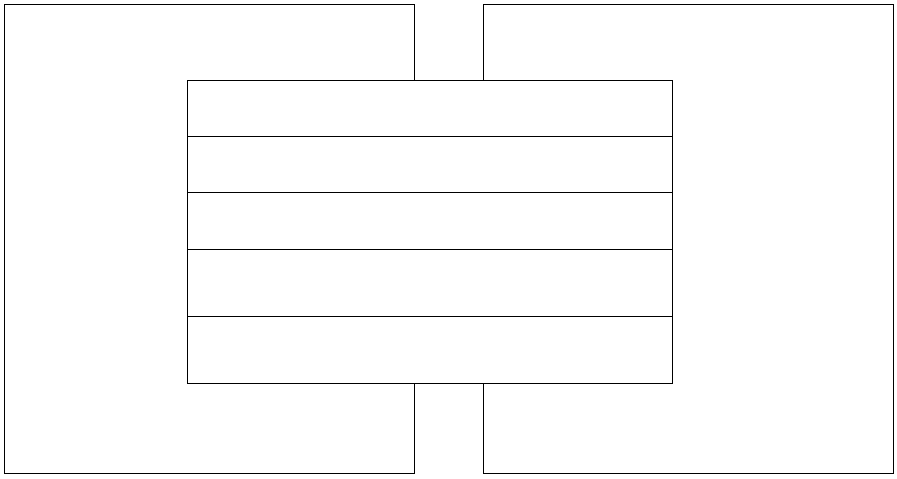
社団法人トロン協会

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

Editor: Yukikazu Nakamoto
Assistant Editor: Kazutoshi Usui
Page Layout Design: Kazutoshi Usui
Special thanks to
Natsuko Noda
Yoshiharu Asakura

はしがき

Java の有力な応用領域として、組込みシステムがある。組込みシステムでは従来からリアルタイム OS が利用されてきている。特に日本では ITRON が標準化され、多くの組込みシステムで使用されてきている。 Java を組込みシステムで使用する場合、リアルタイム処理をリアルタイムタスク、一般処理を Java プログラムという機能分割を行い、リアルタイムタスクと Java プログラムを協調動作させ実行処理を行う方法が有力である。この場合、システムを開発する視点からは、リアルタイムタスクや Java プログラムの移植性、再利用性、ひいてはプログラムの流通を促進するためにリアルタイムタスクと Java プログラムの間のインタフェースの標準化が必要である。本仕様書ではこれを規定している。

ITRON 専門委員会
Java Technology on ITRON-specification OS 技術委員会
1998 年 9 月 25 日

注意

- 仕様に関しては、下記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 Techonogy on ITRON-specification OS 技術委員会

天野亘孝 (元トロン協会)

石田 克彦 (日立製作所)

植田省司 (メトロワークス)

臼井和敏 (NEC)

大江哲男 (沖電気工業)

岡崎健二 (メンター・グラフィックス・ジャパン)

加藤雅也 (東芝)

鎌田富久 (アクセス)

亀井達也 (三菱電機)

工藤健治 (富士通デバイス)

小林康浩 (富士通)

柴下哲 (メンター・グラフィックス・ジャパン)

鈴木浩之 (アクセス)

是津達也 (元東京大学)

高田広章 (豊橋技術科学大学)

高梨修二 (東芝)

竹内透 (トロン協会)

多田幸生 (ヤマハ)

田中憲昭 (デンソークリエイト)

田丸喜一郎 (東芝)

中村憲一 (日本シグナスソリューションズ)

中本幸一 (NEC: 幹事)

成田武司 (東芝情報システム)

八谷祥一 (アプリックス)

林田聖司 (東芝)

平山亮 (Hewlett-Packard Company)

宮内哲夫 (NEC マイコンテクノロジー)

村木宏行 (三菱電機セミコンダクタシステム)

村中高大 (三菱電機)

吉田明廣 (アプリックス)

渡辺洋幸 (セイコーインスツルメンツ)

目次

図目次

表目次

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 編集 · 発行, 「μ ITRON3.0 標準ハンドブック 改訂新版」, パーソナルメディア, 1997.

[3] JavaSoft, “Java Native Interface Specification Release 1.1”, May,1997.

