーネルアタッチクラスの `referConfiguration()` **で返されるコンフィグレーション情報を表わすクラスである。**

■クラス定義

```
package org.jtron.attach;

public class T_RCFG {

 // 下層のμITRON4.0 仕様 OS に合わせた実装独自のフィールド
}
```

【JTRON1.0 **仕様との相違】**

μITRON4.0 **仕様に対応した新クラスである。**

4.2.18.8 バージョン番号クラス(T_RVER)

```
java.lang.Object
    |
    └── org.jtron.T_RVER
```

public class T_RVER **標準仕様**

バージョン情報のパケット形式クラス。

カーネルアタッチクラスの `referVersion()` **で返されるバージョン情報を表わすクラスである。**

■クラス定義

```
package org.jtron.attach;

public class T_RVER {
        public int maker;
        public int prid;
        public int spver;
        public int prver;
        public int prno0,
                   prno1,
                   prno2,
                   prno3;
}
```

■変数

int	maker	カーネルのメーカーコード(★)
int	prid	カーネルの識別番号(★)
int	spver	ITRON 仕様のバージョン番号(★)
int	prver	カーネルのバージョン番号(★)
int	prno0～3	カーネル製品の管理情報(★)

【JTRON1.0 仕様との相違】

μITRON4.0 仕様に対応した新クラスである。

第 5 章 タイプ 2 インタフェース

(共有オブジェクト/スレッド操作 API)

5.1 概要

タイプ 2 インタフェースは共有オブジェクトインタフェースとスレッド操作 API から構成される。

共有オブジェクトインタフェースは、Java プログラムとリアルタイムタスク間でデータの交換を行うことによる通信手段を提供する(図 5.1)。
Java プログラムでは共有対象となるオブジェクト作成時にリアルタイムタスクから利用できるようにする。このオブジェクトを共有オブジェクトと呼ぶ。リアルタイムタスクでは Java プログラムで作成された、共有オブジェクトのデータを操作する。このようにして共有オブジェクトを利用して Java プログラムとリアルタイムタスクでデータの交換を行う。
共有オブジェクトの一貫性を保持するためにロック機能を設ける。
図 5.1 にあるように共有オブジェクトでは幾つかのデータ型が利用可能である。リアルタイムタスクの API では型毎に API が異なるので、API 名を<type>で代表させている。

スレッド操作 API はリアルタイムタスクから Java スレッドを制御するものである。スレッド操作 API は、リアルタイムタスクから共有オブジェクトに対して操作を行なう場合に、Java スレッドの一時停止、停止解除を行なうためのものである。

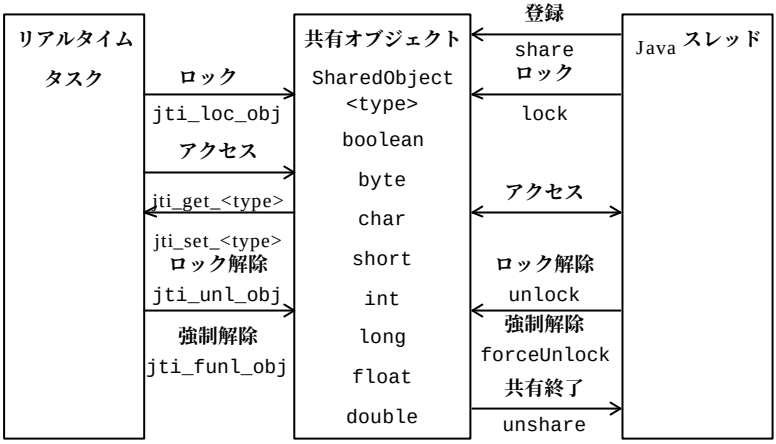


図 5.1:共有オブジェクト

Java プログラムでは、Java のクラスライブラリとして提供する SharedObject クラスを継承する、あるいは Sharable インタフェースを実装することにより、リアルタイムタスク側とやりとりを行う共有オブジェクトを生成することができる。本共有オブジェクト生成時に名前を付与して共有オブジェクトの識別に利用する。 Java では複数のオブジェクトに同一の名前を付与することが可能であるが、共有オブジェクトに対する名前はシステム上の全て Java プログラムにおいて一意なものとしている。共有対象となるオブジェクトを登録する時点でシステム側で一意性をチェックする。
共有オブジェクトアクセス時は、ロック、ロック解除(Java の場合 SharedObject の lock メソッド、unlock メソッド、ITRON の場合は、jti_loc_obj、jti_unl_obj API)を行い排他制御する。リアルタイムタスクからは共有オブジェクトをロックしたあとアクセス(jti_get_<type>、jti_set_<type> API)し、アクセス完了後アンロックする。Java スレッドから unshare メソッドを実行することにより、共有オブジェクトの共有を終了させることができる。

ロックのセマンティックス

表 5,1 に示すように共有オブジェクトには、共有状態、未共有状態の 2 状態があり、共有状態はさらに、ロックされた状態、ロックのない状態がある。ロックされた状態には、Java スレッド、またはリアルタイムタスクがロック、ロック解除を実行した場合の状態遷移を示す。
「同ロックオーナ」は直前にロック操作を行ったタスク・スレッドと現在の操作を行うスレッドまたはリアルタイムタスクが同一の場合を表わす。「異ロックオーナ(Java スレッド)」は直前にロック操作を行ったスレッドと現在の操作を行うスレッドまたはリアルタイムタスク、「異ロックオーナ(リアルタイムタスク)」は直前にロック操作を行ったタスクと現在の操作を行うスレッドまたはリアルタイムタスクが異なる場合をそれぞれ意味する。各々の状態に対して操作を行なった場合の動作を表 5.1 に示す。

Java スレッド、リアルタイムタスク共に、共有オブジェクトのロック時の待ち順序は実装定義である。

表 5.1:共有オブジェクトのロックの状態遷移

操作		共有				未共有
		ロック無し	ロックあり			
			同ロックオーナ	異ロックオーナ (Java スレッド)	異ロックオーナ (リアルタイムタスク)	
Java	lock	OK (ロック)	OK (no effect)	ブロックされる	ブロックされる	例外発生
	unlock	OK (no effect)	OK (ロック解除)	例外発生	例外発生	例外発生
	forceUnlock	OK (no effect)	OK (ロック解除)	OK (ロック解除)	OK (no effect)	例外発生
	unshare	OK (共有終了)	*1	*2	*2	例外発生
ITRON API	jti_loc_obj	OK (ロック)	OK (no effect)	待ちに入る	待ちに入る	E_OBJ エラー
	jti_unl_obj	OK (no effect)	OK (ロック解除)	E_OBJ エラー	E_OBJ エラー	E_OBJ エラー
	jti_funl_obj	OK (no effect)	OK (ロック解除)	OK (ロック解除)	OK (ロック解除)	E_OBJ エラー

*1 ロックを解除した後、オブジェクトの共有を終了する。
*2 他スレッド・タスクがロックを解除するまで待ちに入る。ロック解除後共有を終了する。

【補足説明】

●期待する操作手順

図 5.2 の順序で操作することを期待している。

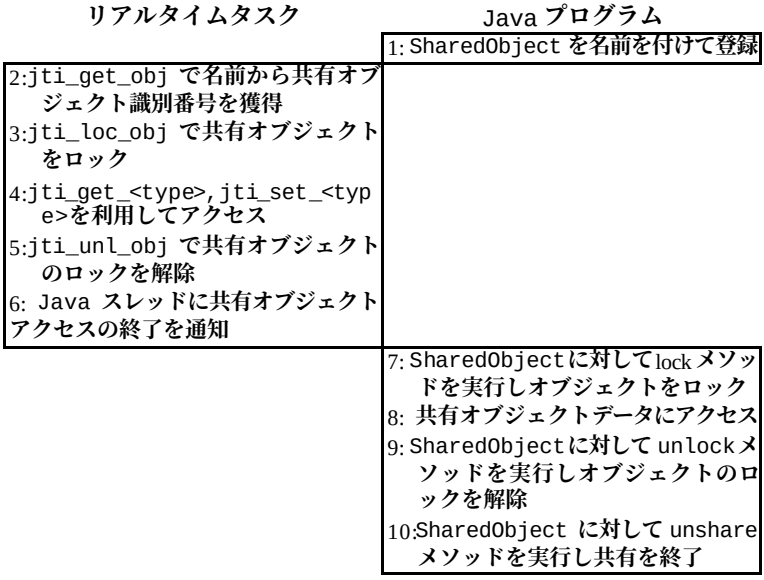


図 5.2:期待する操作順序

共有される Java オブジェクトに関する仮定

jti_loc_obj と jti_unl_obj の実行する間に一般にどのようなサービスコールでも実行することができ、従って任意のタスク状態に移行することができる。しかし、ロック中のタスクが RUNNING/READY 以外の状態になるとアンロックできなくなる可能性がある。従って、ロック中のタスクは RUNNING/READY 以外の状態にならないようにするのが望ましい。

try_lock の機能(ロックできる場合、ロックする。ロックできない場合にエラー、もしくは例外を発生する)はロック時に待ち時間 0 を指定する

ことで実現できる。

Java スレッド A で lock メソッド実行後、他のスレッド B から stop メソッドにより ThreadDeath 例外が発生した場合、スレッド A ではこの例外を処理する finally 文で unlock メソッドを実行してロックを解除する。

5.2 ITRON API

5.2.1 共有オブジェクトアクセスのための ITRON API

API 名	概　　要	種別
jti_get_obj	名前から共有オブジェクト識別番号を求める。	標準仕様
jti_get_<type>	指定された<type>属性の共有オブジェクトのデータを取得する。	標準仕様
jti_set_<type>	指定された<type>属性の共有オブジェクトにデータを設定する。	標準仕様
jti_loc_obj	指定された共有オブジェクトをロックする。	標準仕様
jti_unl_obj	指定された共有オブジェクトのロックを解除する。	標準仕様
jti_funl_obj	指定された共有オブジェクトのロックを強制解除する。	標準仕様

【補足説明】

<type>には Java の基本型(boolean, byte, char, short, int, long, float, double)をサポートする。

jti_get_obj 標準仕様

名前から共有オブジェクト識別番号を求める。

【C 言語 API】

```
ER ercd = jti_get_obj( const char *objnm, JNO *p_objno);
```

【パラメータ】

const char *objnm	共有オブジェクトの名前

【リターンパラメータ】

ER	ercd	正常終了(E_OK)またはエラーコード
JNO	objno	共有オブジェクトの識別番号

【エラーコード】

E_PAR	パラメータエラー(objnm が不正)
E_OBJ	オブジェクト状態エラー (objnm に対応する共有オブジェクトが存在していない)

【機能】

objnm に対応する Java の共有オブジェクトの識別番号を p_objno の示す領域に返す。

【補足説明】

本 API は objnm の文字列を UTF-8 の文字列とみなして、これと同一の名前をもった Java のオブジェクトの識別番号を返す。objnm を ASCII 文字列に限定してもよい。

jti_get_<type> 標準仕様

指定された<type>属性の共有オブジェクトのデータを取得する。

【C 言語 API】

```
ER ercd =jti_get_<type>(JNO objno, const char * clsnm,
          const char *fidnm, <c_type> *p_retval);
```

【型】

<type> 以下の<type>をサポートする。また、表に記述されていない<type>(文字列や配列など)を実装定義で追加してもよい。

<type>	Java の型
boolean	boolean
byte	byte
char	char
short	short
int	int
long	long
float	float
double	double

<c_type> <type>に対応する適切な C 言語の型。各<type>と C 言語の型との対応は実装定義とする。

【パラメータ】

JNO	objno	共有オブジェクトの識別番号
const char *	clsnm	クラス名
const char *	fldnm	フィールド名

【リターンパラメータ】

ER	ercd	正常終了(E_OK)またはエラーコード
<c_type>*	p_retval	<c_type>型の共有オブジェクトのデータ

【エラーコード】

E_PAR	パラメータエラー(objno が不正)
E_OBJ	オブジェクト状態エラー(objno に対応する共有オブジェクトが存在していない)

【機能】

objno で指定された共有オブジェクトにおいて、クラス名 clsnm とフィールド名 fldnm で指定されたフィールドで、<type>に対応する<c_type>型のデータを返す。なお、clsnm と fldnm で指定されたクラスフィールドの型が、<type>に一致していなければならない。クラス名 clsnm の記述方式は JNI の FindClass() API の引数と同じ記述方式とする。(例えば java.lang.String の場合、"java/lang/String"と指定する。)

【JTRON2.0 仕様との相違】

jti_get_mem 関数が Java の JNI 仕様に対応できないため、JNI 仕様に対応できるように変更された。

jti_set_<type> 標準仕様

指定された<type>属性の共有オブジェクトのデータを設定する。

【C 言語 API】

```
ER ercd =jti_set_<type>(JNO objno, const char * clsnm,
                        const char *fidnm, <c_type> val);
```

【型】

<type> 以下の<type>をサポートする。また、表に記述されていない<type>(文字列や配列など)を実装定義で追加してもよい。

<type>	Java の型
boolean	boolean
byte	byte
char	char
short	short
int	int
long	long
float	float
double	double

<c_type> <type>に対応する適切な C 言語の型。各<type>と C 言語の型との対応は実装定義とする。

【パラメータ】

JNO	objno	共有オブジェクトの識別番号
const char *	clsnm	クラス名
const char *	fldnm	フィールド名
<c_type>	val	<c_type>型の共有オブジェクトへ設定するデータ

【リターンパラメータ】

ER	ercd	正常終了(E_OK)またはエラーコード

【エラーコード】

E_PAR	パラメータエラー(objno が不正)
E_OBJ	オブジェクト状態エラー(objno に対応する共有オブジェクトが存在していない)

【機能】

objno で指定された共有オブジェクトのクラス名 clsnm とフィールド名 fldnm で指定された<type>に対応する<c_type>型のデータを設定する。なお、clsnm と fldnm で指定されたクラスフィールドの型が、<type>に一致していなければならない。クラス名 clsnm の記述方式は JNI の FindClass() API の引数と同じ記述方式とする。(例えば java.lang.String の場合、"java/lang/String"と指定する。)

【JTRON2.0 仕様との相違】

jti_get_mem 関数が Java の JNI 仕様に対応できないため、JNI 仕様に対応できるように変更された。

jti_loc_obj 標準仕様

指定された共有オブジェクトをロックする。

【C 言語 API】

```
ER ercd = jti_loc_obj(JNO objno, TMO tmout);
```

【パラメータ】

JNO	objno	共有オブジェクトの識別番号
TMO	tmout	タイムアウト時間(単位:ms)

【リターンパラメータ】

ER	ercd	正常終了(E_OK)またはエラーコード

【エラーコード】

E_PAR	パラメータエラー(objno が不正)
E_OBJ	オブジェクト状態エラー (objno に対応する共有オブジェクトが存在していない)
E_TMOUT	ポーリング失敗またはタイムアウト
E_RLWAI	待ち状態の強制解除
E_DLT	待ちオブジェクトの削除(共有が解除された)

【機能】

objno で指定された共有オブジェクトをロックする。既にロックされている場合として、以下の場合がある。
(1)Java 側クラス SharedObject の lock メソッドでロックされている。
(2)異なるリアルタイムタスク側の jti_loc_obj でロックされている。
既にロックされている場合は、待ち状態になる。この待ち状態は、Java 側クラス SharedObject の unlock メソッド、もしくはリアルタイムタスクの jti_unl_obj によりロックが解除される。ロックが解除された後本 API により指定されたオブジェクトをロックする。同一のリアルタイムタスクでロックされている場合は何もしないで正常終了する。

タイムアウト時間（ミリ秒）tmout に TMO_POL(=0)を指定した場合ポーリング、TMO_FEVR(=-1)を指定した場合無限待ちとなる。

jti_unl_obj 標準仕様

指定された共有オブジェクトのロックを解除する。

【C 言語 API】

```
ER ercd = jti_unl_obj(JNO objno);
```

【パラメータ】

JNO	objno	共有オブジェクトの識別番号

【リターンパラメータ】

ER	ercd	正常終了(E_OK)またはエラーコード

【エラーコード】

E_PAR	パラメータエラー(objno が不正)
E_OBJ	オブジェクト状態エラー(objno に対応する共有オブジェクトが存在していない、異なるタスクによるロックを解除しようとした。)

【機能】

objno で指定された同一のリアルタイムタスクによってロックされた共有オブジェクトのロックを解除する。objno で指定された共有オブジェクトが異なるリアルタイムタスク、もしくは Java スレッドによってロックされていた場合は、E_OBJ エラーを返す。

jti_funl_obj 標準仕様

指定された共有オブジェクトのロックを強制解除する。

【C 言語 API】

```
ER ercd = jti_funl_obj(JNO objno);
```

【パラメータ】

JNO	objno	共有オブジェクトの識別番号

【リターンパラメータ】

ER	ercd	正常終了(E_OK)またはエラーコード

【エラーコード】

E_PAR	パラメータエラー(objno が不正)
E_OBJ	オブジェクト状態エラー(objno に対応する共有オブジェクトが存在していない)

【機能】

objno で指定された共有オブジェクトを、ロックした Java スレッド、リアルタイムタスクに関わらず強制的にロックを解除する。リアルタイムタスク、Java スレッドがロックした共有オブジェクトを、リアルタイムタスクが強制ロック解除しようとする場合は、ロックが解除される。

【補足説明】

本 API はシステムが異常状態になった場合に強制的に状態を回復するための利用を想定している。

5.2.2 Java スレッド操作のための ITRON API

リアルタイムタスクから、Java スレッドを操作するための ITRON API を定義する。

API 名	概　　要	種別
jti_get_thr	名前からスレッド識別番号を求める。	拡張仕様
jti_isa_thr	Java の Thread クラス中の isAlive メソッドを呼ぶ。	拡張仕様
jti_int_thr	Java の Thread クラス中の interrupt メソッドを呼ぶ。	拡張仕様
jti_isi_thr	Java の Thread クラス中の isInterrupted メソッドを呼ぶ。	拡張仕様
jti_sus_thr	Java の Thread クラス中の suspend メソッドを呼ぶ。	拡張仕様
jti_rsm_thr	Java の Thread クラス中の resume メソッドを呼ぶ。	拡張仕様
jti_sta_thr	Java の Thread クラス中の start メソッドを呼ぶ。	拡張仕様
jti_stp_thr	Java の Thread クラス中の stop メソッドを呼ぶ。	拡張仕様
jti_get_jpr	Java の Thread クラス中の getPriority メソッドを呼ぶ。	拡張仕様
jti_set_jpr	Java の Thread クラス中の setPriority メソッドを呼ぶ。	拡張仕様
jti_des_thr	Java の Thread クラス中の destroy メソッドを呼ぶ。	拡張仕様

jti_get_thr 拡張仕様

名前からスレッド識別番号を求める。

【C言語API】

```
ER ercd = jti_get_thr( const char *thrnm, JNO *p_thrno);
```

【パラメータ】

const char *thrnm	Javaスレッドの名前

【リターンパラメータ】

ER	ercd	正常終了(E_OK)またはエラーコード
JNO	thrno	Javaスレッドの識別番号

【エラーコード】

E_OBJ	オブジェクト状態エラー(スレッドが存在していない)
E_PAR	パラメータエラー(thrnmが不正)