[4] J.Gosling, B. Joy and G. Steele, “The Java Language Specification”, Addison-Wesley, 1996.

[5] ErichGamma 他著, 本位田真一他監訳, 「オブジェクト指向における再利用にためのデザインパターン」, ソフトバンク, 1995.

[6] 戸松豊和著, 「JAVA プログラムデザイン」, ソフトバンク, 1997.

第 1 章

総論

1.1 概要

Java の有力な応用領域として、組込みシステムがある。組込みシステムでは従来からリアルタイム OS が利用されてきている。特に日本では ITRON が標準化され、多くの組込みシステムで使用されてきている。 Java を組込みシステムで使用する場合、リアルタイム処理をリアルタイムタスク、一般処理を Java プログラムという機能分割を行い、リアルタイムタスクと Java プログラムを協調動作させ実行処理を行う方法が有力である。この場合、システムを開発する視点からは、リアルタイムタスクや Java プログラムの移植性、再利用性、ひいてはプログラムの流通を促進するためにリアルタイムタスクと Java プログラムの間のインタフェースの標準化が必要である。本仕様書ではこれを規定する。

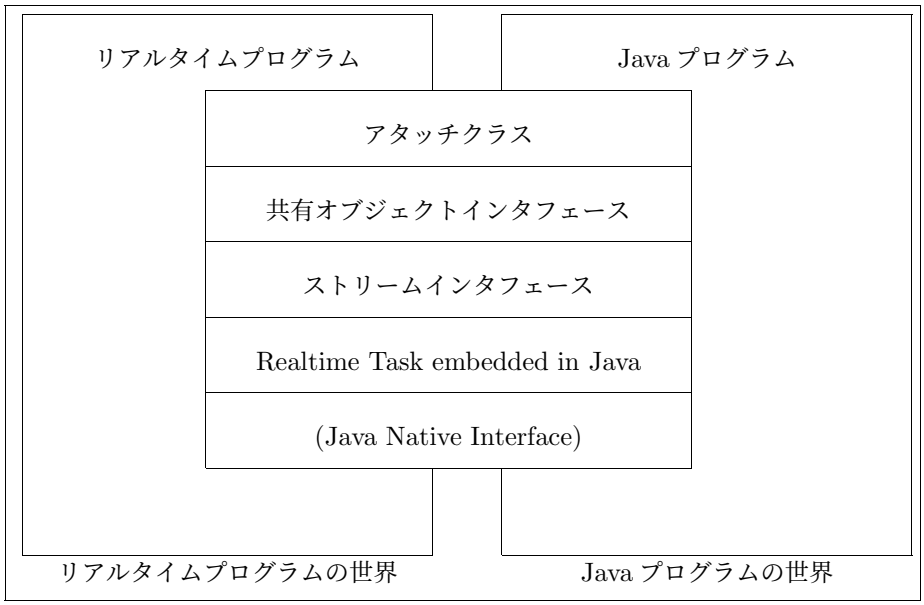


図 1.1: Java プログラムとリアルタイムプログラムの協調

リアルタイムタスクと Java プログラムの間のインタフェースには次の 2 つがある。

(1) Java スレッドとリアルタイムタスク間の関係の定義
Java スレッドはリアルタイムタスクに 1 対 1 にマッピングするものとする。このマッピングの方法を規定する。

(2) Java プログラムとリアルタイムタスクの協調動作の定義

(2)に関しては以下の形態が考えられる。

タイプ 1: アタッチクラス

Java プログラムで ITRON カーネルのシステムコールが利用できるようにする (JTRON1 仕様 [1] が対応)。

タイプ 2: 共有オブジェクトインタフェース
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。

タイプ 3: ストリームインタフェース
Java プログラムとリアルタイムタスクがストリームにより通信する。

タイプ 1 からタイプ 3 に行くにしたがって、 Java プログラムとリアルタイムタスクの関係は密結合から疎結合になる。

これらのほかに次のアプローチがあり利用可能である。

タイプ 4 リアルタイムタスクに Java のプログラミングを持ち込む。
これは、リアルタイムプログラムから JNI(Java Native method Interface[3]) を使って、 Java の API を呼び出すことで実現できる。

本仕様書で規定した内容は、 ITRON カーネルのみならず、一般のリアルタイム OS でも適用可能である。

1.2 全体規則 (ITRON カーネル)

1.2.1 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 Interace) 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

1.2.2 静的 API と動的 API

オブジェクトを生成する各 API に対して、システム構成ファイル中に記述することで、システム初期化時にオブジェクトを自動的に生成する静的な設定方法 (これを，静的 API と呼ぶ) を用意する。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。

1.2.3 API の戻り値とエラーコード

各 API の戻り値は、 ITRON 仕様のコンベンションに従い、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には 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、メインエラーコード、サブエラーコード、実装依存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、実装依存エラーコード共に負の値とし、それらを組み合わせたエラーコードも負の値とする。なお、 JTRON 仕様ではメインエラーコード、サブエラーコード共に 8 ビット、実装依存エラーコードは 0 ビット以上である。

メインエラーコード、サブエラーコード、実装依存エラーコードの実装を隠蔽する以下のマクロを提供する。

JTI_MAINERCD(ercd) メインエラーコード
JTI_SUBERCD(ercd) サブエラーコード
JTI_IMPLERCD(ercd) 実装依存エラーコード

注意: 上記のマクロ名は ITRON 全体で仕様が決まった場合は置換される。

メインエラーコードのニーモニックと値および意味は、 ITRON カーネル仕様のエラーコードと同じになるように標準化する。ただし、 ITRON カーネル仕様で足りないエラーコード (E_CLS) は追加定義する。

サブエラーコードは以下のように利用されることを期待する。すなわち、リアルタイムタスクから Java オブジェクトにアクセスする API 実行時に Java 側で例外が発生した場合に、 ITRON API のエラーコードのメインエラーコードは E_OBJ とし、サブエラーコードに Java の例外コードを割り当てる。

実装依存エラーコードについては実装により定義される。サブエラーコードはデバッグ時に有用な情報の返却などに利用されることを期待する。

この仕様書では、各 API が返すエラーコードとして、メインエラーコードのみを定義する。
以下のエラーコードは、 API 毎には記述しないが、すべての API が返す可能性がある。具体的にどの API がどのエラーを返す可能性があるかは実装依存である。

E_SYS	システムエラー
E_NOMEM	メモリ不足
E_NOSPT	未サポート機能
E_MACV	メモリアクセス違反

1.2.4 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 を呼び出したこと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元の状態に戻せない場合は除く。
ポーリング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、タイムアウトがない (永久待ちの) 場合の振舞いを説明している。 API の機能説明中で「待ち状態となる」とある場合でも、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。

1.2.5 API とタスクの関係

本仕様の API は、パラメタが同じであれば、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依存である。

1.3 共通定義

1.3.1 ヘッダファイル

ヘッダファイル名:"jti_XXX.h"とする。

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"

1.3.2 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、長さは実装依存
ER	整数型、 JTRON では 16 ビット以上