【機能】

Javaスレッドの名前thrnmに対応するJavaスレッドの識別番号thrnoを返す。

【補足説明】

Javaスレッドの名前の文字列をUTF-8文字列とみなして、これと同一の名前を持ったJavaスレッドの識別番号を返す。実装ではASCII文字列に限定してもよい。名前による検索順序は、Javaのスレッド名は同一の名前も付ける事ができるため、同一の名前の検索も含め実装依存となる。例としては、頂点のスレッドグループからThreadGroup#enumerate(ThreadGroup[])メソッドを利用してスレッドグループの一覧を取得する。取得した配列順のスレッドグループ一覧からThreadGroup#enumerate(Thread[])メソッドにて

スレッドの一覧を取得する。取得した配列順のスレッド一覧から一致する名前のスレッドを検索する手順が想定される。

jti_isa_thr 拡張仕様

Java の Thread クラス中の isAlive メソッドを呼ぶ。

【C 言語 API】

```
ER_BOOL ercd = jti_isa_thr(JNO thrno);
```

【パラメータ】

JNO	thrno	Java スレッドの識別番号

【リターンパラメータ】

ER_BOOL	ercd	エラーコードまたは真偽値(メソッドの戻り値)

【エラーコード】

E_PAR	パラメータエラー(thrno が不正)

【機能】

Java スレッドの識別番号 thrno に対応する Java スレッドに対して Thread クラスの isAlive メソッドを呼び出し、その結果を返す。

jti_int_thr 拡張仕様

Java の Thread クラス中の interrupt メソッドを呼ぶ。

【C 言語 API】

```
ER ercd = jti_int_thr(JNO thrno);
```

【パラメータ】

JNO	thrno	Java スレッドの識別番号

【リターンパラメータ】

ER	ercd	正常終了(E_OK)またはエラーコード

【エラーコード】

E_PAR	パラメータエラー(thrno が不正)

【機能】

Java スレッドの識別番号 thrno に対応する Java スレッドに対して Thread クラスの interrupt メソッドを呼び出す。

jti_isi_thr 拡張仕様

Java の Thread クラス中の isInterrupted メソッドを呼ぶ。

【C 言語 API】

```
ER_BOOL ercd = jti_isi_thr(JNO thrno);
```

【パラメータ】

JNO	thrno	Java スレッドの識別番号

【リターンパラメータ】

ER_BOOL	ercd	エラーコードまたは真偽値(メソッドの戻り値)

【エラーコード】

E_PAR	パラメータエラー(thrno が不正)

【機能】

Java スレッドの識別番号 thrno に対応する Java スレッドに対して Thread クラスの isInterrupted メソッドを呼び出し、その結果を返す。

jti_sus_thr 拡張仕様

Java の Thread クラス中の suspend メソッドを呼ぶ。

【C 言語 API】

```
ER ercd = jti_sus_thr(JNO thrno);
```

【パラメータ】

JNO	thrno	Java スレッドの識別番号

【リターンパラメータ】

ER	ercd	正常終了(E_OK)またはエラーコード

【エラーコード】

E_PAR	パラメータエラー(thrno が不正)
E_OBJ	オブジェクト状態エラー (SecurityException)

【機能】

Java スレッドの識別番号 thrno に対応する Java スレッドに対して Thread クラスの suspend メソッドを呼び出す。どういう条件でセキュリティ例外が発生したかは実装依存である。

jti_rsm_thr 拡張仕様

Java の Thread クラス中の resume メソッドを呼ぶ。

【C 言語 API】

```
ER ercd = jti_rsm_thr(JNO thrno);
```

【パラメータ】

JNO	thrno	Java スレッドの識別番号

【リターンパラメータ】

ER	ercd	正常終了(E_OK)またはエラーコード

【エラーコード】

E_PAR	パラメータエラー(thrno が不正)
E_OBJ	オブジェクト状態エラー (SecurityException)

【機能】

Java スレッドの識別番号 thrno に対応する Java スレッドに対して Thread クラスの resume メソッドを呼び出す。どういう条件でセキュリティ例外が発生したかは実装依存である。

jti_sta_thr 拡張仕様

Java の Thread クラス中の start メソッドを呼ぶ。

【C 言語 API】

```
ER ercd = jti_sta_thr(JNO thrno);
```

【パラメータ】

JNO	thrno	Java スレッドの識別番号

【リターンパラメータ】

ER	ercd	正常終了(E_OK)またはエラーコード

【エラーコード】

E_PAR	パラメータエラー(thrno が不正)
E_OBJ	オブジェクト状態エラー (IllegalThreadStateException)

【機能】

Java スレッドの識別番号 thrno に対応する Java スレッドに対して Thread クラスの start メソッドを呼び出す。スレッドがすでに開始された場合に、IllegalThreadStateException が発生され E_OBJ を返す。

jti_stp_thr 拡張仕様

Java の Thread クラス中の stop メソッドを呼ぶ。

【C 言語 API】

```
ER ercd = jti_stp_thr(JNO thrno);
```

【パラメータ】

JNO	thrno	Java スレッドの識別番号

【リターンパラメータ】

ER	ercd	正常終了(E_OK)またはエラーコード

【エラーコード】

E_PAR	パラメータエラー(thrno が不正)
E_OBJ	オブジェクト状態エラー (SecurityException, NullPointerException)

【機能】

Java スレッドの識別番号 thrno に対応する Java スレッドに対して Thread クラスの stop メソッドを呼び出す。どういう条件でセキュリティ例外が発生したかは実装依存である。

【補足説明】

オーバロードメソッド stop(Throwable)は使用頻度が低いと思われるためリアルタイムタスクから呼び出し可能なメソッドから除外する。

【JTRON2.0 仕様との相違】

命名規則に合わず jti_thr_stp から名称変更。

jti_get_jpr 拡張仕様

Java の Thread クラス中の getPriority メソッドを呼ぶ。

【C 言語 API】

```
ER ercd = jti_get_jpr(JNO thrno, INT *p_rslt);
```

【パラメータ】

JNO	thrno	Java スレッドの識別番号

【リターンパラメータ】

ER	ercd	正常終了(E_OK)またはエラーコード
INT	rslt	Java スレッドの優先度

【エラーコード】

E_PAR	パラメータエラー(thrno が不正)

【機能】

Java スレッドの識別番号 thrno に対応する Java スレッドに対して Thread クラスの getPriority メソッドを呼び出し、その結果を返す。

【注意】

本 API で取得できる優先度は Java プログラムの世界での優先度である。

jti_set_jpr 拡張仕様

Java の Thread クラス中の setPriority メソッドを呼ぶ。

【C 言語 API】

```
ER ercd = jti_set_jpr(JNO thrno, INT newpri);
```

【パラメータ】

JNO	thrno	Java スレッドの識別番号
INT	newpri	Java スレッドの優先度

【リターンパラメータ】

ER	ercd	正常終了(E_OK)またはエラーコード

【エラーコード】

E_PAR	パラメータエラー(thrno が不正, IllegalArgumentException)
E_OBJ	オブジェクト状態エラー (SecurityException)

【機能】

Java スレッドの識別番号 thrno に対応する Java スレッドに対して Thread クラスの setPriority メソッドを呼び出す。どういう条件でセキュリティ例外が発生したかは実装依存である。

【注意】

本 API で設定できる優先度は Java プログラムの世界での優先度である。

jti_des_thr 拡張仕様

Java の Thread クラス中の destroy メソッドを呼ぶ。

【C 言語 API】

```
ER ercd = jti_des_thr(JNO thrno);
```

【パラメータ】

JNO	thrno	Java スレッドの識別番号

【リターンパラメータ】

ER	ercd	正常終了(E_OK)またはエラーコード

【エラーコード】

E_PAR	パラメータエラー(thrno が不正)
E_OBJ	オブジェクト状態エラー (SecurityException)

【機能】

Java スレッドの識別番号 thrno に対応する Java スレッドに対して Thread クラスの destroy メソッドを呼び出す。どういう条件でセキュリティ例外が発生したかは実装依存である。

5.2.3 Java スレッドグループ操作のための ITRON API

リアルタイムタスクから、Java スレッドグループを操作するための ITRON API を定義する。

API 名	概　　要	種別
jti_get_tgr	名前からスレッドグループ識別番号を求める。	拡張仕様
jti_des_tgr	Java の ThreadGroup クラス中の destroy メソッドを呼ぶ。	拡張仕様
jti_sus_tgr	Java の ThreadGroup クラス中の suspend メソッドを呼ぶ。	拡張仕様
jti_rsm_tgr	Java の ThreadGroup クラス中の resume メソッドを呼ぶ。	拡張仕様
jti_stp_tgr	Java の ThreadGroup クラス中の stop メソッドを呼ぶ。	拡張仕様

jti_get_tgr 拡張仕様

名前からスレッドグループ識別番号を求める。

【C 言語 API】

```
ER ercd = jti_get_tgr( const char *tgrnm, JNO *p_tgrno);
```

【パラメータ】

`const char *tgrnm`	Java スレッドグループの名前

【リターンパラメータ】

ER	ercd	正常終了(E_OK)またはエラーコード
JNO	tgrno	Java スレッドグループの識別番号

【エラーコード】

E_OBJ	オブジェクト状態エラー(スレッドグループが存在していない)
E_PAR	パラメータエラー(tgrnm が不正)

【機能】

Java スレッドグループの名前 tgrnm に対応する Java スレッドグループの識別番号を返す。

【補足説明】

Java スレッドグループの名前の文字列を UTF-8 文字列とみなして、これと同一の名前を持った Java スレッドグループの識別番号を返す。実装では ASCII 文字列に限定してもよい。名前による検索順序は、Java のスレッドグループ名は同一の名前も付ける事ができるため、同一の名前の検索も含め実装依存となる。例としては、頂点のスレッドグループから ThreadGroup#enumerate(ThreadGroup[]) メソッドを利用してスレッドグループの一覧を取得する。取得した配列順のスレッドグループ一覧から一致する名前のスレッドグループを検索する手

順が想定される。

jti_des_tgr 拡張仕様

Java の ThreadGroup クラス中の destroy メソッドを呼ぶ。

【C 言語 API】

```
ER ercd = jti_des_tgr(JNO tgrno);
```

【パラメータ】

JNO	tgrno	Java スレッドグループの識別番号

【リターンパラメータ】

ER	ercd	正常終了(E_OK)またはエラーコード

【エラーコード】

E_PAR	パラメータエラー(tgrno が不正)
E_OBJ	オブジェクト状態エラー (SecurityException)

【機能】

Java スレッドグループの識別番号 tgrno に対応する Java スレッドグループに対して ThreadGroup クラスの destroy メソッドを呼び出す。どういう条件でセキュリティ例外が発生したかは実装依存である。

jti_sus_tgr 拡張仕様

Java の ThreadGroup クラス中の suspend メソッドを呼ぶ。

【C 言語 API】

```
ER ercd = jti_sus_tgr(JNO tgrno);
```

【パラメータ】

JNO	tgrno	Java スレッドグループの識別番号

【リターンパラメータ】

ER	ercd	正常終了(E_OK)またはエラーコード

【エラーコード】

E_PAR	パラメータエラー(tgrno が不正)
E_OBJ	オブジェクト状態エラー (SecurityException)

【機能】

Java スレッドグループの識別番号 tgrno に対応する Java スレッドグループに対して ThreadGroup クラスの suspend メソッドを呼び出す。どういう条件でセキュリティ例外が発生したかは実装依存である。

jti_rsm_tgr 拡張仕様

Java の ThreadGroup クラス中の resume メソッドを呼ぶ。

【C 言語 API】

```
ER ercd = jti_rsm_tgr(JNO tgrno);
```

【パラメータ】

JNO	tgrno	Java スレッドグループの識別番号

【リターンパラメータ】

ER	ercd	正常終了(E_OK)またはエラーコード

【エラーコード】

E_PAR	パラメータエラー(tgrno が不正)
E_OBJ	オブジェクト状態エラー (SecurityException)

【機能】

Java スレッドグループの識別番号 tgrno に対応する Java スレッドグループに対して ThreadGroup クラスの resume メソッドを呼び出す。どういう条件でセキュリティ例外が発生したかは実装依存である。

jti_stp_tgr 拡張仕様

Java の ThreadGroup クラス中の stop メソッドを呼ぶ。

【C 言語 API】

```
ER ercd = jti_stp_tgr(JNO tgrno);
```

【パラメータ】

JNO	tgrno	Java スレッドグループの識別番号

【リターンパラメータ】

ER	ercd	正常終了(E_OK)またはエラーコード

【エラーコード】

E_PAR	パラメータエラー(tgrno が不正)
E_OBJ	オブジェクト状態エラー (SecurityException)

【機能】

Java スレッドグループの識別番号 tgrno に対応する Java スレッドグループに対して ThreadGroup クラスの stop メソッドを呼び出す。どういう条件でセキュリティ例外が発生したかは実装依存である。

5.3 JavaAPI

5.3.1 パッケージ構成

共有オブジェクトを提供するクラスは、org.jtron.shared パッケージにまとめられている。以下の例外クラス、インタフェース、クラスから構成される。

例外クラス： SharedObjectException,
SharedObjectIllegalStateException,
SharedObjectTimeoutException

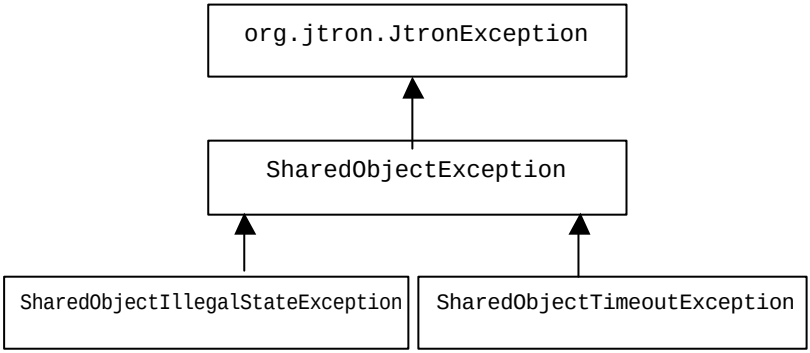


図 5.3: org.jtron.shared パッケージの例外クラス構成

インタフェース：Sharable
クラス： SharedObject

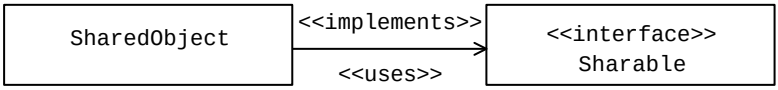


図 5.4: org.jtron.shared パッケージのクラス構成

5.3.2 共有オブジェクトインタフェース(Sharable)

public interface Sharable 標準仕様

共有オブジェクトインタフェースを規定する。

共有オブジェクトのインタフェースを規定する。共有オブジェクトとして使用するオブジェクトのクラスは、このインタフェースを実装しなければならない。

■クラス定義

```
package org.jtron.shared;

public interface Sharable {
        public abstract void lock();
        public abstract void lock(int timeout);
        public abstract void unlock();
        public abstract void forceUnlock();
        public abstract void unshare();
        public abstract void unshare(int timeout);
        public abstract Object getContent();
}
```

■メソッド

```
public abstract void lock()
        throws SharedObjectIllegalStateException;
```

【例外】

SharedObjectIllegalStateException	共有オブジェクトが不正な状態の時に発生。
OBJECT_UNSHARED	オブジェクトが共有されていない。

【機能】

オブジェクトをロックする。同一のスレッドにより既にロックされている場合は、何も起こらない。異なるスレッド、あるいはリアルタイムタスクにより既にロックされている場合は、ロックが解除されるまでブロックする。

```
public abstract void lock(int timeout)
        throws SharedObjectIllegalStateException,
               SharedObjectTimeoutException;
```

【パラメータ】

int	timeout	タイムアウト時間(単位:ms)

【例外】

SharedObjectIllegalStateException	共有オブジェクトが不正な状態の時に発生。
OBJECT_UNSHARED	オブジェクトが共有されていない。
SharedObjectTimeoutException	タイムアウトの例外が発生。

【機能】

オブジェクトをロックする。同一のスレッドにより既にロックされている場合は、何も起こらない。異なるスレッド、あるいはリアルタイムタスクにより既にロックされている場合は、タイムアウト時間の間、ロックが解除されるまでブロックする。タイムアウト時間を超えた場合、SharedObjectTimeoutException を発生する。

```
public abstract void unlock()
        throws SharedObjectIllegalStateException;
```

【例外】

SharedObjectIllegalStateException	共有オブジェクトが不正な状態の時に発生。
OBJECT_UNSHARED	オブジェクトが共有されていない。
OBJECT_LOCKED	オブジェクトが、既に他のスレッド、あるいはタスクによってロックされ

ている。

【機能】

同一のスレッドによりロックされたオブジェクトのロックを解除する。既にロックが解除されている場合はなにもしない。異なるスレッド、あるいはリアルタイムタスクによりロックされたオブジェクトに対しては SharedObjectIllegalStateException 例外が発生する。

```
public abstract void forceUnlock()
        throws SharedObjectIllegalStateException;
```

【例外】

SharedObjectIllegalStateException	共有オブジェクトが不正な状態の時に発生。
OBJECT_UNSHARED	オブジェクトが共有されていない。

【機能】

スレッドによりロックされたオブジェクトのロックを強制的に解除する。リアルタイムタスクによりロックされたものに関しては何もしない。
このメソッドは共有データをロックした Java スレッドが終了した場合に、そのロックを別の Java スレッドが強制解除するためのものである。

```
public abstract void unshare()
        throws SharedObjectIllegalStateException;
```

【例外】

SharedObjectIllegalStateException	共有オブジェクトが不正な状態の時に発生。
OBJECT_UNSHARED	オブジェクトが共有されていない。

【機能】

リアルタイム側との共有オブジェクトの共有を終了する。対象となる共有オブジェクトがスレッド、あるいはリアルタイムタスクによりロックしている場合はロックが解除されるまでブロックされ、ロックを

解除した後に共有を終了する。これは unshare メソッドの実行が非同期に行われるので、リアルタイムタスク、Java スレッドのアクセス終了後安全に共有を解除するためである。すでに共有されていない場合は、SharedObjectIllegalStateException 例外が発生する。

```
public abstract void unshare(int timeout)
        throws SharedObjectIllegalStateException,
               SharedObjectTimeoutException;
```

【パラメータ】

int	timeout	タイムアウト時間(単位:ms)

【例外】

SharedObjectIllegalStateException	共有オブジェクトが不正な状態の時に発生。
OBJECT_UNSHARED	オブジェクトが共有されていない。
SharedObjectTimeoutException	タイムアウトの例外が発生。

【機能】

リアルタイム側との共有オブジェクトの共有を終了する。対象となる共有オブジェクトがスレッド、あるいはリアルタイムタスクによりロックされている場合は、タイムアウト時間の範囲で、ロックが解除されるまで呼び出したスレッドはブロックされる。タイムアウト時間を超えた場合は、SharedObjectTimeoutException 例外が発生する。ロックを解除した後共有を終了する。これは unshare メソッドの実行が非同期に行われるので、リアルタイムタスク、Java スレッドのアクセス終了後安全に共有を解除するためである。既に共有されていない場合は、SharedObjectIllegalStateException 例外が発生する。

```
public abstract Object getContent();
```

【機能】

共有されているオブジェクトを返す。

5.3.3 共有オブジェクトクラス(SharedObject)

public class SharedObject implements Sharable標準仕様

共有オブジェクトクラス。

共有オブジェクトクラス。プログラマはこのクラスを継承するサブクラスを定義することによって、簡単に共有オブジェクトクラスを作成することができる。

■クラス定義

```
package org.jtron.shared;

public class SharedObject implements Sharable {
        protected Sharable shm;

        public SharedObject(String name);
        public SharedObject(Sharable shm, String name);
        public void lock();
        public void lock(int timeout);
        public void unlock();
        public void forceUnlock();
        public void unshare();
        public void unshare(int timeout);
        public Object getContent();
}
```

■コンストラクタ

```
public SharedObject(String name)
        throws  SharedObjectIllegalStateException;
```

【パラメータ】

String	name	共有オブジェクト名

【例外】

SharedObjectIllegalStateException	共有オブジェクトが不正な状態の時に発生。
ILLEGAL_NAME	不正な名前である。

【機能】

name という名前で共有オブジェクトを生成する。生成に失敗した時は、SharedObject IllegalStateException 例外が発生する。

```
public SharedObject(Sharable shm, String name)
        throws SharedObjectIllegalStateException;
```

【パラメータ】

Sharable	shm	共有オブジェクト
String	name	共有オブジェクト名

【例外】

SharedObjectIllegalStateException	共有オブジェクトが不正な状態の時に発生。
ILLEGAL_NAME	不正な名前である。

【機能】

Shareble を実装したクラスのオブジェクト shm を name という名前で共有オブジェクトを生成する。生成に失敗した時は、SharedObjectIllegalStateException 例外が発生する。

■メソッド

```
public void lock() throws SharedObjectIllegalStateException;
```

【例外】

SharedObjectIllegalStateException	共有オブジェクトが不正な状態の時に発生。
OBJECT_UNSHARED	オブジェクトが共有されていない。

【機能】

オブジェクトをロックする。同一のスレッドにより既にロックされている場合は、何も起こらない。異なるスレッド、あるいはリアルタイムタスクにより既にロックされている場合は、ロックが解除されるまでブロックする。

```
public void lock(int timeout)
        throws SharedObjectIllegalStateException,
               SharedObjectTimeoutException;
```

【パラメータ】

int	timeout	タイムアウト時間(単位:ms)

【例外】

SharedObjectIllegalStateException	共有オブジェクトが不正な状態の時に発生。
OBJECT_UNSHARED	オブジェクトが共有されていない。
SharedObjectTimeoutException	タイムアウトの例外が発生。

【機能】

オブジェクトをロックする。同一のスレッドにより既にロックされている場合は、何も起こらない。異なるスレッド、あるいはリアルタイムタスクにより既にロックされている場合は、タイムアウト時間の範囲で、ロックが解除されるまでブロックする。タイムアウト時間を超えた場合、SharedObject TimeoutException を発生する。

```
public void unlock( ) throws SharedObjectIllegalStateException;
```

【例外】

SharedObjectIllegalStateException	共有オブジェクトが不正な状態の時に発生。
OBJECT_UNSHARED	オブジェクトが共有されていない。
OBJECT_LOCKED	オブジェクトが、既に他のスレッド、あるいはタスクによってロックされている。

【機能】

同一のスレッドによりロックされたオブジェクトのロックを解除する。既にロックが解除されている場合はなにもしない。異なるスレッド、あるいはリアルタイムタスクによりロックされたオブジェクトに対しては SharedObjectIllegalStateException 例外が発生する。

```
public void forceUnlock()
        throws SharedObjectIllegalStateException;
```

【例外】

SharedObjectIllegalStateException	共有オブジェクトが不正な状態の時に発生。
OBJECT_UNSHARED	オブジェクトが共有されていない。

【機能】

スレッドによりロックされたオブジェクトのロックを強制的に解除する。リアルタイムタスクによりロックされたものに関しては何もしない。
このメソッドは共有データをロックした Java スレッドが終了した場合に、そのロックを別の Java スレッドが強制解除するためのものである。

```
public void unshare() throws SharedObjectIllegalStateException;
```

【例外】

SharedObjectIllegalStateException	共有オブジェクトが不

正な状態の時に発生。
OBJECT_UNSHARED オブジェクトが共有されていない。

【機能】

リアルタイム側との共有オブジェクトの共有を終了する。対象となる共有オブジェクトがスレッド、あるいはリアルタイムタスクによりロックしている場合はロックが解除されるまでブロックされ、ロックを解除した後に共有を終了する。これは unshare メソッドの実行が非同期に行われるので、リアルタイムタスク、Java スレッドのアクセス終了後安全に共有を解除するためである。すでに共有されていない場合は、SharedObjectIllegalStateException 例外が発生する。

```
public void unshare(int timeout)
        throws SharedObjectIllegalStateException,
               SharedObjectTimeoutException;
```

【パラメータ】

int timeout タイムアウト時間(単位:ms)

【例外】

SharedObjectIllegalStateException 共有オブジェクトが不正な状態の時に発生。
OBJECT_UNSHARED オブジェクトが共有されていない。
SharedObjectTimeoutException タイムアウトの例外が発生。

【機能】

リアルタイム側との共有オブジェクトの共有を終了する。対象となる共有オブジェクトがスレッド、あるいはリアルタイムタスクによりロックされている場合はタイムアウト時間の範囲で、ロックが解除されるまでブロックされる。タイムアウト時間を超えた場合は、SharedObjectTimeoutException 例外が発生する。ロックを解除した後共有を終了する。これは unshare メソッドの実行が非同期に行われるので、リアルタイムタスク、Java スレッドのアクセス終了後安全に共有を解除するためである。既に共有していない場合は、

SharedObjectIllegalStateException 例外が発生する。

public Object getContent()

【機能】

共有されているオブジェクトを返す。SharedObject では、このメソッドは、コンストラクタの引数で指定された Sharable なオブジェクトを返す(インスタンス変数 shm を返す実装となっている)。

■インスタンス変数

Sharable	shm	コンストラクタの引数で指定されたオブジェクト shm を保持する。指定が無い場合は、this が設定される。

5.3.4 共有オブジェクト関連の例外クラス (SharedObjectException)

```
org.jtron.JtronException
    |
    └── org.jtron.shared.SharedObjectException
```

public class SharedObjectException 標準仕様

共有オブジェクト関連の例外クラス。

共有オブジェクト関連の例外が発生したことを通知する。このクラスは、このパッケージ内の全例外クラスのスーパークラスである。

■クラス定義

```
package org.jtron.shared;

public class SharedObjectException extends java.lang.Exception
{
        public SharedObjectException();
        public SharedObjectException(String msg);
}
```

■コンストラクタ

public SharedObjectException();

【機能】

詳細なメッセージ無しで SharedObjectException を生成する。

public SharedObjectException(String msg);

【パラメータ】

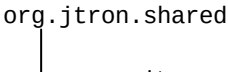
`String msg` 詳細メッセージ文字列

【機能】

指定された詳細メッセージ msg **をもつ** `SharedObjectException` **を生成する。**

5.3.5 メソッド発行状態に関する例外クラス

(SharedObjectIllegalStateException)



public class SharedObjectIllegalStateException 標準仕様

メソッドが実行できない不正な状態である時に発生する。

オブジェクトがあるメソッドを発行した時に、そのメソッドが実行できない不正な状態にある時に発生する。

■クラス定義

```
package org.jtron.shared;

public class SharedObjectIllegalStateException
                        extends SharedObjectException {
        public static final int ILLEGAL_MANAGER = 1;
        public static final int OBJECT_IN_USE = 2;
        public static final int OBJECT_NOEXIST = 3;
        public static final int ILLEGAL_NAME = 4;
        public static final int OBJECT_UNSHARED = 5;
        public static final int OBJECT_LOCKED = 6;

        public SharedObjectIllegalStateException(int cause);
        public SharedObjectIllegalStateException(
                                int cause, String msg);
        public int getCause();
}
```

■定数

ILLEGAL_MANAGER	1	共有オブジェクトマネージャ

		が不正である、または、存在しない。
OBJECT_IN_USE	2	既にオブジェクトが登録されている。
OBJECT_NOEXIST	3	そのオブジェクトは存在しない、または、既に削除されている。
ILLEGAL_NAME	4	不正な名前である。
OBJECT_UNSHARED	5	そのオブジェクトは共有されていない。
OBJECT_LOCKED	6	そのオブジェクトが他のスレッド、もしくはタスクによってロックされている。

■コンストラクタ

```
public SharedObjectIllegalStateException(int cause);
```

【パラメータ】

int	cause	例外の詳細原因

【機能】

指定された原因の例外オブジェクトを生成する。パラメータ cause には例外の詳細原因を渡す。

```
publicSharedObjectIllegalStateException(intcause,Stringmsg)
```

【パラメータ】

int	cause	例外の詳細原因
String	msg	メッセージ文字列

【機能】

指定された原因で、詳細なメッセージを持つ例外オブジェクトを生成する。cause には例外の詳細原因が、msg にはオブジェクト名が指定

される。

■メソッド

```
public int getCause()
```

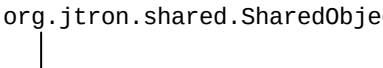
【戻り値】

int	例外の詳細原因

【機能】

例外の詳細原因を返す。

5.3.6 タイムアウトに関する例外クラス (SharedObjectTimeoutException)



public class SharedObjectTimeoutException 標準仕様

タイムアウトに関する例外クラス。

タイムアウト時間がすぎたことを通知する。

■クラス定義

```
package org.jtron.shared;

public class SharedObjectTimeoutException
                        extends SharedObjectException {
        public SharedObjectTimeoutException();
        public SharedObjectTimeoutException(String msg);
}
```

■コンストラクタ

public SharedObjectTimeoutException();

【機能】

詳細なメッセージ無しで SharedObjectTimeoutException を生成する。

public SharedObjectTimeoutException(String msg);

【パラメータ】

String msg 詳細メッセージ文字列

【機能】

指定された詳細メッセージ msg をもつ SharedObjectTimeoutException を生成する。

5.4 参考情報

共有オブジェクトの Java 側のクラス構成例を示す。ここでは共有オブジェクトを一括管理するマネージャ(共有オブジェクトマネージャ)を定義する。共有オブジェクトは生成されると、共有オブジェクトマネージャに登録される。各共有オブジェクトで実行された `lock()`メソッドなどは共有オブジェクトマネージャの `lock()`メソッドを呼び出すように実装することで、詳細実装を共有オブジェクトマネージャですべて行うことが可能となる。

5.4.1 パッケージ構成図

以下の例外クラス、インタフェース、クラスからなる。

例外クラス： `SharedObjectException`,
`SharedObjectIllegalStateException`,
`SharedObjectTimeoutException`,
`SharedObjectIllegalManagerException`

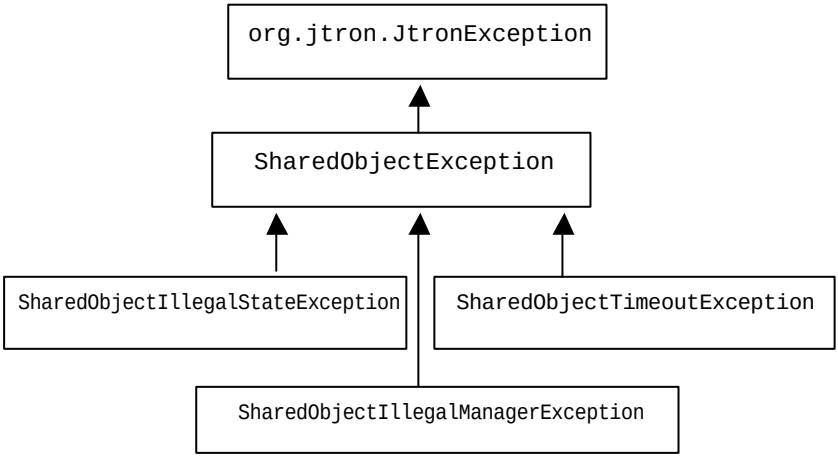


図 5.5: 例外クラス構成

インタフェース：Sharable
クラス： SharedObject, SharedObjectManager

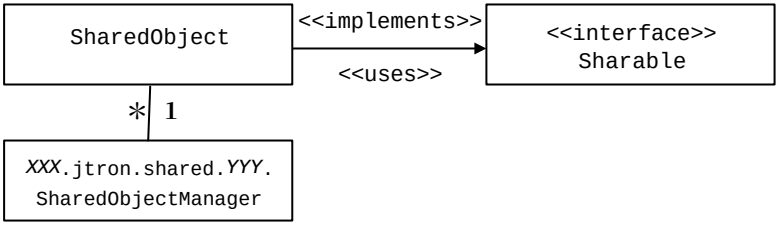


図 5.6: **クラス構成**

5.4.2 共有オブジェクト管理クラス (SharedObjectManager)

```
java.lang.Object
   |
   +--XXX.jtron.shared.YYY.SharedObjectManager
```

public class SharedObjectManager

共有オブジェクト管理クラス。

共有オブジェクトの管理クラス。リアルタイムタスク側とのインタフェースを受け持つ。ベンダ実装のパッケージに作成する。

■クラス定義

```
package XXX.jtron.shared.YYY;

public class SharedObjectManager {
        public static SharedObjectManager
                        getSharedObjectManager();

        public void share(Sharable obj, String name);
        public void unshare(String name);
        public void unshare(String name, int timeout);
        public void lock(Sharable obj);
        public void lock(Sharable obj, int timeout);
        public void unlock(Sharable obj);
        public void forceUnlock(Sharable obj);
}
```

■メソッド

```
public static SharedObjectManager getSharedObjectManager()
                throws SharedObjectIllegalManagerException;
```

【戻り値】

SharedObjectManager	共有オブジェクトマネージャの実態クラス

【例外】

SharedObjectIllegalManagerException	共有オブジェクトマネージャが不正な状態の時に発生。

【機能】

共有オブジェクトマネージャオブジェクトを返す。マネージャが用意できない場合や不正な場合は、SharedObjectIllegalManagerException 例外が発生する。

```
public void share(Sharable obj, String name)
        throws SharedObjectIllegalStateException;
```

【パラメータ】

Sharable	obj	共有オブジェクト
String	name	共有オブジェクト名

【例外】

SharedObjectIllegalStateException		共有オブジェクトが不正な状態の時に発生。
	OBJECT_IN_USE	既にオブジェクトが登録されている。
	ILLEGAL_NAME	不正な名前である。

【機能】

共有オブジェクト obj を共有オブジェクト名 name という名前で登録する。既に登録されている場合や、不正な名前の場合は、SharedObjectIllegalStateException 例外が発生する。

```
public void unshare(String name)
        throws SharedObjectIllegalStateException;
```

【パラメータ】

	String name	共有オブジェクト名

【例外】

SharedObjectIllegalStateException	共有オブジェクトが不正な状態の時に発生。
OBJECT_NOEXIST	オブジェクトが存在しない(すでに削除されている)。
OBJECT_UNSHARED	オブジェクトが共有されていない。

【機能】

共有オブジェクト名 name に対応するオブジェクトを削除する。すでに削除されている場合は、SharedObjectIllegalStateException 例外が発生する。

```
public void unshare(String name, int timeout)
        throws SharedObjectIllegalStateException,
                        SharedObjectTimeoutException;
```

【パラメータ】

	String name	共有オブジェクト名
	int timeout	タイムアウト時間(単位:ms)

【例外】

SharedObjectIllegalStateException	共有オブジェクトが不正な状態の時に発生。
OBJECT_NOEXIST	オブジェクトが存在しない(すでに削除されている)。
OBJECT_UNSHARED	オブジェクトが共有されていない。
SharedObjectTimeoutException	タイムアウトの例外が発生。

【機能】

共有オブジェクト名 name に対応するオブジェクトを削除する。オブジェクトがロックされている場合はロックが解除されるまでブロックし、ロックを解除した後に削除する。タイムアウト時間を超えた場合は、SharedObjectTimeoutException 例外が発生する。すでに削除されている場合は、SharedObjectIllegalStateException 例外が発生する。

```
public void lock(Sharable obj)
        throws SharedObjectIllegalStateException;
```

【パラメータ】

Sharable	obj	共有オブジェクト

【例外】

SharedObjectIllegalStateException		共有オブジェクトが不正な状態の時に発生。
	OBJECT_NOEXIST	オブジェクトが存在しない(すでに削除されている)。
	OBJECT_UNSHARED	オブジェクトが共有されていない。

【機能】

オブジェクトをロックする。同一のスレッドにより既にロックされている場合は、何も起こらない。異なるスレッド、あるいはリアルタイムタスクにより既にロックされている場合は、ロックが解除されるまでブロックする。

```
public void lock(Sharable obj, int timeout)
        throws SharedObjectIllegalStateException,
                        SharedObjectTimeoutException;
```

【パラメータ】

Sharable	obj	共有オブジェクト
int	timeout	タイムアウト時間(単位:ms)

【例外】

SharedObjectIllegalStateException	共有オブジェクトが不正な状態の時に発生。
OBJECT_NOEXIST	オブジェクトが存在しない(すでに削除されている)。
OBJECT_UNSHARED	オブジェクトが共有されていない。
SharedObjectTimeoutException	タイムアウトの例外が発生。

【機能】

オブジェクトをロックする。同一スレッドにより既にロックされている場合は何も起こらない。リアルタイムタスク、もしくは異なるスレッドにより既にロックされている場合は、ロックが解除されるまでブロックする。タイムアウト時間(単位:ms)が指定でき、タイムアウト時間の範囲でロックが解除されるまでブロックする。タイムアウト時間が過ぎると、SharedObjectTimeoutException 例外が発生する。

```
public void unlock(Sharable obj)
        throws SharedObjectIllegalStateException;
```

【パラメータ】

Sharable	obj	共有オブジェクト

【例外】

SharedObjectIllegalStateException	共有オブジェクトが不正な状態の時に発生。
OBJECT_NOEXIST	オブジェクトが存在しない(すでに削除されている)。
OBJECT_UNSHARED	オブジェクトが共有されていない。
OBJECT_LOCKED	オブジェクトが既に他のスレッド、あるいはタスクによってロックされている。