(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 */
（他に実装依存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 */
（他に実装依存のフィールドがあってもよい）
} T_JTI_RSTM;
```

1.3.3 定数

(1) 一般

NADR −1 無効アドレス

(2) API の機能コード

(略)

(3) メインエラーコード

E_OK	0	正常終了
E_SYS	−5	システムエラー
E_NOMEM	−10	メモリ不足
E_NOSPT	−17	未サポート機能
E_RSATR	−24	予約属性
E_PAR	−33	パラメタエラー
E_ID	−35	不正 ID 番号
E_NOEXS	−52	オブジェクト未生成
E_OBJ	−63	オブジェクト状態エラー
E_MACV	−65	メモリアクセス違反
E_DLT	−81	待ちオブジェクトの削除
E_RLWAI	−86	処理のキャンセル，待ち状態の強制解除
E_CLS	−87	接続の切断

(4) BOOL の値

TRUE	1	真
FALSE	0	偽

(5) タイムアウト指定

TMO_POL	0	ポーリング
TMO_FEVR	−1	永久待ち

(6) Java スレッド / リアルタイムタスク優先度対応指定

JTI_DFL_HPR JRE を実装するリアルタイムタスクのディフォルト最高優先度値、値は実装依存である。

(7) 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

(8) エラー取得マクロ

JTI_MAINERCD(ercd)	メインエラーコード
JTI_SUBERCD(ercd)	サブエラーコード
JTI_IMPLERCD(ercd)	実装依存エラーコード

1.4 全体規則 (Java)

1.4.1 JTRON 標準 Java パッケージ構成

JTRON2.0 仕様を提供する Java のクラスパッケージ名は同一仕様のものであればユニークにする。 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 1.1)。

表 1.1: JTRON 標準 Java パッケージ名

種別	パッケージ名の形式	JTRON 標準 Java パッケージ名
タイプ 1 で使用されるパッケージ:	XXX.jtron.attach.YYY	jp.gr.itron.jtron.attach.YYY
タイプ 2 で使用されるパッケージ:	XXX.jtron.shared.YYY	jp.gr.itron.jtron.shared.YYY
タイプ 3 で使用されるパッケージ:	XXX.jtron.stream.YYY	jp.gr.itron.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 Java 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。

1.4.2 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ーライドしなければならない。

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

1.4.3 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 :
提供する JTRON の仕様バージョン番号 (μ ITRON のバージョン番号の表記に準じる)。

jtron.type :
タイプ番号。以下の英数字の 1 文字以上の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: 将来の拡張のために予約

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、 jp.gr.itron.jtron.JtiSystem クラスの getProperty メソッドで取得できる。

1.5 運用規則

(1) メンテナンス方法
各社実装、評価が行われると思われる 1999 年上期あたりで仕様見直しを行う。また、その時点で ITRON4.0 仕様との整合性も図る。

(2) 準拠性について
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「JTRON2.0 に準拠している」と言ってよいものとする。

(3) 登録制度と検定
仕様に合致している製品であることを認証するために登録制度を運用する。なお、仕様に合致しているかどうか検定は行わない。

第 2 章

Java スレッドとリアルタイムタスクの対応

2.1 概要

JTRON2.0 では、 Java スレッドはリアルタイムタスクと 1 対 1 にマッピングする。以下のマッピングに関して規定する。

(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， JRE(Java Runtime Environment) 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できる。

【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表の設定、参照ができるものとする。
- JTRON2.0 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。

- リアルタイムタスク側は JTRON2.0 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。
- リアルタイムタスクで利用可能な優先度は静的に取得可能であるべきである。

2.2 ITRON API

API 名	概要	種別
jti_set_hpr	JRE 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 スレッドの優先度からリアルタイムタスク優先度を求める	標準仕様
jti_get_lpr	JRE 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

標準仕様

jti_set_hpr

JRE 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】

```
void jti_set_hpr(hijpr);
```

【静的 API】

```
JTI_SET_HPR(hijpr)
```

【パラメタ】

PRI *hijpr* リアルタイムタスクの優先度

【[戻り値】

なし

【API の機能】

JRE を実装するリアルタイムタスクの最高優先度の値を *hijpr* にする。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ている。

【注意】

動的に最高優先度の値を変更しても、既に動作している Java スレッドの優先度は変更されない。

標準仕様

jti_get_hpr

Java スレッドの優先度からリアルタイムタスク優先度を求める

【C 言語 API】

```
PRI pri = jti_get_hpr(hijpr, jpr);
```

【静的 API】

```
PRI pri = JTI_GET_HPR(hijpr, jpr);
```

【パラメタ】

PRI *hijpr* JRE 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

【戻り値】

PRI *pri* リアルタイムタスクの優先度

【API の機能】

Java スレッドを実装するリアルタイムタスクの最高優先度値 *hijpr* から、 Java スレッド *jpr* の ITRON での優先度を求め返却する。静的 API は提供するかどうかも含めて実装依存とする。

標準仕様

jti_get_lpr

JRE 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】

```
PRI pri = jti_get_lpr(hijpr);
```

【静的 API】

```
PRI pri = JTI_GET_HPR(hijpr);
```

【パラメタ】

PRI *hijpr* JRE を実装するリアルタイムタスクの最高優先度

【戻り値】

PRI *pri* JRE を実装するリアルタイムタスクの最低優先度

【API の機能】

Java スレッドを実装するリアルタイムタスクの最高優先度値 *hijpr* から、JRE を実装するリアルタイムタスクの最低優先度の値を求め返却する。静的 API は提供するかどうかも含めて実装依存とする。

2.3 Java API

2.3.1 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、jp.gr.itron.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。

クラス: JtiSystem

2.3.2 クラス jp.gr.itron.jtron.JtiSystem