【機能】

同一のスレッドによりロックされたオブジェクトのロックを解除する。既にロックが解除されている場合はなにもしない。異なるスレッド、

あるいはリアルタイムタスクによりロックされたオブジェクトに対しては SharedObjectIllegalStateException 例外が発生する。

```
public void forceUnlock(Sharable obj)
        throws SharedObjectIllegalStateException;
```

【例外】

SharedObjectIllegalStateException		共有オブジェクトが不正な状態の時に発生。
	OBJECT_UNSHARED	オブジェクトが共有されていない。

【パラメータ】

Sharable	obj	共有オブジェクト

【機能】

ロックしたスレッドに関わらず、ロックされたオブジェクトのロックを解除する。既にロックが解除されている場合は何もしない。

【補足説明】

用途は、ロックしたオブジェクトをスレッドがロックを解除しないで終了した場合に使用する。

5.4.3 共有オブジェクトマネージャに関する例外クラス (SharedObjectTimeoutException)

```
org.jtron.shared.SharedObjectException
    |
    +-- XXX.jtron.shared.YYY.SharedObjectIllegalManagerException
```

public class SharedObjectIllegalManagerException

共有オブジェクトマネージャに関する例外クラス。

共有オブジェクトマネージャの作成に失敗した時に発生する。

■クラス定義

```
package XXX.jtron.shared.YYY;

public class SharedObjectIllegalManagerException
                        extends SharedObjectException {
        public SharedObjectIllegalManagerException();
        public SharedObjectIllegalManagerException(
                                          String msg);
}
```

■コンストラクタ

public SharedObjectIlleaglManagerException();

【機能】

詳細なメッセージなしで SharedObjectIllegalManagerException を生成する。

public SharedObjectIllegalManagerException(String msg);

【パラメータ】

String	msg	詳細メッセージ文字列

【機能】

指定された詳細メッセージ msg をもつ SharedObjectIllegalManagerException を生成する。

第 6 章 タイプ 3 インタフェース (ストリーム通信)

6.1 概要

6.1.1 ストリームインタフェースの位置付け

ストリームインタフェースは、Java の標準的な入出力インタフェースである InputStream、OutputStream クラスを利用して、リアルタイムタスクと Java プログラムの間の通信手段を提供する。
Java 側では、リアルタイムタスクと通信するストリームが、InputStream、OutputStream の抽象クラスの実装として提供される。これは、Socket クラスから InputStream、OutputStream の抽象クラスを実装するのに対応している。
ITRON 側では、Java プログラムとのストリーム通信を行うための機構を OS の機能を利用して提供する。

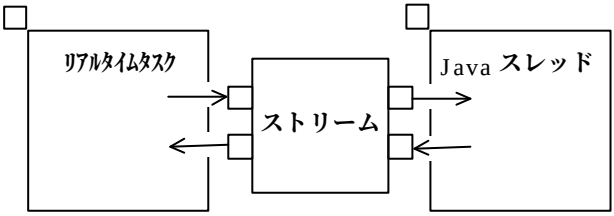


図 6.1:ストリーム

6.1.2 ストリームとチャネルの状態

ストリーム通信用のバッファなどのリソース管理は ITRON 側で行う。したがって、ストリームの生成・削除は、リアルタイムタスクから行う (jti_cre_stm/jti_del_stm)。
ストリームは、リアルタイムタスクから Java プログラムへデータを送るためのチャネルと、Java プログラムからリアルタイムタスクへデータを送るためのチャネルの、2 つのチャネルからなる。また、生成時の指定

により、片方のチャネルのみを持つストリームを生成することもできる。生成直後のストリームは未接続状態である。未接続状態のストリームに対して、Java プログラムからストリームがオープンされると、両方のチャネルが接続状態となる(片方のチャネルのみを持つストリームの場合は、片方のチャネルのみが接続状態になる。もう片方のチャネルは、常に切断状態になっていると考える)。

チャネルの送信側がチャネルを正常にクローズすると(Java プログラムからは OutputStream を close メソッド、リアルタイムタスクからは jti_sht_stm API を呼び出す)、チャネルは送信終了状態になる。チャネルの受信側がバッファに入ったデータを取り出し、正常クローズされたことを検出し(Java プログラムは InputStream の read メソッドが -1 を返すことで、リアルタイムタスクは jti_rea_stm が 0 を返すことで正常クローズされたことを検出)、それを確認した時点で(Java プログラムは、InputStream を close することで確認。リアルタイムタスクは、jti_rea_stm が 0 を返した時点で確認したものとみなす)、チャネルは切断状態となる。両方のチャネルが切断状態になった時、ストリームが未接続状態に戻る。

Java プログラム側で受信に使用するチャネルを InputStream、送信に使用するチャネルを OutputStream とする。Java プログラムが受信側となるチャネルを強制クローズすると(InputStream を close)、チャネルは強制切断状態になる。チャネルの送信側であるリアルタイムタスクがそれを検出・確認した時点で(リアルタイムタスクは、jti_wri_stm ないしは jti_sht_stm が E_CLS を返すことで、チャネルの強制クローズを検出し、それを確認したものとみなす)、チャネルは切断状態となる。それに対して、リアルタイムタスクは、リアルタイムタスクが受信側となるチャネルを強制クローズすることができない(強制クローズするための API が用意されていない)。

JtronStream の close は、OutputStream と InputStream の両方の close と等価である。

ストリームは識別子番号で識別する。

【補足説明】

jti_rea_stm が 0 を返す(または jti_wri_stm か jti_sht_stm が E_CLS を返す)と、リアルタイムタスクがチャネルの正常クローズ(または強制クローズ)を確認したとみなし、チャネルの状態は変化する。そのため、再度 jti_rea_stm(または jti_wri_stm か jti_sht_stm)を呼び出しても、0(または E_CLS)は返らないので注意が必要である。

●期待する操作手順

図 6.2 の順序で操作することを期待している。

リアルタイムタスク	Java プログラム
1.jti_cre_stm でストリームを生成	
	2.JtronStream でストリームを接続状態にする。
3.jti_rea_stm,jti_wri_stm を使ってストリームにデータを読み書き。	3.Stream の read(),write()を使ってストリームにデータを読み書き
リアルタイムタスクから送信終了の場合 4.jti_sht_stm でストリームを送信終了状態に遷移	5.InputStream.close() を実行してストリームを切断状態に遷移。
5.jti_rea_stm が 0 を返して、ストリームを切断状態に遷移。	Java プログラムから送信終了の場合 4.OutputStream.close() でストリームを送信終了状態に遷移。
5.jti_wri_stm または jti_sht_stm が E_CLS を返すと切断状態に遷移	Java プログラムから強制切断する場合 4.InputStream.close() でストリームを強制切断状態に遷移。

図 6.2:期待する操作順序

【仕様決定の理由】

リアルタイムタスクから Java プログラムへのチャネルでは、リアルタイムタスクが受信側となるチャネルを強制クローズする機能は用意していない。これはリアルタイムタスクは常時動作してデータの取得やコマンド送出を行っているため、リアルタイムタスクからチャネルを強制クローズすることはないと考えたからである。

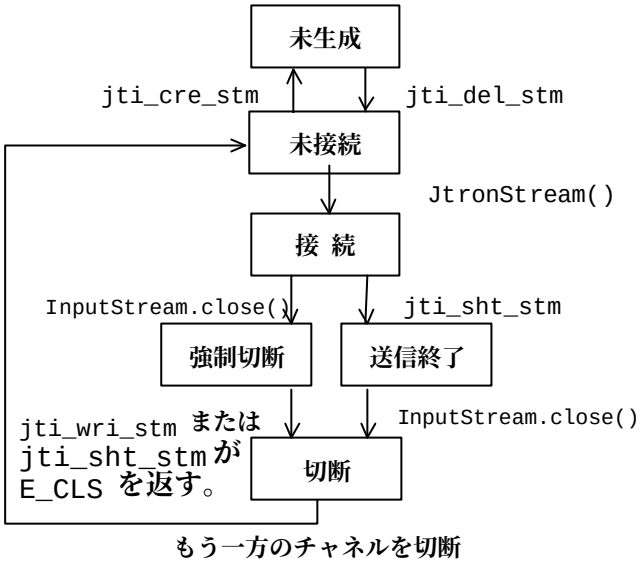


図 6.3:リアルタイムタスクから Java プログラムへのチャネルの状態遷移

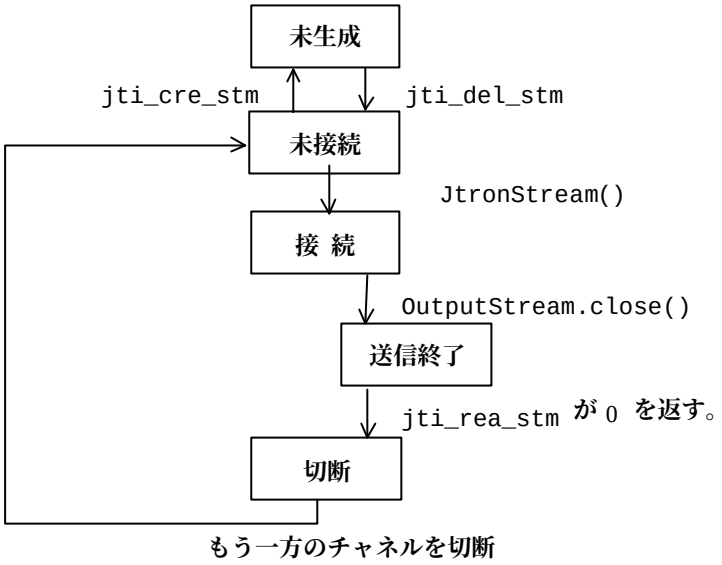


図 6.4: Java プログラムからリアルタイムタスクへのチャネルの状態遷移

6.2 ITRON API

ストリームを操作するための ITRON API
リアルタイムタスクから、ストリームを操作するための ITRON API **を定義する。**

API 名	概　　要	種別
jti_cre_stm	ストリームを生成する。	標準仕様
jti_del_stm	ストリームを削除する。	標準仕様
jti_wri_stm	ストリームのデータを送信する。	標準仕様
jti_rea_stm	ストリームのデータを受信する。	標準仕様
jti_sht_stm	ストリームのデータ送信を終了する。	標準仕様
jti_ref_stm	ストリームの状態参照を行う。	標準仕様

JTI_CRE_STM 標準仕様
jti_cre_stm 標準仕様

ストリームを生成する。

【C 言語 API】

```
ER ercd = jti_cre_stm(ID stmid, T_JTI_CSTM *pk_cstm);
```

【静的 API】

```
JTI_CRE_STM(ID stmid,
           { VP exinf, ATR stmpatr, VP wbuf,
            INT wbufsz, VP rbuf, INT rbufsz });
```

【パラメータ】

ID	tmid	ストリーム識別子
T_JTI_CSTM	*pk_cstm	ストリーム生成情報

【リターンパラメータ】

ER	ercd	正常終了(E_OK)またはエラーコード
pk_cstm の内容(T_JIT_CSTM 型)		
VP	exinf	拡張情報
ATR	stmatr	ストリーム属性
VP	wbuf	送信バッファの先頭
INT	wbufsz	送信バッファのサイズ
VP	rbuf	受信バッファの先頭
INT	rbufsz	受信バッファのサイズ

(他に実装定義のパラメータがあってもよい)

【エラーコード】

E_ID	不正 ID 番号
E_RSATR	予約属性
E_PAR	パラメータエラー(pk_cstm のアド

	レス、wbuf, wbufsz, rbuf, rbufsz が不正、ストリーム属性が不正)
E_OBJ	オブジェクト状態エラー(指定した識別子のストリームが生成済み)

【機能】

指定した識別子のストリームを生成する。ストリーム属性を用いて、ストリームを送信専用ないしは受信専用にすることができる。具体的には、ストリーム属性に(TA_WRITE|TA_READ)を指定すると両方向の通信が可能となり、TA_WRITE を指定すると送信専用、TA_READ を指定すると受信専用となる。なお、TA_WRITE と TA_READ いずれも指定しない場合には、E_PAR エラーとなる。
ストリームが送信専用の場合、rbuf と rbufsz は無視される。逆に受信専用の場合には、wbuf と wbufsz は無視される。送信バッファサイスが負の場合は、E_PAR エラーとなる(バッファサイズが 0 の場合には、同期通信となる)。

【補足説明】

バッファを内部で確保する実装では、バッファサイズの先頭アドレスとして NULL(0)を指定させるものとする。その場合にも、バッファサイズの指定は有効である。また、NULL(0)が指定された場合にはバッファを内部で確保し、それ以外の場合は与えられたバッファを用いる実装も可能である。

【JTRON2.0 仕様との相違】

μITRON4.0 仕様の変更に合わせて、NADR(-1)の指定を NULL(0)指定に変更する。

jti_del_stm 標準仕様

ストリームを削除する。

【C 言語 API】

```
ER ercd = jti_del_stm(ID stmid);
```

【パラメータ】

ID	stmid	ストリーム識別子

【リターンパラメータ】

ER	ercd	正常終了(E_OK)またはエラーコード

【エラーコード】

E_ID	不正 ID 番号
E_NOEXS	オブジェクト未生成
E_OBJ	オブジェクト状態エラー(指定したストリームが未接続状態でない)

【機能】

指定したストリームを削除する。未接続状態以外のストリームを削除しようとした場合、E_OBJ エラーとなる。jti_rea_stm, jti_wri_stm を発行して待ち状態になっているタスクは起こされて、E_DLT を返す。

jti_wri_stm 標準仕様

ストリームのデータを送信する。

【C 言語 API】

```
ER ercd = jti_wri_stm(ID stmid, const VP data,
                          INT len, TMO timeout);
```

【パラメータ】

ID	stmid	ストリーム識別子
const VP	data	送信データの先頭アドレス
INT	len	送信したいデータの長さ
TMO	timeout	タイムアウト指定(単位:ms)

【リターンパラメータ】

ER	ercd	正常終了(E_OK)またはエラーコード

【エラーコード】

E_ID	不正 ID 番号
E_NOEXS	オブジェクト未生成
E_PAR	パラメータエラー(data, len, timeout が不正)
E_OBJ	オブジェクト状態エラー(指定したストリームが受信専用である、jti_wri_stm がペンディング中、未接続状態で待っているストリームが削除された。)
E_TMOUT	ポーリング失敗またはタイムアウト
E_RLWAI	待ち状態の強制解除
E_CLS	待ちオブジェクトの状態変化(送信用のチャネルが強制切断された)

【機能】

指定したストリームにデータを送信する。データが送信バッファに入った時点でこの API からリターンする。空いている送信バッファ長が送信しようとしたデータよりも短い場合、送信バッファに空きがない場合には、空きが生じるまで待ち状態となる。
ストリームへのデータ送信が可能なのは、送信用のチャネルが接続状態の時のみである。チャネルが強制切断状態時は、jti_wri_stm は E_CLS エラーを返し、チャネルは切断状態に遷移する。その他の状態(切断状態、未接続状態)の場合には、チャネルが接続状態になるまで、jti_wri_stm を呼び出したタスクは待ち状態となる。
同一のストリームに対する jti_wri_stm がペンディングしている間に jti_wri_stm を発行すると E_OBJ エラーとなる。
タイムアウト時間（ミリ秒）timeout に TMO_POL(=0)を指定した場合ポーリング、TMO_FEVR(=-1)を指定した場合無限待ちとなる。

jti_rea_stm 標準仕様

ストリームのデータを受信する。

【C 言語 API】

```
ER ercd = jti_rea_stm(ID stmid, VP data,
                          INT len, TMO timeout);
```

【パラメータ】

ID	stmid	ストリーム識別子
VP	data	受信データを入れる領域の先頭アドレス
INT	len	受信したいデータの長さ
TMO	timeout	タイムアウト指定(単位:ms)

【リターンパラメータ】

ER	ercd	取り出したデータの長さまたはエラーコード

【エラーコード】

正の値	取り出したデータの長さ
0	データ終結(接続状態が正常切断された)
E_ID	不正 ID 番号
E_NOEXS	オブジェクト未生成
E_PAR	パラメータエラー(data, len, timeout が不正)
E_OBJ	オブジェクト状態エラー(指定したストリームが送信専用である、jti_rea_stm がペンディング中、未接続状態で待っているストリームが削除された。)
E_TMOUT	ポーリング失敗またはタイムアウト
E_RLWAI	待ち状態の強制解除

【機能】

指定したストリームからデータを受信する。受信バッファに入ったデータを取り出した時点でこの API からリターンする。受信バッファに入っているデータ長が受信しようとしたデータ長よりも短い場合、受信バッファが空になるまでデータを取り出し、取り出したデータの長さを返す。受信バッファが空の場合には、データを受信するまで待ち状態となる。Java プログラムが受信用のチャネルを正常クローズし、受信バッファにデータがなくなった時点で本 API を実行すると、0 が返る。
ストリームへのデータ受信が可能なのは、受信用のチャネルが接続状態の時のみである。チャネルがその他の状態(切断状態、接続状態)の場合には、チャネルが接続状態になるまで、jti_rea_stm を呼び出したタスクは待ち状態となる。
同一のストリームに対する jti_rea_stm がペンディングしている間に jti_rea_stm を発行すると E_OBJ エラーとなる。
タイムアウト時間（ミリ秒）timeout に TMO_POL(=0)を指定した場合ポーリング、TMO_FEVR(=-1)を指定した場合無限待ちとなる。

jti_sht_stm 標準仕様

ストリームのデータ送信を終了する。

【C 言語 API】

```
ER ercd = jti_sht_stm(ID stmid);
```

【パラメータ】

ID	stmid	ストリーム識別子

【リターンパラメータ】

ER	ercd	正常終了(E_OK)またはエラーコード

【エラーコード】

E_ID	不正 ID 番号
E_NOEXS	オブジェクト未生成
E_OBJ	オブジェクト状態エラー(指定したストリームが受信専用である、送信用のチャネルが切断ないしは未接続状態、jti_wri_stm がペンディング中)
E_CLS	待ちオブジェクトの状態変化(送信用のチャネルが強制切断された)

【機能】

指定したストリームに対するデータ送信を終了し、送信用のチャネルを送信終了状態にする。ストリームへのデータ送信の終了が可能なのは、送信用のチャネルが接続状態の時のみである。チャネルが強制切断状態の時は、jti_sht_stm は E_CLS エラーを返し、チャネルは切断状態に遷移する。その他の状態(切断状態、未接続状態)の場合には、E_OBJ エラーとなる。
同一のストリームに対する jti_wri_stm がペンディングしている間に jti_sht_stm を発行すると E_OBJ エラーとなる。

jti_ref_stm 標準仕様

ストリームの状態参照を行う。

【C 言語 API】

```
ER ercd = jti_ref_stm(ID stmid, T_JTI_RSTM *pk_rstm);
```

【パラメータ】

ID	stmid	ストリーム識別子
T_JTI_RSTM	*pk_rstm	ストリーム状態を返すパケットのアドレス

【リターンパラメータ】

ER	ercd	正常終了(E_OK)またはエラーコード
pk_rstm の内容(T_JTI_RSTM 型)		
VP	exinf	拡張情報
INT	wrisz	待たずに送信可能なデータ長(バイト数)
INT	reasz	待たずに受信可能なデータ長(バイト数)

(他に実装定義のパラメータがあってもよい)

【エラーコード】

E_ID	不正 ID 番号
E_NOEXS	オブジェクト未生成
E_PAR	パラメータエラー(pk_rstm のアドレスが不正)

【機能】

指定したストリームの状態を参照し、pk_rstm に返す。
exinf には、jti_cre_stm で指定した拡張情報がそのまま返る。
wrisz には、待たすに送信可能なデータ長(バイト数)が返る。指定したストリームが受信専用の場合には-1 となる。reasz には、待たずに

受信可能なデータ長(バイト数)が返る。指定したストリームが送信専用の場合には-1 となる。

6.3 Java API

6.3.1 パッケージ構成

ストリームインタフェースのクラスは、org.jtron.stream パッケージにまとめられている。以下のクラスから構成される。

例外クラス： JtronStreamException, JtronStreamIllegalStateException, JtronStreamTimeoutException

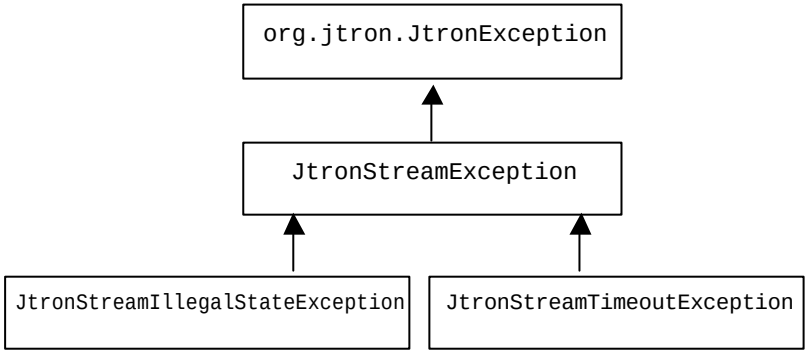


図 6.5: org.jtron.stream パッケージの例外クラス構成

クラス：JtronStream

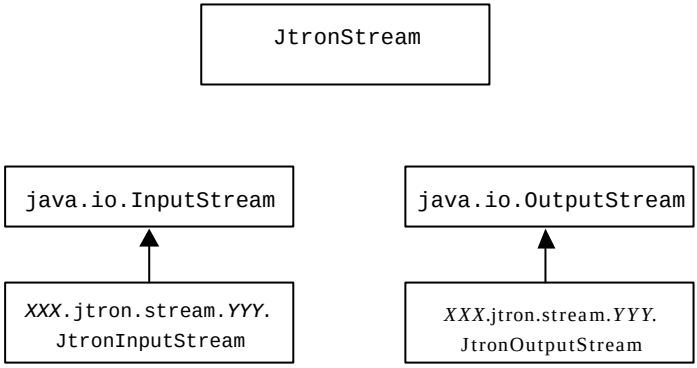


図 6.6: org.jtron.stream パッケージのクラス構成

6.3.2 ストリームクラス(JtronStream)

```
java.lang.Object
   |
   +--org.jtron.stream.JtronStream
```

public class JtronStream 標準仕様

ストリームを使ってリアルタイムタスクと通信を行うためのクラス。

■クラス定義

```
package org.jtron.stream;

public class JtronStream extends java.lang.Object {
        public static final int MAIN_STREAM = 1;

        public JtronStream(int stmid);
        public JtronStream(int stmid, int timeout);
        public synchronized InputStream getInputStream();
        public synchronized OutputStream getOutputStream();
        public synchronized void setTimeout(int timeout);
        public synchronized int getTimeout();
        public synchronized void close();
}
```

■定数

MAIN_STREAM	1	標準的なストリーム識別子

■コンストラクタ

```
public JtronStream(int stmid)
        throws JtronStreamIllegalStateException;
```

【パラメータ】

int	stmid	ストリーム識別子

【例外】

JtronStreamIllegalStateException	メソッド発行状態に関する例外クラス
STREAM_IN_USE	指定した識別子のストリームはすでに使用されている。
ILLEGAL_ARGUMET	指定した引数が不正。

【機能】

指定した識別子のストリームをオープンする。両方のチャネルが接続状態になる(片方のチャネルのみを持つストリームの場合は、片方のチャネルのみ)。指定した識別子が既に使用されている場合は、JtronStreamException が発生する。ストリームが生成されていない場合は永久に待つ。

```
public JtronStream(int stmid, int timeout)
        throws JtronStreamIllegalStateException,
                JtronStreamTimeoutException;
```

【パラメータ】

int	stmid	ストリーム識別子
int	timeout	タイムアウト時間(単位:ms)

【例外】

JtronStreamIllegalStateException	メソッド発行状態に関する例外クラス
STREAM_IN_USE	指定した識別子のストリームはすでに使用されている。
ILLEGAL_ARGUMET	指定した引数が不正。
JtronStreamTimeoutException	タイムアウトの例外が発生。

【機能】

タイムアウト時間（ミリ秒）を指定し、指定した識別子のストリーム

をオープンする。ストリームが生成されていない場合は指定された時間待つ。指定した識別子が既に使用されている場合は、JtronStreamIllegalStateException例外が発生し、タイムアウト発生時はJtronStreamTimeoutException例外が発生する。

■メソッド

```
public synchronized InputStream getInputStream()
        throws JtronStreamIllegalStateException;
```

【戻り値】

InputStream	受信ストリームクラス

【例外】

JtronStreamIllegalStateException	メソッド発行状態に関する例外クラス
STREAM_CLOSED	指定した識別子のストリームはすでにクローズしている。

【機能】

受信ストリームを取得する。取得できない場合は、JtronStreamIllegalStateException例外を発生する。

```
public synchronized OutputStream getOutputStream()
        throws JtronStreamIllegalStateException;
```

【戻り値】

OutputStream	送信ストリームクラス

【例外】

JtronStreamIllegalStateException	メソッド発行状態に関する例外クラス
STREAM_CLOSED	指定した識別子のストリームはすでにクローズしている。

【機能】

送信ストリームを取得する。取得できない場合は、JtronIllegalStateStreamException 例外を発生する。

```
public synchronized void setTimeout(int timeout)
        throws JtronStreamIllegalStateException;
```

【パラメータ】

int	timeout	タイムアウト時間(単位:ms)

【例外】

JtronStreamIllegalStateException	メソッド発行状態に関する例外クラス
STREAM_CLOSED	指定した識別子のストリームはすでにクローズしている。

【機能】

InputStream に対して read メソッドを実行した場合のタイムアウト時間(ミリ秒)を設定する。timeout に 0 を指定した場合ポーリング、負を指定した場合無限待ちとなる。タイムアウト発生時は java.io.InterruptedException 例外が発生する。なお、OutputStream に対して write メソッドを実行した場合は、他のストリーム操作同様タイムアウト指定はできない。

【JTRON2.0 仕様との相違】

パラメータ timeout の 0 の意味づけが無限待ちからポーリングに変更された。無限待ちはパラメータ timeout が負の値の時に変更された。

```
public synchronized int getTimeout()
        throws JtronStreamIllegalStateException;
```

【戻り値】

int	タイムアウト時間(単位:ms)

【例外】

JtronStreamIllegalStateException	メソッド発行状態に関する例外クラス
STREAM_CLOSED	指定した識別子のストリームはすでにクローズしている。

【機能】

InputStream に対して read メソッドを実行した場合のタイムアウト時間（ミリ秒）を取得する。戻り値が 0 の場合はポーリング、負の場合は無限待ちを意味する。

【JTRON2.0 仕様との相違】

戻り値の 0 の意味づけが無限待ちからポーリングに変更された。無限待ちは戻り値が負の値の時に変更された。

```
public synchronized void close()
        throws JtronStreamIllegalStateException;
```

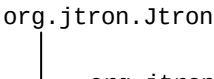
【例外】

JtronStreamIllegalStateException	メソッド発行状態に関する例外クラス
STREAM_CLOSED	指定した識別子のストリームはすでにクローズしている。

【機能】

ストリームをクローズする。送信用のチャネルが接続状態ならば正常 close して送信終了状態にし、受信用のチャネルが接続状態ならば強制 close して強制切断状態にする。また、受信用のチャネルが送信終了状態の場合は切断状態にする。

6.3.3 ストリーム関連の例外クラス (JtronStreamException)



public class JtronStreamException標準仕様

ストリーム通信関連の例外が発生したことを通知する。

■クラス定義

```
package org.jtron.stream;

public class JtronStreamException
                        extends org.jtron.JtronException {
        public JtronStreamException();
        public JtronStreamException(String msg);
}
```

■コンストラクタ

public JtronStreamException();

【機能】

詳細なメッセージ無しで JtronStreamException を生成する。

public JtronStreamException(String msg);

【パラメータ】

String msg 詳細メッセージ文字列

【機能】

指定された詳細メッセージ msg をもつ JtronStreamException を生成する。

【JTRON2.0 仕様との相違】

例外クラスの構成を共有オブジェクトクラス(type2)と合わせた。

6.3.4 メソッド発行状態に関する例外クラス (JtronStreamIllegalStateException)

```
org.jtron.stream.JtronStreamException
    |
    └── org.jtron.stream.JtronStreamIllegalStateException
```

public class JtronStreamIllegalStateException標準仕様

メソッドが実行できない不正な状態である時に発生する。

■クラス定義

```
package org.jtron.stream;

public class JtronStreamIllegalStateException
                        extends JtronStreamException{
        public static final int ILLEGAL_STREAM   = 1;
        public static final int STREAM_IN_USE    = 2;
        public static final int STREAM_NOEXIST   = 3;
        public static final int ILLEGAL_ARGUMENT = 4;
        public static final int STREAM_CLOSED    = 5;

        public JtronStreamIllegalStateException(int cause);
        public JtronStreamIllegalStateException(
                                int cause, String msg);

        public int getCause();
}
```

■定数

ILLEGAL_STREAM	1	ストリームインタフェースが不正(存在しない)。
STREAM_IN_USE	2	指定した識別子のストリームはすでに使用されている。
STREAM_NOEXIST	3	指定した識別子のストリームは存在

		しない(未生成)。
ILLEGAL_ARGUMENT	4	指定した引数が不正。
STREAM_CLOSED	5	指定した識別子のストリームはすでにクローズされている。

■コンストラクタ

```
public JtronStreamIllegalStateException(int cause);
```

【パラメータ】

int	cause	詳細原因

【機能】

詳細なメッセージ無しで JtronStreamIllegalStateException を生成する。パラメータ cause には、例外の詳細原因を渡す。

```
public JtronStreamIllegalStateException(int cause, String msg);
```

【パラメータ】

int	cause	詳細原因
String	msg	詳細メッセージ

【機能】

指定された詳細メッセージ msg を持つ JtronStreamIllegalStateException を生成する。パラメータ cause には、例外の詳細原因を渡す。

■メソッド

```
public int getCause();
```

【戻り値】

```
int                    詳細原因
```

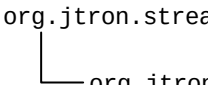
【機能】

例外の詳細原因を返す。

【JTRON2.0 仕様との相違】

例外クラスの構成を共有オブジェクトクラス(type2)と合わせた。

6.3.5 タイムアウトに関する例外クラス (JtronStreamTimeoutException)



public class JtronStreamTimeoutException標準仕様

タイムアウトに関する例外クラス。

■クラス定義

```
package org.jtron.stream;

public class JtronStreamTimeoutException
                        extends JtronStreamException {
        public JtronStreamTimeoutException();
        public JtronStreamTimeoutException(String msg);
}
```

■コンストラクタ

public JtronStreamTimeoutException();

【機能】

詳細なメッセージ無しで JtronStreamTimeoutException を生成する。

public JtronStreamTimeoutException(String msg);

【パラメータ】

	String msg	詳細メッセージ文字列

【機能】

指定された詳細メッセージ msg をもつ JtronStreamTimeoutException を生成する。

【JTRON2.0 仕様との相違】

タイムアウト時に InterruptedException が発生するのを、例外クラスの構成を共有オブジェクトクラス(type2)と合わせた。

6.4 参考情報

ストリームの Java 側のクラス構成例を示す。ここではブリッジパターンを用いて仕様と実装の分離を行う。JtronStream クラスの各メソッドは実装クラスの抽象クラス JtronStreamImpl のメソッドを呼びだすように記述し、実際のメソッドの実装は PlainJtronStream クラスで行う。ブリッジパターンを用いることにより PlainJtronStream クラスを交換するだけでさまざまな機器に対応することができる。

6.4.1 パッケージ構成図

以下のクラスから構成される。

クラス： JtronStream
ベンダ実装クラス： JtronStreamImpl,PlainJtronStream,
JtronInputStream, JtronOutputStream

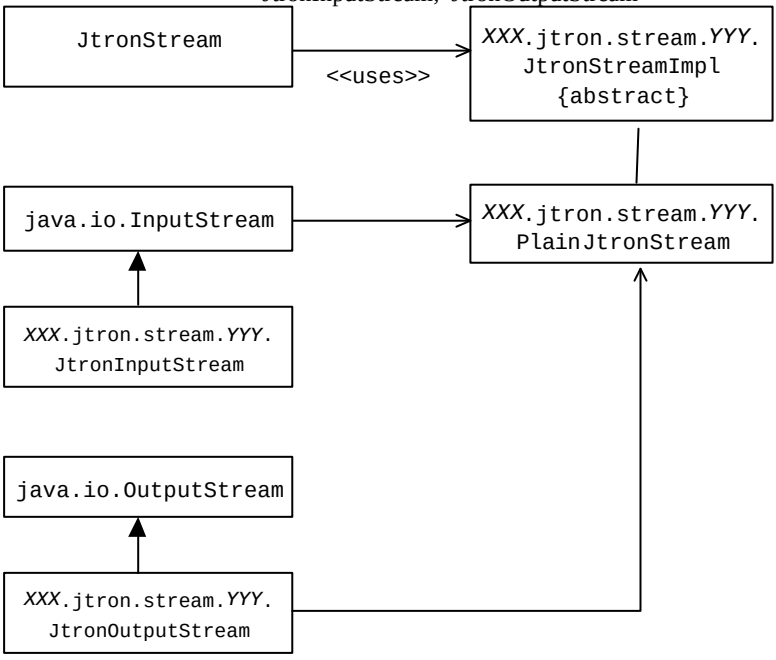


図 6.7:ベンダ実装クラス構成

6.4.2 ストリームインタフェースの実装を持つ抽象クラス (JtronStreamImpl)

```
java.lang.Object
    |
    +--- XXX.jtron.stream.YYY.JtronStreamImpl
```

public abstract class JtronStreamImpl

ストリームインタフェースの実装を持つ抽象クラス(abstract)。

ストリームインタフェースの実装をもつクラスを定義するための抽象クラス。仕様と実装を分離するために用意されたクラスである。ベンダ実装のパッケージに作成する。

■クラス定義

```
package XXX.jtron.stream.YYY;

public abstract class JtronStreamImpl {
        public abstract void setStreamId(int stmid);
        public abstract int getStreamId();
        public abstract InputStream getInputStream();
        public abstract OutputStream getOutputStream();
        public abstract void setTimeout(int timeout);
        public abstract int getTimeout();
        public abstract void close();
}
```

■メソッド

public abstract void setStreamId(int stmid);

【パラメータ】

int	stmid	ストリーム識別子

【機能】

識別子を設定する。

```
public abstract int getStreamId();
```

【戻り値】

int	ストリーム識別子

【機能】

識別子を取得する。

```
public abstract InputStream getInputStream()
        throws JtronStreamIllegalStateException;
```

【戻り値】

InputStream	受信ストリームクラス

【例外】

JtronStreamIllegalStateException	メソッド発行状態に関する例外クラス。
STREAM_CLOSED	指定した識別子のストリームはすでにクローズしている。

【機能】

受信ストリームを取得する。取得できない場合は、JtronStreamException例外を発生する。

```
public abstract OutputStream getOutputStream()
        throws JtronStreamIllegalStateException;
```

【戻り値】

OutputStream	送信ストリームクラス

【例外】

JtronStreamIllegalStateException	メソッド発行状態に関する例外クラス。
STREAM_CLOSED	指定した識別子のストリームはすでにクローズしている。

【機能】

送信ストリームを取得する。取得できない場合は、JtronStreamException 例外を発生する。

```
public abstract void setTimeout(int timeout)
        throws JtronStreamIllegalStateException;
```

【パラメータ】

int	timeout	タイムアウト時間(単位:ms)

【例外】

JtronStreamIllegalStateException	メソッド発行状態に関する例外クラス。
STREAM_CLOSED	指定した識別子のストリームはすでにクローズしている。

【機能】

InputStream に対して read メソッドを実行した場合のタイムアウト時間(ミリ秒)を設定する。timeout に 0 を指定した場合ポーリング、負の値を指定した場合無限待ちとなる。タイムアウト発生時は java.io.InterruptedException 例外が発生する。なお、OutputStream に対して write メソッドを実行した場合は、他のストリーム操作同様タイムアウト指定はできない。

```
public abstract int getTimeout()
        throws JtronStreamIllegalStateException;
```

【戻り値】

int	タイムアウト時間(単位:ms)

【例外】

JtronStreamIllegalStateException	メソッド発行状態に関する例外クラス。
STREAM_CLOSED	指定した識別子のストリームはすでにクローズしている。

【機能】

InputStream に対して read メソッドを実行した場合のタイムアウト時間（ミリ秒）を取得する。戻り値が 0 の場合はポーリング、負の値の場合は無限待ちを意味する。

```
public abstract void close()
        throws JtronStreamIllegalStateException;
```

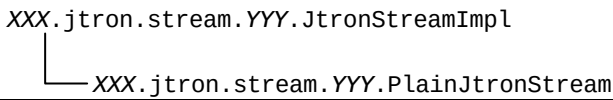
【例外】

JtronStreamIllegalStateException	メソッド発行状態に関する例外クラス。
STREAM_NOEXIST	指定した識別子のストリームは存在しない(未生成)。
STREAM_CLOSED	指定した識別子のストリームはすでにクローズしている。

【機能】

ストリームをクローズする。送信用のチャネルが接続状態ならば正常 close して送信終了状態にし、受信用のチャネルが接続状態ならば強制 close して強制切断状態にする。また、受信用のチャネルが送信終了状態の場合は切断状態にする。

6.4.3 ストリームインタフェースの実装クラス (PlainJtronStream)



public class PlainJtronStream
ストリームインタフェースの実装クラス。

ストリームインタフェースの実装をもつクラス。

■クラス定義

```
package XXX.jtron.stream.YYY;

public class PlainJtronStreamImpl extends JtronStreamImpl {
        public PlainJtronStream(int stmid)
        public PlainJtronStream(int stmid, int timeout)

        public void setStreamId(int stmid);
        public int getStreamId();
        public InputStream getInputStream();
        public OutputStream getOutputStream();
        public void setTimeout(int timeout);
        public int getTimeout();
        public void close();
}
```