```
java.lang.Object
   |
   +--- jp.gr.itron.jtron.JtiSystem
```

---

public JtiSystem

JTRON に関するプロパティなどの情報などを管理する。

■ コンストラクタ

protected JtiSystem()

■ メソッド

public static JtiSystem getJtiSystem()
JtiSystem クラスのオブジェクトを得る。このメソッドを呼び出すことにより Java 側の JTRON の機構 (ITRON 側から Java のリソースを制御する機構など) が利用可能になることが保証される。

public String getProperty(String key)
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 String getProperty(String key, String default)
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、 default を返す。

public Properties getProperties()
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 :
提供する JTRON の仕様バージョン番号 (µ ITRON のバージョン番号の表記に準じる)。

jtron.type :
タイプ番号。以下の英数字の 1 文字以上の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: 将来の拡張のために予約

jtron.vendor :
ベンダ名 (ベンダで自由に定めてよい)

第 3 章

アタッチクラス

文献 [1] の JTRON 仕様を本仕様では JTRON1 仕様と呼ぶ。

アタッチクラスに関してはリアルタイム OS の仕様がμ ITRON3.0[2] までは JTRON1 仕様をそのまま参照する。μ ITRON4.0 に対応するアタッチクラスの仕様は、 μ ITRON4.0 仕様作成と並行して仕様化する計画である。

第 4 章

共有オブジェクトインタフェース

4.1 概要

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 4.1)。

Java プログラムでは共有対象のオブジェクトをリアルタイムタスクのやりとりを司る共有オブジェクトマネジャーに登録する。リアルタイムタスクでは登録された共有オブジェクトの先頭アドレスを取得する。この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。

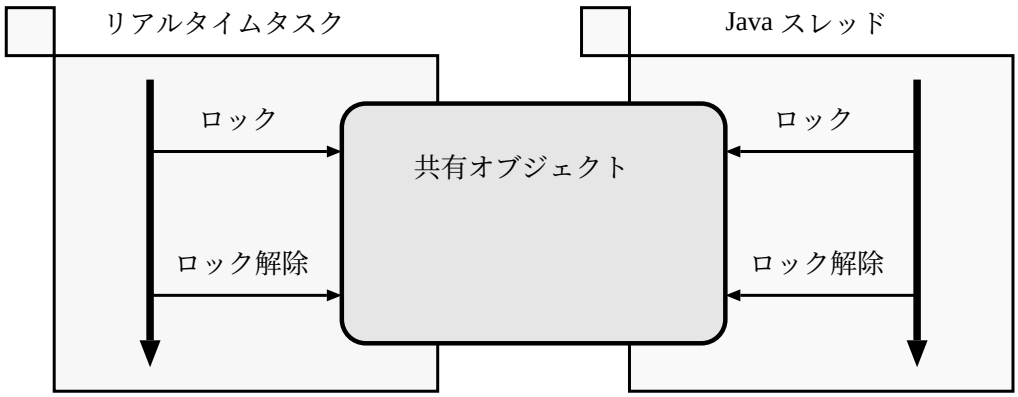


図 4.1: 共有オブジェクト

リアルタイムタスク側とやりとりするクラスとして **SharedObjectManager** クラスを設け、他のクラスはリアルタイムタスク側とのやりとりはすべて **SharedOzbjectManager** を通して行うようにする。 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共有するオブジェクトを生成する。本共有オブジェクト生成時に 名前を付与して **SharedObjectManager** に登録される。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、 **loc_shm**、 **unl_shm** 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後、その先頭アドレスを取り出してアクセスし、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。ロック解除後の共有オブジェクトのアドレスは保証されない。共有が終了しているかもしれないからである。

Java の **Thread** クラス、 **ThreadGroup** クラスのうち状態遷移を伴う実行制御メソッドをリアルタイムタスクから拡張仕様として呼び出すこともできる。共有オブジェクトで通信する場合に、リアルタイムタスクから Java のスレッドを制御することが必要となるからである。

ロックのセマンティクス

共有オブジェクトに対して、共有状態、未共有状態、共有状態はさらに分割して、ロックした状態、ロックのない状態で、 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し、「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が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表 4.1示す。

表 4.1: 共有オブジェクトのロックの状態遷移

操作		共有				未共有
		ロックなし	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 メソッド	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 *1	OK (no effect)	OK (ロック解除)	OK (ロック解除)	OK (no effect)	例外発生
	unshare *2	OK	*3	*4	*4	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 *5	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** **forceUnlock** メソッドにより任意の Java スレッドのロックを強制解除する。リアルタイムタスクのロックを解除しない。このメソッドは共有データをロックした Java スレッドが死んだ場合に、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。

***2** **unshare** メソッドにより共有オブジェクトの共有を終了する。共有の終了は Java 側からのみ可能である。 **unshare** メソッドでは、共有オブジェクトをロック後、オブジェクトの共有終了処理を行い、オブジェクトのロック解除が行われる。したがって、すでに、共有終了対象のオブジェクトが Java スレッド、リアルタイムタスクによりロックされている場合は、ロックが解除されるまで **unshare** メソッドを発行したスレッドはブロックされる。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。

***3** ロックを解除した後、オブジェクトの共有を終了する。

***4** 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。

***5** 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 (**jti_funl_obj**) しようとする場合は、ロックが解除される。

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.

共有オブジェクトのロック時の待ち順序は実装依存である.

ガーベジコレクション (GC) との関係

(1) **share** メソッド発行後の共有オブジェクト (通常はコンストラクタ中で **share** メソッドが発行される) は、GC の対象にならない。

(2) **unshare** メソッド発行後の共有オブジェクトは、 GC に対象になる。 sharable なオブジェクトは自動的には消えない。明示的に **unshare** メソッドを実行しないと GC の対象にならないので注意が必要である。

【補足説明】

- **期待する操作順序**

  図 4.2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: ShareObject を名前を付けて登録
2: jti_get_obj で名前から共有オブジェクト識別番号を獲得	
3: jti_loc_obj で共有オブジェクトをロック	
4: jti_get_mem で共有オブジェクトをアドレスを獲得	
5: アドレスを使ってアクセス	
6: jti_unl_obj で共有オブジェクトをロックを解除	
7: Java スレッドに共有オブジェクトアクセスの終了を通知	
	8: SharedObject に対して **lock** メソッドを実行
	9: SharedObject に対して **unlock** メソッドを実行

図 4.2: 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**

  Java オブジェクト内のメモリ配置 (Endian，アライメント， パディング等) は javah のようなツールにより C の構造体の形で求めることができるものとする．ただし，この仮定は JNI 以外のインタフェースを生成する機能の存在が保証されていないので早晩見直した方がよいと思われる．

  jti_loc_obj と **jti_unl_obj** の実行する間に一般にどのようなシステム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 run/ready 以外の状態になるとロックを放すことができなくなる可能性がある．従って，ロック中のタスクは run/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，もしくは例外が発生する) はロック時に待ち時間 0 を指定することで実現できる．

4.2 ITRON API

4.2.1 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_mem	指定された共有オブジェクト (クラス名は **Sharable**) に対応するメモリ領域の先頭のポインタを返す	標準仕様
jti_loc_obj	指定された Java 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

標準仕様

jti_get_obj

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(char *objnm, JNO *p_objno);
```