■コンストラクタ

```
public PlainJtronStream(int stmid)
         throws JtronStreamIllegalStateException;
```

【例外】

JtronStreamIllegalStateException	メソッド発行状態に関する例外クラス。
STREAM_IN_USE	指定した識別子のストリームはすでに使用されている。
ILLEGAL_ARGUMET	指定した引数が不正。

【パラメータ】

int	stmid	ストリーム識別子

【機能】

指定した識別子のストリームをオープンする。指定した識別子が既に使用されている場合は、JtronStreamIllegalStateException 例外が発生する。

```
public PlainJtronStream(int stmid, int timeout)
        throws JtronStreamIllegalStateException,
               JtronStreamTimeoutException;
```

【例外】

JtronStreamIllegalStateException	メソッド発行状態に関する例外クラス。
STREAM_IN_USE	指定した識別子のストリームはすでに使用されている。
ILLEGAL_ARGUMET	指定した引数が不正。
JtronStreamTimeoutException	タイムアウトの例外が発生。

【パラメータ】

int	stmid	ストリーム識別子
int	timeout	タイムアウト時間(単位:ms)

【機能】

タイムアウト時間（ミリ秒）を指定し、指定した識別子のストリームをオープンする。指定した識別子が既に使用されている場合は、JtronStreamIllegalStateException 例外が発生し、タイムアウト発生時は JtronStreamTimeoutException 例外が発生する。

■メソッド

JtronStreamImpl の各メソッドを実装する。また、必要な protected、private メソッドを定義、実装する。

第 7 章 リファレンス

7.1 ITRON API 一覧

7.1.1 共通仕様

(1) Java スレッドとリアルタイムタスクの対応

```
ER ercd = jti_set_hpr(PRI hijpr);
ER ercd = jti_get_hpr(PRI *p_hijpr);
ER ercd = jti_cnv_jpr(INT jpr, PRI *p_pri);
ER ercd = jti_cnv_lpr(PRI *p_lwjpr);
```

・静的 API 一覧

```
JTI_SET_HPR(PRI hijpr);
PRI pri = JTI_CNV_JPR(PRI hijpr, INT jpr);
PRI lwjpr = JTI_CNV_LPR(PRI hijpr);
```

7.1.2 タイプ 2 インタフェース

(1) 共有オブジェクトアクセスのための ITRON API

```
ER ercd = jti_get_obj(const char *objnm, JNO *p_objno);
ER ercd =jti_get_<type>(JNO objno, const char * clsnm,
          const char *fidnm, <c_type> *p_retval);
ER ercd =jti_set_<type>(JNO objno, const char * clsnm,
                  const char *fidnm, <c_type> val);
ER ercd = jti_loc_obj(JNO objno, TMO tmout);
ER ercd = jti_unl_obj(JNO objno);
ER ercd = jti_funl_obj(JNO objno);
```

(2) Java スレッド操作のための ITRON API

```
ER ercd = jti_get_thr(const char *thrnm, JNO *p_thrno);
ER_BOOL ercd = jti_isa_thr(JNO thrno);
ER ercd = jti_int_thr(JNO thrno);
ER_BOOL ercd = jti_isi_thr(JNO thrno);
ER ercd = jti_sus_thr(JNO thrno);
ER ercd = jti_rsm_thr(JNO thrno);
ER ercd = jti_sta_thr(JNO thrno);
ER ercd = jti_stp_thr(JNO thrno);
ER ercd = jti_get_jpr(JNO thrno, INT *p_rslt);
ER ercd = jti_set_jpr(JNO thrno, INT newpri);
ER ercd = jti_des_thr(JNO thrno);
```

(3) Java スレッドグループ操作のための ITRON API

```
ER ercd = jti_get_tgr(const char *tgrnm, JNO *p_tgrno);
ER ercd = jti_des_tgr(JNO tgrno);
ER ercd = jti_sus_tgr(JNO tgrno);
ER ercd = jti_rsm_tgr(JNO tgrno);
ER ercd = jti_stp_tgr(JNO tgrno);
```

7.1.3 タイプ 3 インタフェース

(1) ストリームを操作するための ITRON API

```
ER ercd = jti_cre_stm(ID stmid, T_JTI_CSTM *pk_cstm);
ER ercd = jti_del_stm(ID stmid);
ER ercd = jti_wri_stm(ID stmid, const VP data,
                      INT len, TMO timeout);
ER ercd = jti_rea_stm(ID stmid, VP data,
                      INT len, TMO timeout);
ER ercd = jti_sht_stm(ID stmid);
ER ercd = jti_ref_stm(ID stmid, T_JTI_RSTM *pk_rstm);
```

・静的 API 一覧

```
JTI_CRE_STM(ID stmid,
            { VP exinf, ATR stmpatr, VP wbuf,
              INT wbufsz, VP rbuf, INT rbufsz });
```

7.2 JTRON API 一覧

7.2.1 共通仕様

(1) JTRON システムクラス

```
public class JtronSystem {
  public static String getProperty(String key);
  public static String getProperty(String key,
                        String defaultValue);
  public static Properties getProperties();
}
```

(2) JTRON 例外クラス

```
public class JtronException extends Exception {
  public JtronException();
  public JtronException(String msg);
}
```

7.2.2 タイプ 1 インタフェース

(1) ITRON による例外クラス

```
public class ItronCauseException
                        extends org.jtron.JtronException {
  public ItronCauseException(String message,int resourceId,
                        int functionCode, int errorCode);
        public int resourceId;
        public int functionCode;
        public int errorCode;
  public final int MERCD();
  public final int SERCD();
        public static final int // ITRON 機能コード一覧
          // -0x01～-0x04 までは予約
          TFN_CRE_TSK   = -0x05, TFN_DEL_TSK   = -0x06,
          TFN_ACT_TSK   = -0x07, TFN_CAN_ACT   = -0x08,
          TFN_STA_TSK   = -0x09, TFN_EXT_TSK   = -0x0a,
          TFN_EXD_TSK   = -0x0b, TFN_TER_TSK   = -0x0c,
          TFN_CHG_PRI   = -0x0d, TFN_GET_PRI   = -0x0e,
          TFN_REF_TSK   = -0x0f, TFN_REF_TST   = -0x10,
          TFN_SLP_TSK   = -0x11, TFN_TSLP_TSK  = -0x12,
          TFN_WUP_TSK   = -0x13, TFN_CAN_WUP   = -0x14,
          TFN_REL_WAI   = -0x15, TFN_SUS_TSK   = -0x16,
          TFN_RSM_TSK   = -0x17, TFN_FRSM_TSK  = -0x18,
          TFN_DLY_TSK   = -0x19, // -0x1a は予約
          TFN_DEF_TEX   = -0x1b, TFN_RAS_TEX   = -0x1c,
          TFN_DIS_TEX   = -0x1d, TFN_ENA_TEX   = -0x1e,
          TFN_SNS_TEX   = -0x1f, TFN_REF_TEX   = -0x20,
          TFN_CRE_SEM   = -0x21, TFN_DEL_SEM   = -0x22,
          TFN_SIG_SEM   = -0x23, // -0x24 は予約
          TFN_WAI_SEM   = -0x25, TFN_POL_SEM   = -0x26,
          TFN_TWAI_SEM  = -0x27, TFN_REF_SEM   = -0x28,
          TFN_CRE_FLG   = -0x29, TFN_DEL_FLG   = -0x2a,
          TFN_SET_FLG   = -0x2b, TFN_CLR_FLG   = -0x2c,
          TFN_WAI_FLG   = -0x2d, TFN_POL_FLG   = -0x2e,
          TFN_TWAI_FLG  = -0x2f, TFN_REF_FLG   = -0x30,
          TFN_CRE_DTQ   = -0x31, TFN_DEL_DTQ   = -0x32,
```

```
// -0x33～-0x34までは予約
TFN_SND_DTQ   = -0x35, TFN_PSND_DTQ  = -0x36,
TFN_TSND_DTQ  = -0x37, TFN_FSND_DTQ  = -0x38,
TFN_RCV_DTQ   = -0x39, TFN_PRCV_DTQ  = -0x3a,
TFN_TRCV_DTQ  = -0x3b, TFN_REF_DTQ   = -0x3c,
TFN_CRE_MBX   = -0x3d, TFN_DEL_MBX   = -0x3e,
TFN_SND_MBX   = -0x3f, // -0x40は予約
TFN_RCV_MBX   = -0x41, TFN_PRCV_MBX  = -0x42,
TFN_TRCV_MBX  = -0x43, TFN_REF_MBX   = -0x44,
TFN_CRE_MPF   = -0x45, TFN_DEL_MPF   = -0x46,
TFN_REL_MPF   = -0x47, // -0x48は予約
TFN_GET_MPF   = -0x49, TFN_PGET_MPF  = -0x4a,
TFN_TGET_MPF  = -0x4b, TFN_REF_MPF   = -0x4c,
TFN_SET_TIM   = -0x4d, TFN_GET_TIM   = -0x4e,
TFN_CRE_CYC   = -0x4f, TFN_DEL_CYC   = -0x50,
TFN_STA_CYC   = -0x51, TFN_STP_CYC   = -0x52,
TFN_REF_CYC   = -0x53, // -0x54は予約
TFN_ROT_RDQ   = -0x55, TFN_GET_TID   = -0x56,
// -0x57～-0x58までは予約
TFN_LOC_CPU   = -0x59, TFN_UNL_CPU   = -0x5a,
TFN_DIS_DSP   = -0x5b, TFN_ENA_DSP   = -0x5c,
TFN_SNS_CTX   = -0x5d, TFN_SNS_LOC   = -0x5e,
TFN_SNS_DSP   = -0x5f, TFN_SNS_DPN   = -0x60,
TFN_REF_SYS   = -0x61, // -0x62は予約
// -0x63～-0x64までは予約
TFN_DEF_INH   = -0x65, TFN_CRE_ISR   = -0x66,
TFN_DEL_ISR   = -0x67, TFN_REF_ISR   = -0x68,
TFN_DIS_INT   = -0x69, TFN_ENA_INT   = -0x6a,
TFN_CHG_IXX   = -0x6b, TFN_GET_IXX   = -0x6c,
TFN_DEF_SVC   = -0x6d, TFN_DEF_EXC   = -0x6e,
TFN_REF_CFG   = -0x6f, TFN_REF_VER   = -0x70,
TFN_IACT_TSK  = -0x71, TFN_IWUP_TSK  = -0x72,
TFN_IREL_WAI  = -0x73, TFN_IRAS_TEX  = -0x74,
TFN_ISIG_SEM  = -0x75, TFN_ISET_FLG  = -0x76,
TFN_IPSND_DTQ = -0x77, TFN_IFSND_DTQ = -0x78,
TFN_IROT_RDQ  = -0x79, TFN_IGET_TID  = -0x7a,
TFN_ILOC_CPU  = -0x7b, TFN_IUNL_CPU  = -0x7c,
```

```
TFN_ISIG_TIM  = -0x7d, // -0x7e は予約
// -0x7f～-0x80 までは予約
TFN_CRE_MTX   = -0x81, TFN_DEL_MTX   = -0x82,
TFN_UNL_MTX   = -0x83, // -0x84 は予約
TFN_LOC_MTX   = -0x85, TFN_PLOC_MTX  = -0x86,
TFN_TLOC_MTX  = -0x87, TFN_REF_MTX   = -0x88,
TFN_CRE_MBF   = -0x89, TFN_DEL_MBF   = -0x8a,
// -0x8b～-0x8c までは予約
TFN_SND_MBF   = -0x8d, TFN_PSND_MBF  = -0x8e,
TFN_TSND_MBF  = -0x8f, // -0x90 は予約
TFN_RCV_MBF   = -0x91, TFN_PRCV_MBF  = -0x92,
TFN_TRCV_MBF  = -0x93, TFN_REF_MBF   = -0x94,
TFN_CRE_POR   = -0x95, TFN_DEL_POR   = -0x96,
TFN_CAL_POR   = -0x97, TFN_TCAL_POR  = -0x98,
TFN_ACP_POR   = -0x99, TFN_PACP_POR  = -0x9a,
TFN_TACP_POR  = -0x9b, TFN_FWD_POR   = -0x9c,
TFN_RPL_RDV   = -0x9d, TFN_REF_POR   = -0x9e,
TFN_REF_RDV   = -0x9f, // -0xa0 は予約
TFN_CRE_MPL   = -0xa1, TFN_DEL_MPL   = -0xa2,
TFN_REL_MPL   = -0xa3, // -0xa4 は予約
TFN_GET_MPL   = -0xa5, TFN_PGET_MPL  = -0xa6,
TFN_TGET_MPL  = -0xa7, TFN_REF_MPL   = -0xa8,
TFN_CRE_ALM   = -0xa9, TFN_DEL_ALM   = -0xaa,
TFN_STA_ALM   = -0xab, TFN_STP_ALM   = -0xac,
TFN_REF_ALM   = -0xad, // -0xae は予約
// -0xaf～-0xb0 までは予約
TFN_DEF_OVR   = -0xb1, TFN_STA_OVR   = -0xb2,
TFN_STP_OVR   = -0xb3, TFN_REF_OVR   = -0xb4,
// -0xb5～-0xc0 までは予約
TFN_ACRE_TSK  = -0xc1, TFN_ACRE_SEM  = -0xc2,
TFN_ACRE_FLG  = -0xc3, TFN_ACRE_DTQ  = -0xc4,
TFN_ACRE_MBX  = -0xc5, TFN_ACRE_MTX  = -0xc6,
TFN_ACRE_MBF  = -0xc7, TFN_ACRE_POR  = -0xc8,
TFN_ACRE_MPF  = -0xc9, TFN_ACRE_MPL  = -0xca,
TFN_ACRE_CYC  = -0xcb, TFN_ACRE_ALM  = -0xcc,
TFN_ACRE_ISR  = -0xcd; // -0xce は予約
// -0xcf～-0xe0 までは予約
```

```
        // -0xe1～-0xff までは実装独自のサービスコール
        public static final int // エラーコード一覧
         E_SYS   = -5,     // システムエラー
         E_NOSPT = -9,     // 未サポート機能
         E_RSFN  = -10,    // 予約機能コード
         E_RSTAR = -11,    // 予約属性
         E_PAR   = -17,    // パラメータエラー
         E_ID    = -18,    // 不正 ID 番号
         E_CTX   = -25,    // コンテキストエラー
         E_MACV  = -26,    // メモリアクセス違反
         E_OACV  = -27,    // オブジェクトアクセス違反
         E_ILUSE = -28,    // サービスコール不正使用
         E_NOMEM = -33,    // メモリ不足
         E_NOID  = -34,    // ID 番号不足
         E_OBJ   = -41,    // オブジェクト状態エラー
         E_NOEXS = -42,    // オブジェクト未生成
         E_QOVR  = -43,    // キューイングオーバーフロー
         E_RLWAI = -49,    // 待ち状態の強制解除
         E_TMOUT = -50,    // ポーリング失敗またはタイムアウト
         E_DLT   = -51,    // 待ちオブジェクトの削除
         E_CLS   = -52,    // 待ちオブジェクトの状態変化
         E_WBLK  = -57,    // ノンブロッキング受付け
         E_BOVR  = -58;    // バッファオーバーフロー
}
```

(2) **システムエラー**

```
public class ItronSYSException extends ItronCauseException {
  public ItronSYSException(String message,
         int resourceId, int functionCode, int errorCode);
}
```

(3) **未サポート機能**

```
public class ItronNOSPTException extends ItronCauseException {
  public ItronNOSPTException(String message,
         int resourceId, int functionCode, int errorCode);
}
```

(4) **予約機能コード**

```
public class ItronRSFNException extends ItronCauseException {
  public ItronRSFNException(String message,
```

```
        int resourceId, int functionCode, int errorCode);
}
```

(5) **予約属性**

```
public class ItronRSATRException extends ItronCauseException {
  public ItronRSATRException(String message,
           int resourceId, int functionCode, int errorCode);
}
```

(6) **パラメータエラー**

```
public class ItronPARException extends ItronCauseException {
  public ItronPARException(String message,
           int resourceId, int functionCode, int errorCode);
}
```

(7) **不正 ID 番号**

```
public class ItronIDException extends ItronCauseException {
  public ItronIDException(String message,
           int resourceId, int functionCode, int errorCode);
}
```

(8) **コンテキストエラー**

```
public class ItronCTXException extends ItronCauseException {
  public ItronCTXException(String message,
           int resourceId, int functionCode, int errorCode);
}
```

(9) **メモリアクセス違反**

```
public class ItronMACVException extends ItronCauseException {
  public ItronMACVException(String message,
           int resourceId, int functionCode, int errorCode);
}
```

(10) **オブジェクトアクセス違反**

```
public class ItronOACVException extends ItronCauseException {
  public ItronOACVException(String message,
           int resourceId, int functionCode, int errorCode);
}
```

(11) **サービスコール不正使用**

```
public class ItronILUSEException extends ItronCauseException {
        public ItronILUSEException(String message,
           int resourceId, int functionCode, int errorCode);
}
```

(12) メモリ不足

```
public class ItronNOMEMException extends ItronCauseException {
  public ItronILUSEException(String message,
            int resourceId, int functionCode, int errorCode);
}
```

(13) ID 番号不正

```
public class ItronNOIDException extends ItronCauseException {
  public ItronNOIDException(String message,
            int resourceId, int functionCode, int errorCode);
}
```

(14) オブジェクト状態エラー

```
public class ItronOBJException extends ItronCauseException {
  public ItronOBJException(String message,
           int resourceId, int functionCode, int errorCode);
}
```

(15) オブジェクト未生成

```
public class ItronNOEXSException extends ItronCauseException {
  public ItronNOEXSException(String message,
            int resourceId, int functionCode, int errorCode);
}
```

(16) キューイングオーバーフロー

```
public class ItronQOVRException extends ItronCauseException {
  public ItronQOVRException(String message,
            int resourceId, int functionCode, int errorCode);
}
```

(17) 待ち状態の強制解除

```
public class ItronRLWAIException extends ItronCauseException {
  public ItronRLWAIException(String message,
            int resourceId, int functionCode, int errorCode);
}
```

(18) ポーリング失敗またはタイムアウト

```
public class ItronTMOUTException extends ItronCauseException {
  public ItronTMOUTException(String message,
            int resourceId, int functionCode, int errorCode);
}
```

(19) 待ちオブジェクトの削除

```
public class ItronDLTException extends ItronCauseException {
```

```
  public ItronDLTException(String message,
           int resourceId, int functionCode, int errorCode);
}
```

(20) **待ちオブジェクトの状態変化**

```
public class ItronCLSException extends ItronCauseException {
  public ItronCLSException(String message,
           int resourceId, int functionCode, int errorCode);
}
```

(21) **ノンブロッキング受付け**

```
public class ItronWBLKException extends ItronCauseException {
  public ItronWBLKException(String message,
            int resourceId, int functionCode, int errorCode);
}
```

(22) **バッファオーバーフロー**

```
public class ItronBOVRException extends ItronCauseException {
  public ItronBOVRException(String message,
            int resourceId, int functionCode, int errorCode);
}
```

(23) JTRON **による例外クラス**

```
public class JtronCauseException
                        extends org.jtron.JtronException {
        // 実装定義
}
```

(24) **メモリ操作クラス**

```
public class ItronMemory {
  public ItronMemory(int length) throws JtronException;
  public void release() throws JtronException;
  public int getLength() throws JtronCauseException;
  public void disableWrite() throws JtronCauseException;
  public void enableWrite() throws JtronCauseException;
  public boolean isWriteable() throws JtronCauseException;
  public void seek(int offset) throws JtronCauseException;
  public void skipBytes(int n) throws JtronCauseException;
  public int getOffset() throws JtronCauseException;
  public byte readB() throws JtronCauseException;
  public byte readB(int offset) throws JtronCauseException;
  public short readH() throws JtronCauseException;
```

```
public short readH(int offset) throws JtronCauseException;
public int readW() throws JtronCauseException;
public int readW(int offset) throws JtronCauseException;
public long readD() throws JtronCauseException;
public long readD(int offset) throws JtronCauseException;
public short readUB() throws JtronCauseException;
public short readUB(int offset) throws JtronCauseException;
public int readUH() throws JtronCauseException;
public int readUH(int offset) throws JtronCauseException;
public long readUW() throws JtronCauseException;
public long readUW(int offset) throws JtronCauseException;
public int read(byte[] b) throws JtronCauseException;
public int read(int offset, byte[] b)
              throws JtronCauseException;
public int read(byte[] b, int bOff, int len)
              throws JtronCauseException;
public int read(int offset, byte[] b, int bOff, int len)
              throws JtronCauseException;
public void writeB(byte value) throws JtronCauseException;
public void writeB(int offset, byte value)
              throws JtronCauseException;
public void writeH(short value) throws JtronCauseException;
public void writeH(int offset, short value)
               throws JtronCauseException;
public void writeW(int value) throws JtronCauseException;
public void writeW(int offset, int value)
               throws JtronCauseException;
public void writeD(long value) throws JtronCauseException;
public void writeD(int offset, long value)
               throws JtronCauseException;

public int write(byte[] b) throws JtronCauseException;
public int write(int offset, byte[] b)
               throws JtronCauseException;
public int write(int byte[], int bOff, int len)
               throws JtronCauseException;
public int write(int offset, byte[] b, int bOff, int len)
```

```
                throws JtronCauseException;
}
```

(25) **タスクアタッチクラス**

```
public class Task {
  public Task(int tskid) throws JtronException;
  public Task(Thread thread) throws JtronCauseException;
  public Task(int tskid, T_CTSK pk_ctsk)
                throws ItronCauseException;
  public Task(T_CTSK pk_ctsk)  throws ItronCauseException;
  public int getId();
  public static Task currentTask() throws JtronCauseException;
  public void delete() throws ItronCauseException;
  public void activate() throws ItronCauseException;
  public int cancelActivate() throws ItronCauseException;
  public void start(int stacd) throws ItronCauseException;
  public void terminate() throws ItronCauseException;
  public void changePriority(int tskpri)
                throws ItronCauseException;
  public int getPriority() throws ItronCauseException;
  public T_RTSK refer() throws ItronCauseException;
  public T_RTST referSimple() throws ItronCauseException;
  public static void sleep() throws ItronCauseException;
  public static void sleep(int timeout)
                throws ItronCauseException;
  public void wakeup() throws ItronCauseException;
  public int cancelWakeup() throws ItronCauseException;
  public void releaseWait() throws ItronCauseException;
  public void suspend() throws ItronCauseException;
  public void resume() throws ItronCauseException;
  public void forceResume() throws ItronCauseException;
  public static void delay(int dlytim)
                throws ItronCauseException;
  public void defineTaskException(T_DTEX pk_dtex)
                throws ItronCauseException;
  public void raiseTaskException(int rasptn)
                throws ItronCauseException;
  public T_RTEX referTaskException()
```

```
                throws ItronCauseException;
  public void startOverrunHandler(int ovrtim)
                throws ItronCauseException;
  public void stopOverrunHandler( ) throws ItronCauseException;
  public T_ROVR referOverrunHandler()
                throws ItronCauseException;
}
```

(26) **タスク生成情報クラス**

```
public class T_CTSK {
  public T_CTSK(String task, int itskpri, int stksz);
  public T_CTSK(int tskatr, int exinf,
               String task, int itskpri, int stksz);
        public int tskatr;
        public int exinf;
        public String task;
        public int itskpri;
        public int stksz;
        public static final int
                TA_HLNG = 0x00,
                TA_ASM  = 0x01;
        public static final int
                TA_ACT  = 0x02,
                TA_RSTR = 0x04;
 // 下層のμITRON4.0 仕様 OS に合わせた実装独自のフィールド
}
```

(27) **簡易タスク状態クラス**

```
public class T_RTST {
        public int tskstat;
        public int tskwait;
        public static final int
                TTS_RUN = 0x01,
                TTS_RDY = 0x02,
                TTS_WAI = 0x04,
                TTS_SUS = 0x08,
                TTS_WAS = 0x0C,
                TTS_DMT = 0x10;
        public static final int
```

```
            TTW_SLP  = 0x0001,
            TTW_DLY  = 0x0002,
            TTW_SEM  = 0x0004,
            TTW_FLG  = 0x0008,
            TTW_SDTQ = 0x0010,
            TTW_RDTQ = 0x0020,
            TTW_MBX  = 0x0040,
            TTW_MTX  = 0x0080,
            TTW_SMBF = 0x0100,
            TTW_RMBF = 0x0200,
            TTW_CAL  = 0x0400,
            TTW_ACP  = 0x0800,
            TTW_RDV  = 0x1000,
            TTW_MPF  = 0x2000,
            TTW_MPL  = 0x4000;
 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```

(28) **タスク状態クラス**

```
public class T_RTSK extends T_RTST {
        public int tskpri;
        public int tskbpri;
        public int wobjid;
        public int lefttmo;
        public int actcnt;
        public int wupcnt;
        public int suscnt;
 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```

(29) **タスク例外処理定義情報クラス**

```
public class T_DTEX {
  public T_DTEX(String texrtn);
  public T_DTEX(int texatr, String texrtn);
        public int texatr;
        public String texrtn;
        public static final int
                TA_HLNG = 0x00,
                TA_ASM  = 0x01;
```

```
 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```
(30) タスク例外状態クラス
```
public class T_RTEX {
        public int texstat;
        public int pndptn;
        public static final int
                TTEX_ENA = 0x00,
                TTEX_DIS = 0x01;
 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```
(31) オーバーランハンドラ状態クラス
```
public class T_ROVR {
        public int ovrstat;
        public int leftotm;
        public static final int
                TOVR_STP = 0x00,
                TOVR_STA = 0x01;
 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```
(32) セマフォアタッチクラス
```
public class Semaphore {
  public Semaphore(int semid) throws JtronException;
  public Semaphore(int semid, T_CSEM pk_csem)
                throws ItronCauseException;
  public Semaphore(T_CSEM csem) throws ItronCauseException;
  public int getId();
  public void delete() throws ItronCauseException;
  public void signal() throws ItronCauseException;
  public void waitSemaphore() throws ItronCauseException;
  public void poll() throws ItronCauseException;
  public void waitSemaphore(int timeout)
                throws ItronCauseException;
  public T_RSEM refer() throws JtronException;
}
```
(33) セマフォ生成情報クラス
```
public class T_CSEM {
```

```
  public T_CSEM(int cnt);
  public T_CSEM(int sematr, int semcnt, int maxsem);
        public int sematr;
        public int isemcnt;
        public int maxsem;
        public static final int
                TA_TFIFO = 0x00,
                TA_TPRI  = 0x01;
 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```

(34) **セマフォ状態クラス**

```
public class T_RSEM {
        public int wtskid;
        public int semcnt;
 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```

(35) **イベントフラグアタッチクラス**

```
public class EventFlag {
  public EventFlag(int flgid) throws JtronException;
  public EventFlag(int flgid, T_CFLG pk_cflg)
                throws ItronCauseException;
  public EventFlag(T_CFLG pk_cflg) throwsItronCauseException;
  public int getId();
  public void delete() throws ItronCauseException;
  public void set(int setptn) throws ItronCauseException;
  public void clear(int clrptn) throws ItronCauseException;
  public int waitFlag(int waiptn, int wfmode)
                throws ItronCauseException;
  public int poll(int waiptn, int wfmode)
                throws ItronCauseException;
  public int waitFlag(int waiptn, int wfmode, int tmout)
                throws ItronCauseException;
  public int T_RFLG refer() throws JtronException;
        public static final int
                TWF_ANDW = 0x00,
                TWF_ORW  = 0x01;
}
```

(36) **イベントフラグ生成情報クラス**

```
public class T_CFLG {
  public T_CFLG();
  public T_CFLG(int flgatr, int iflgptn);
        public int flgatr;
        public int iflgptn;
        public static final int
                TA_TFIFO = 0x00,
                TA_TPRI  = 0x01;
        public static final int
                TA_WSGL  = 0x00,
                TA_WMUL  = 0x02;
        public static final int
                TA_CLR   = 0x04;
 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```

(37) **イベントフラグ状態クラス**

```
public class T_RFLG {
        public int wtskid;
        public int flgptn;
 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```

(38) **データキューアタッチクラス**

```
public class DataQueue {
  public DataQueue(int dtqid) throws JtronException;
  public DataQueue(int dtqid, T_CDTQ pk_cdtq)
                throws ItronCauseException;
  public DataQueue(T_CDTQ pk_cdtq) throws ItronCauseException;
  public int getId();
  public void delete() throws ItronCauseException;
  public void send(ItronMemory data) throws ItronCauseException;
  public void pollSend(ItronMemory data)
                throws ItronCauseException;
  public void send(ItronMemory data, int tmout)
                throws ItronCauseException;
  public void forceSend(ItronMemory data)
                throws ItronCauseException;
```

```
  public void sendValue(int data) throws ItronCauseException;
  public void pollSendValue(int data)
                throws ItronCauseException;
  public void sendValue(int data, int tmout)
                throws ItronCauseException;
  public void forceSendValue(int data)
                throws ItronCauseException;
  public ItronMemory receive(int length) throws JtronException;
  public ItronMemory pollReceive(int length)
                throws JtronException;
  public ItronMemory receive(int length, int tmout)
                throws JtronException;
  public int receiveValue() throws ItronCauseException;
  public int pollReceiveValue() throws ItronCauseException;
  public int receiveValue(int tmout) throws ItronCauseException;
  public T_RDTQ refer() throws JtronException;
}
```

(39) **データキュー生成情報クラス**

```
public class T_CDTQ {
  public T_CDTQ(int dtqcnt);
  public T_CDTQ(int dtqatr, int dtqcnt);
        public int dtqatr;
        public int dtqcnt;
        public static final int
                TA_TFIFO = 0x00,
                TA_TPRI  = 0x01;
  public static final int TSZ_DTQ(int dtqcnt);
 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```

(40) **データキュー状態クラス**

```
public class T_RDTQ {
        public int stskid;
        public int rtskid;
        public int sdtqcnt;
 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```

(41) **メールボックスアタッチクラス**

```
public class MailBox {
  public MailBox(int mbxid) throws JtronException;
  public MailBox(int mbxid, T_CMBX pk_cmbx)
                throws ItronCauseException;
  public MailBox(T_CMBX pk_cmbx) throws ItronCauseException;
  public int getId();
  public void delete() throws ItronCauseException;
  public void send(ItronMemory pk_msg)
                throws ItronCauseException;
  public ItronMemory receive(int length) throws JtronException;
  public ItronMemory pollReceive(int length)
                throws JtronException;
  public ItronMemory receive(int length, int tmout)
                throws JtronException;
  public T_RMBX refer(int length) throws JtronException;
  public static void seekNextToHeader(ItronMemory msg);
  public static int readPriority(ItronMemory msg);
  public static void writePriority(ItronMemory msg, int msgpri);
}
```

(42) **メールボックス生成情報クラス**

```
public class T_CMBX {
  public T_CMBX();
  public T_CMBX(int mbxatr, int maxmpri);
        public int mbxatr;
        public int maxmpri;
        public static final int
                TA_TFIFO = 0x00,
                TA_TPRI  = 0x01;
        public static final int
                TA_MFIFO = 0x00,
                TA_MPRI  = 0x02;
  public static final int TSZ_MPRIHD(int maxmpri);
 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```

(43) **メールボックス状態クラス**

```
public class T_RMBX {
        public int wtskid;
```

```
        public ItronMemory pk_msg;
 // 下層のμITRON4.0 仕様 OS に合わせた実装独自のフィールド
}
```

(44) ミューテックスアタッチクラス

```
public class Mutex {
  public Mutex(int mtxid) throws JtronException;
  public Mutex(int mtxid, T_CMTX pk_cmtx)
                throws ItronCauseException;
  public Mutex(T_CMTX pk_cmtx) throws ItronCauseException;
  public int getId();
  public void delete() throws ItronCauseException;
  public void lock() throws ItronCauseException;
  public void pollLock() throws ItronCauseException;
  public void lock(int tmout) throws ItronCauseException;
  public void unlock() throws ItronCauseException;
  public T_RMTX refer() throws JtronException;
}
```

(45) ミューテックス生成情報クラス

```
public class T_CMTX {
  public T_CMTX();
  public T_CMTX(int mtxatr, int ceilpri);
        public int mtxatr;
        public int ceilpri;
        public static final int
                TA_TFIFO   = 0x00,
                TA_TPRI    = 0x01,
                TA_INHERIT = 0x02,
                TA_CEILING = 0x03;
 // 下層のμITRON4.0 仕様 OS に合わせた実装独自のフィールド
}
```

(46) ミューテックス状態クラス

```
public class T_RMTX {
        public int htskid;
        public int wtskid;
 // 下層のμITRON4.0 仕様 OS に合わせた実装独自のフィールド
}
```

(47) メッセージバッファアタッチクラス

```
public class MessageBuffer {
  public MessageBuffer(int mbfid) throws JtronException;
  public MessageBuffer(int mbfid, T_CMBF pk_cmbf)
                throws ItronCauseException;
  public MessageBuffer(T_CMBF pk_cmbf)
                throws ItronCauseException;
  public int getId();
  public void delete() throws ItronCauseException;
  public void send(ItronMemory msg) throws ItronCauseException;
  public void send(ItronMemory data, int off, int len)
                throws ItronCauseException;
  public void pollSend(ItronMemory msg)
                throws ItronCauseException;
  public void pollSend(ItronMemory data, int off, int len)
                throws ItronCauseException;
  public void send(ItronMemory msg, int tmout)
                throws ItronCauseException;
  public void send(ItronMemory data, int off, int len, int tmout)
                throws ItronCauseException;
  public int receive(ItronMemory msg)
                throws ItronCauseException;
  public int pollReceive(ItronMemory msg)
                throws ItronCauseException;
  public int receive(ItronMemory msg, int tmout)
                throws ItronCauseException;
  public T_RMBF refer() throws JtronException;
}
```

(48) **メッセージバッファ生成情報クラス**

```
public class T_CMBF {
  public T_CMBF(int maxmsz, int mbfsz);
  public T_CMBF(int mbfatr, int maxmsz, int mbfsz);
        public int mbfatr;
        public int maxmsz;
        public int mbfsz;
        public static final int
                TA_TFIFO = 0x00,
                TA_TPRI  = 0x01;
```

```
  public static final int TSZ_MBF(int msgcnt, int msgsz);
 // 下層のμITRON4.0 仕様 OS に合わせた実装独自のフィールド
}
```

(49) メッセージバッファ状態クラス

```
public class T_RMBF {
        public int stskid;
        public int rtskid;
        public int smsgcnt;
        public int fmbfsz;

 // 下層のμITRON4.0 仕様 OS に合わせた実装独自のフィールド
}
```

(50) ランデブポートアタッチクラス

```
public class RendezvousPort {
  public RendezvousPort(int porid) throws JtronException;
  public RendezvousPort(int porid, T_CPOR pk_cpor)
                throws ItronCauseException;
  public RendezvousPort(T_CPOR pk_cpor)
                throws ItronCauseException;
  public int getId();
  public void delete() throws ItronCauseException;
  public int call(int calptn, ItronMemory msg, int cmsgsz)
                throws ItronCauseException;
  public int call(int calptn, ItronMemory msg, int cmsgsz,
                int tmout) throws ItronCauseException;
  public int accept(int acpptn, Rendezvous msg)
                throws ItronCauseException;
  public int pollAccept(int acpptn, Rendezvous msg)
                throws ItronCauseException;
  public int accept(int acpptn, Rendezvous msg, int tmout)
                throws ItronCauseException;
  public T_RPOR refer() throws JtronException;
}
```

(51) ランデブポート生成情報クラス

```
public class T_CPOR {
  public T_CPOR(int maxcmsz, int maxrmsz);
  public T_CPOR(int poratr, int maxcmsz, int maxrmsz);
```

```
        public int poratr;
        public int maxcmsz;
        public int maxrmsz;
        public static final int
                TA_TFIFO = 0x00,
                TA_TPRI  = 0x01;
 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```

(52) ランデブポート状態クラス

```
public class T_RPOR {
        public int ctskid;
        public int atskid;
 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```

(53) ランデブアタッチクラス

```
public class Rendezvous extends ItronMemory {
  public Rendezvous(int length) throws JtronException;
  public int getNo() throws JtronCauseException;
  public void forwardTo(int porid, int calptn)
                throws JtronException;
  public void forwardTo(int porid, int calptn,
        ItronMemory msg, int cmsgsz) throws JtronException;
  public void reply(int rmsgsz) throws JtronException;
  public void reply(ItronMemory msg, int rmsgsz)
                throws JtronException;
  public T_RRDV refer() throws JtronException;
}
```

(54) ランデブ状態クラス

```
public class T_RRDV {
        public int wtskid;
 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```

(55) 固定長メモリプールアタッチクラス

```
public class FixedMemoryPool {
  public FixedMemoryPool(int mpfid, int blksz)
                throws JtronException;
  public FixedMemoryPool(int mpfid, T_CMPF pk_cmpf)
```

```
                throws ItronCauseException;
  public FixedMemoryPool(T_CMPF pk_cmpf)
                throws ItronCauseException;
  public int getId();
  public void delete() throws ItronCauseException;
  public ItronFixedMemory get() throws JtronException;
  public ItronFixedMemory poll() throws JtronException;
  public ItronFixedMemory get(int timeout)
                throws JtronException;
  public T_RMPF refer() throws JtronException;
}
```

(56) 固定長メモリプール生成情報クラス

```
public class T_CMPF {
  public T_CMPF(int blkcnt, int blksz);
  public T_CMPF(int mpfatr, int blkcnt, int blksz);
        public int mpfatr;
        public int blkcnt;
        public int blksz;
        public static final int
                TA_TFIFO = 0x00,
                TA_TPRI  = 0x01;
  public static final int TSZ_MPF(int blkcnt, int blksz);
 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```

(57) 固定長メモリプール状態クラス

```
public class T_RMPF {
        public int wtskid;
        public int fblkcnt;
 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```

(58) 固定長メモリブロックアタッチクラス

```
public class ItronFixedMemory extends ItronMemory {
  public int getPoolId();
  public void release() throws ItronCauseException;
}
```

(59) 可変長メモリプールアタッチクラス

```
public class VariableMemoryPool {
```

```
  public VariableMemoryPool(int mplid) throws JtronException;
  public VariableMemoryPool(int mplid, T_CMPL pk_cmpl)
                throws ItronCauseException;
  public VariableMemoryPool(T_CMPL pk_cmpl)
                throws ItronCauseException;
  public int getId();
  public void delete() throws ItronCauseException;
  public ItronVariableMemory get(int blksz)
                throws JtronException;
  public ItronVariableMemory poll(int blksz)
                throws JtronException;
  public ItronVariableMemory get(int blksz, int tmout)
                throws JtronException;
  public T_RMPL refer() throws JtronException;
}
```

(60) **可変長メモリプール生成情報クラス**

```
public class T_CMPL {
  public T_CMPL(int mplsz);
  public T_CMPL(int mplatr, int mplsz);
        public int mplatr;
        public int mplsz;
        public static final int
                TA_TFIFO = 0x00,
                TA_TPRI  = 0x01;
  public static final int TSZ_MPL(int blkcnt, int blksz);
 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```

(61) **可変長メモリプール状態クラス**

```
public class T_RMPL {
        public int wtskid;
        public int fmplsz;
        public int fblksz;
 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```

(62) **可変長メモリブロックアタッチクラス**

```
public class ItronVariableMemory extends ItronMemory {
  public int getPoolId();
```

```
  public void release() throws ItronCauseException;
}
```

(63) **周期ハンドラアタッチクラス**

```
public class CyclicHandler {
  public CyclicHandler(int cycid) throws JtronException;
  public CyclicHandler(int cycid, T_CCYC pk_ccyc)
                throws ItronCauseException;
  public CyclicHandler(T_CCYC pk_ccyc)
                throws ItronCauseException;
  public int getId();
  public void delete() throws ItronCauseException;
  public void start() throws ItronCauseException;
  public void stop() throws ItronCauseException;
  public T_RCYC refer() throws JtronException;
}
```

(64) **周期ハンドラ生成情報クラス**

```
public class T_CCYC {
  public T_CCYC(String cychdr, int cyctim);
  public T_CCYC(int cycatr, int exinf,
                 String cychdr, int cyctim, int cycphs);
        public int cycatr;
        public int exinf;
        public String cychdr;
        public int cyctim;
        public int cycphs;
        public static final int
                TA_HLNG = 0x00,
                TA_ASM  = 0x01;
        public static final int
                TA_STA  = 0x02,
                TA_PHS  = 0x04;
 // 下層のμITRON4.0 仕様 OS に合わせた実装独自のフィールド
}
```

(65) **周期ハンドラ状態クラス**

```
public class T_RCYC {
        public int cycstat;
        public int lefttim;
```

```
        public static final int
                TCYC_STP = 0x00,
                TCYC_STA = 0x01;
 // 下層のμITRON4.0 仕様 OS に合わせた実装独自のフィールド
}
```

(66) **アラームハンドラアタッチクラス**

```
public class AlarmHandler {
  public AlarmHandler(int almid) throws JtronException;
  public AlarmHandler(int almid, T_CALM pk_calm)
                throws ItronCauseException;
  public AlarmHandler(T_CALM pk_calm)
                throws ItronCauseException;
  public int getId();
  public void delete() throws ItronCauseException;
  public void start(int almtim) throws ItronCauseException;
  public void stop() throws ItronCauseException;
  public T_RALM refer() throws JtronException;
}
```

(67) **アラームハンドラ生成情報クラス**

```
public class T_CALM {
  public T_CALM(String almhdr);
  public T_CALM(int almatr, int exinf, String almhdr);
        public int almatr;
        public int exinf;
        public String almhdr;
        public static final int
                TA_HLNG = 0x00,
                TA_ASM  = 0x01;
 // 下層のμITRON4.0 仕様 OS に合わせた実装独自のフィールド
}
```

(68) **アラームハンドラ状態クラス**

```
public class T_RALM {
        public int almstat;
        public int lefttim;
        public static final int
                TALM_STP = 0x00,
                TALM_STA = 0x01;
```

```
 // 下層のμITRON4.0 仕様 OS に合わせた実装独自のフィールド
}
```

(69) 割込みサービスルーチンアタッチクラス

```
public class InterruptServiceRoutine {
  public InterruptServiceRoutine(int isrid)
                throws JtronException;
  public InterruptServiceRoutine(int isrid, T_CISR pk_cisr)
                throws ItronCauseException;
  public InterruptServiceRoutine(T_CISR pk_cisr)
                throws ItronCauseException;
  public int getId();
  public void delete() throws ItronCauseException;
  public T_RISR refer() throws JtronException;
}
```

(70) 割込みサービスルーチン生成情報クラス

```
public class T_CISR {
  public T_CISR(int intno, String isr);
  public T_CISR(int isratr, int exinf, int intno, String isr);
        public int isratr;
        public int exinf;
        public int intno;
        public String isr;
        public static final int
                TA_HLNG = 0x00,
                TA_ASM  = 0x01;
 // 下層のμITRON4.0 仕様 OS に合わせた実装独自のフィールド
}
```

(71) 割込みサービスルーチン状態クラス

```
public class T_RISR {
 // 下層のμITRON4.0 仕様 OS に合わせた実装独自のフィールド
}
```

(72) カーネルアタッチクラス

```
public class Kernel {
  public static void setTime(int systim)
                throws ItronCauseException;
  public static int getTime();
  public static void defineOverrunHandler(T_DOVR pk_dovr)
```

```
                throws ItronCauseException;
  public static void rotateReadyQueue(int tskpri)
                throws ItronCauseException;
  public static int getRunningTaskID();
  public static T_RSYS referSystem() throws JtronException;
  public static void defineInterruptHandler(
        int inhno, T_DINH pk_dinh) throws ItronCauseException;
  public static void disableInterrupt(int intno)
                throws ItronCauseException;
  public static void enableInterrupt(int intno)
                throws ItronCauseException;
  public static void defineServiceCall(int fncd, T_DSVC pk_dsvc)
                throws ItronCauseException;
  public static int callServiceCall(int fncd, int[] parm)
                throws ItronCauseException;
  public static void defineCPUExceptionHandler
                (int excno, T_DEXC pk_dexc)
                        throws ItronCauseException;
  public static T_RCFG referConfiguration()
                throws JtronException;
  public static T_RVER referVersion() throws JtronException;
        public static final int
                TSK_SELF    = 0,
                TSK_NONE    = 0,
                TPRI_SELF   = 0,
                TPRI_INI    = 0;
        public static final int
                TMO_POL     =  0,
                TMO_FEVR    = -1,
                TMO_NBLK    = -2;
        public static final int
                TMIN_TPRI   = 1,
                TMAX_TPRI,
                TMIN_MPRI   = 1,
                TMAX_MPRI,
                TKERNEL_MAKER,
                TKERNEL_PRID,
```

```
                TKERNEL_SPVER,
                TKERNEL_PRVER,
                TMAX_ACTCNT,
                TMAX_WUPCNT,
                TMAX_SUSCNT,
                TBIT_TEXPTN,
                TBIT_FLGPTN,
                TBIT_RDVPTN,
                TIC_NUME,
                TIC_DENO,
                TMAX_MAXSEM;
}
```

(73) **オーバーランハンドラ定義情報クラス**

```
public class T_DOVR {
  public T_DOVR(String ovrhdr);
  public T_DOVR(int ovratr, String ovrhdr);
        public int ovratr;
        public String ovrhdr;
        public static final int
                TA_HLNG = 0x00,
                TA_ASM  = 0x01;
 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```

(74) **システム状態クラス**

```
public class T_RSYS {
 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```

(75) **割込みハンドラ定義情報クラス**

```
public class T_DINH {
  public T_DINH(int inhatr, String inhhdr);
        public int inhatr;
        public String inhhdr;
 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```

(76) **拡張サービスコール定義情報クラス**

```
public class T_DSVC {
  public T_DSVC(String svcrtn);
```

```
  public T_DSVC(int svcatr, String svcrtn);
        public int svcatr;
        public String svcrtn;
        public static final int
                TA_HLNG = 0x00,
                TA_ASM  = 0x01;
 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```

(77) CPU 例外ハンドラ定義情報クラス

```
public class T_DEXC {
  public T_DEXC(int excno, String exchdr);
        public int excno;
        public String exchdr;
 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```

(78) コンフィグレーション情報クラス

```
public class T_RCFG {
 // 下層のμITRON4.0仕様OSに合わせた実装独自のフィールド
}
```

(79) バージョン番号クラス

```
public class T_RVER {
        public int maker;
        public int prid;
        public int spver;
        public int prver;
        public int prno0,
                   prno1,
                   prno2,
                   prno3;
}
```

7.2.3 タイプ 2 インタフェース

(1) 共有オブジェクトインタフェース

```
public interface Sharable {
  public abstract void lock()
                throws SharedObjectIllegalStateException;
  public abstract void lock(int timeout)
                throws SharedObjectIllegalStateException,
                        SharedObjectTimeoutException;
  public abstract void unlock()
                throws SharedObjectIllegalStateException;
  public abstract void forceUnlock()
                throws SharedObjectIllegalStateException;
  public abstract void unshare()
                throws SharedObjectIllegalStateException;
  public abstract void unshare(int timeout)
                throws SharedObjectIllegalStateException,
                        SharedObjectTimeoutException;
  public abstract Object getContent();
}
```

(2) 共有オブジェクトクラス

```
public class SharedObject implements Sharable {
        protected Sharable shm;

  public SharedObject(String name)
                throws  SharedObjectIllegalStateException;
  public SharedObject(Sharable shm, String name)
                throws SharedObjectIllegalStateException;
  public void lock() throws SharedObjectIllegalStateException;
  public void lock(int timeout)
                throws SharedObjectIllegalStateException,
                        SharedObjectTimeoutException;
  public void unlock()
                throws SharedObjectIllegalStateException;
  public  void forceUnlock()
                throws SharedObjectIllegalStateException;
  public void unshare()
```

```
                throws SharedObjectIllegalStateException;
  public void unshare(int timeout)
                throws SharedObjectIllegalStateException,
                        SharedObjectTimeoutException;
  public Object getContent()
}
```

(3) **共有オブジェクト関連の例外クラス**

```
public class SharedObjectException extends java.lang.Exception
{
  public SharedObjectException();
  public SharedObjectException(String msg);
}
```

(4) **メソッド発行状態に関する例外クラス**

```
public class SharedObjectIllegalStateException
                         extends SharedObjectException {
        public static final int ILLEGAL_MANAGER = 1;
        public static final int OBJECT_IN_USE = 2;
        public static final int OBJECT_NOEXIST = 3;
        public static final int ILLEGAL_NAME = 4;
        public static final int OBJECT_UNSHARED = 5;
        public static final int OBJECT_LOCKED = 6;
  public SharedObjectIllegalStateException(int cause);
  public SharedObjectIllegalStateException(int cause,
                                           String msg)
  public int getCause()
}
```

(5) **タイムアウトに関する例外クラス**

```
public class SharedObjectTimeoutException
                         extends SharedObjectException {
  public SharedObjectTimeoutException();
  public SharedObjectTimeoutException(String msg);
}
```

7.2.4 タイプ 3 インタフェース

(1) ストリームクラス

```
public class JtronStream extends java.lang.Object {
        public static final int MAIN_STREAM = 1;

  public JtronStream(int stmid)
                throws JtronStreamIllegalStateException;
  public JtronStream(int stmid, int timeout)
                throws JtronStreamIllegalStateException,
                        JtronStreamTimeoutException;
  public synchronized InputStream getInputStream()
                throws JtronStreamIllegalStateException;
  public synchronized OutputStream getOutputStream()
                throws JtronStreamIllegalStateException;
  public synchronized void setTimeout(int timeout)
                throws JtronStreamIllegalStateException;
  public synchronized int getTimeout()
                throws JtronStreamIllegalStateException;
  public synchronized void close()
                throws JtronStreamIllegalStateException;
}
```

(2) ストリーム関連の例外クラス

```
public class JtronStreamException
                        extends org.jtron.JtronException {
  public JtronStreamException();
  public JtronStreamException(String msg);
}
```

(3) メソッド発行状態に関する例外クラス

```
public class JtronStreamIllegalStateException
                        extends JtronStreamException{
        public static final int ILLEGAL_STREAM = 1;
        public static final int STREAM_IN_USE = 2;
        public static final int STREAM_NOEXIST = 3;
        public static final int ILLEGAL_ARGUMENT = 4;
        public static final int STREAM_CLOSED = 5;
  public JtronStreamIllegalStateException(int cause);
```

```
  public JtronStreamIllegalStateException(int cause,
                                          String msg);
  public int getCause();
}
```

(4) **タイムアウトに関する例外クラス**

```
public class JtronStreamTimeoutException
                        extends JtronStreamException {
  public JtronStreamTimeoutException();
  public JtronStreamTimeoutException(String msg);
}
```

付録

1 仕様と仕様書の利用条件

JTRON2.1 仕様および仕様書の利用条件は次の通りである。

仕様の利用条件

JTRON2.1 仕様は、オープンな仕様である。誰でも自由に、JTRON2.1 仕様に準拠したソフトウェアを開発・使用・配布・販売することができる。トロン協会へのライセンス料の支払いや届け出は必要ない。

ただし、トロン協会は、JTRON2.1 仕様に準拠したソフトウェアの製品マニュアルなどに、以下の文言(ないしは同じ趣旨の文言)を入れることを強く推奨する。

・TRON は“ The Real-time Operating system Nucleus ”の略称です。
・JTRON は、“ Java Technology on ITRON ”の略称です。
・ITRON は“ Industrial TRON ”の略称です。
・μITRON は“ Micro Industrial TRON ”の略称です。
・TRON，JTRON，ITRON，およびμITRON は、特定の商品ないしは商品群を指す名称ではありません。

また、JTRON2.1 仕様に準拠したソフトウェアの製品マニュアルなどに、以下の文言(ないしは同じ趣旨の文言)を入れることを推奨する。

JTRON2.1 仕様は、トロン協会 ITRON 部会 JTRON 仕様 WG が中心となって策定されたオープンな仕様です。JTRON2.1 仕様の仕様書は、TRON プロジェクトホームページ（http://www.tron.org/）から入手することができます。

仕様書を改変して製品マニュアルを作成する許諾を受けた場合（後述）や、JTRON 仕様準拠品登録制度に登録する場合(2 節参照)には、これらの 2 つの推奨に従うことが条件となる。

仕様書の利用条件

2 仕様の保守体制と問い合わせ先、参考情報

JTRON 仕様の保守体制と問い合わせ先

JTRON 仕様および仕様書の作成・保守は、トロン協会が行なっている。仕様ならびに仕様書に関する問い合わせ先は、次の通りである。

(社)トロン協会
〒 108-0073 東京都港区三田 1 丁目 3 番 39 号 勝田ビル 5 階
TEL: 03-3454-3191 FAX: 03-3454-3224

TRON プロジェクトホームページ
トロンプロジェクトに関する各種情報は以下の URL を参照されたい。

http://www.tron.org/

JTRON 仕様準拠製品登録制度

トロン協会では、JTRON 仕様の普及と発展を促進するため、JTRON 仕様準拠製品登録制度を設ける予定である。この制度は、各社で開発された JTRON 仕様準拠の製品の一覧を作成・保守し、トロン協会の広報活動などを通じて、JTRON 仕様ならびに準拠製品の普及を図ることを目的としたものである。なお、この制度は、いわゆる検定制度どは異なり、登録された製品が JTRON 仕様に準拠していることを認証するためのものではない。

仕様のバージョン番号

JTRON の仕様バージョン番号は、μITRON のバージョン番号の表記に準じる。以下に、”μITRON4.0 仕様(Ver.4.00.00) 5.3 仕様のバージョン番号”の抜粋を示す。

ITRON 仕様のバージョン番号は、次の形式とする。

Ver.*X*.*YY*.*ZZ*[.*WW*]

この内、*X* は ITRON 仕様のメジャーなバージョンを示す。カーネル仕様の場合の割当ては次の通りである。

1	ITRON1
2	ITRON2 またはμITRON(Ver.2.0)
3	μITRON3.0
4	μITRON4.0

YY は、仕様の内容に関する変更や追加を伴うようなバージョンの区別を示す番号である。仕様の公開は、仕様のバージョンアップにあわせて、YY=00, 01, 02, ...の順に増加させる。一方、検討中の仕様や暫定仕様(ドラフト)の場合は、*YY* にいずれかの桁を‘A’,‘B’,‘C’のいずれかとする。

バージョン番号の内、*X.YY* の部分は、カーネル構成マクロ TKERNEL_SPVER と ref_ver サービスコールのリターンパラメータ spver として参照することができる。*YY* が‘A’,‘B’,‘C’の場合は、16 進数の A’,‘B’,‘C’を用いる。

ZZ は、仕様書の表記に関するバージョンの区別を示す番号である。仕様書の構成や章立ての変更、ミスプリントの修正などがあった場合に、

ZZ=00, 01, 02, ...の順に増加させる。
仕様書の表記に関してさらにマイナーな区別を行いたい場合には、*WW* の桁を設ける場合がある。*WW* が省略されている場合には、*WW* が 00 であるものとみなす。

3 仕様検討関連者リスト

トロン協会 ITRON 部会 JTRON 仕様 WG メンバリスト(あいうえお順)

	朝倉　義晴	日本電気(株)
	石田　克彦	(株)日立製作所
	加藤　雅也	(株)東芝
	工藤　健治	富士通デバイス(株)
	小林　康浩	富士通(株)
	齋藤　栄治	富士通デバイス(株)
	鈴木　浩之	(株)アクセス
	高田　広章	豊橋技術科学大学
	高梨　修二	(株)東芝
	竹内　透	(社)トロン協会
	田中　勝	(株)エーアイコーポレーション
	丹崎　剛介	(株)アプリックス
幹事：	中本　幸一	日本電気(株)
	成田　武司	東芝情報システム(株)
	縄手　雅徳	東芝情報システム(株)
幹事：	八谷　祥一	(株)アプリックス
	福井　徹	(株)デンソークリエイト
	宮田　尚志	(株)東芝
	若林　隆行	豊橋技術科学大学

索引

メソッド

用語