【パラメタ】

char **objnm* 共有オブジェクトの名前
JNO **p_objno* 共有オブジェクトの識別番号

【戻り値】

ER *ercd* エラーコード

【エラーコード】

E_OK 正常終了
E_OBJ *obknm* に対応する共有オブジェクトが存在していない
E_PAR パラメタの誤り (*objnm* が NULL ポインタ)

【API の機能】

objnm に対応する Java の共有オブジェクトの識別番号を *p_objno* の指す領域に返す． *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* に対応する Java オブジェクトが存在しない場合は，**E_OBJ** を返す． *objnm* が NULL ポインタの場合は，**E_PAR** を返す．実装は objnm を ASCII 文字列に限定してもよい．

jti_get_mem

指定された共有オブジェクト (クラス名は Sharable) に対応するメモリ領域の先頭のポインタを返す

【C 言語 API】

```
ER ercd = jti_get_mem(JNO objno, VP* p_addr);
```

【パラメタ】

JNO *objno* 共有オブジェクトの識別番号
VP* *p_addr* 共有オブジェクトの先頭アドレスを格納する領域へのポインタ

【戻り値】

ER *ercd* エラーコード

【エラーコード】

E_OK 正常終了
E_PAR パラメタの誤り
E_OBJ オブジェクトが存在していない

【API の機能】

obj で指定された共有オブジェクトの先頭領域のポインタを *p_addr* の指す領域に返す。プログラマは、 *p_addr* の指す領域に格納されたアドレスに対して、 Java オブジェクトに対応する型定義で casting して、アクセスすることになる。 Java と C 言語との間の型の対応は JNI の仕様書 [3] を参照のこと。

標準仕様

jti_loc_obj

指定された Java 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

【戻り値】

ER	*ercd*	エラーコード

【エラーコード】

E_OK	正常終了
E_PAR	パラメタの誤り
E_OBJ	共有オブジェクトが存在していない
E_TMOUT	タイムアウトが発生した
E_RLWAI	待ち状態強制解除
E_DLT	共有が解除された

【API の機能】

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 **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** メソッドでのロック解除、もしくはリアルタイムタスクの **unl_obj** でのロックが解除される。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。

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

【C 言語 API】

```
ER ercd = jti_unl_obj(JNO objno);
```

【パラメタ】

JNO *objno* 共有オブジェクトの識別番号

【戻り値】

ER *ercd* エラーコード

【エラーコード】

E_OK 正常終了
E_PAR パラメタの誤り (*objno* が不正)
E_OBJ 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

【API の機能】

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。

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

【C 言語 API】

```
ER ercd = jti_funl_obj(JNO objno);
```

【パラメタ】

JNO *objno* 共有オブジェクトの識別番号

【戻り値】

ER *ercd* エラーコード

【エラーコード】

E_OK 正常終了
E_PAR パラメタの誤り (objno が不正)
E_OBJ オブジェクトが存在していない

【API の機能】

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。

4.2.2 Java スレッド操作のための ITRON API

以下の API の説明で Java の **Thread** クラスのメソッドに関しては [4] を参照.

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 isInterruputed 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_thr_stp	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 クラス中のメソッド **setPriority** 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

拡張仕様

jti_get_thr

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(char *thrnm, JNO *p_thrno);
```

【パラメタ】

char **thrnm* Java スレッドの名前
JNO **p_thrno* Java スレッドの識別番号

【戻り値】

ER *ercd* エラーコード

【エラーコード】

E_OK 正常終了
E_OBJ スレッドが存在していない
E_PAR パラメタの誤り (*thrnm* が NULL ポインタ)

【API の機能】

thrnm に対応する Java スレッドの識別番号を *p_thrno* が指す領域に返す。 *thrnm* 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。 *thrnm* に対応する Java オブジェクトが存在しない場合は、 **E_OBJ** を返す。 *thrnm* が NULL ポインタの場合は、 **E_PAR** を返す。実装は *thrnm* を ASCII 文字列に限定してもよい。

拡張仕様

jti_isa_thr

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);
```

【パラメタ】

JNO *thrno* Java スレッドの識別番号

【戻り値】

ER_BOOL *ercd* メソッドの返却値もしくはエラーコード

【エラーコード】

TRUE 真
FALSE 偽
E_PAR パラメタの誤り (*thrno* が不正)

【機能】

thrno で指定された Java スレッドに対して Thread クラス中の **isAlive** メソッドを呼び、その結果を返す。

拡張仕様

jti_int_thr

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);
```

【パラメタ】

JNO *thrno* Java スレッドの識別番号

【戻り値】

ER *ercd* エラーコード

【エラーコード】

E_OK 正常終了
E_PAR パラメタの誤り (*thrno* が不正)

【API の機能】

thrno で指定された Java スレッドに対して 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 isInterruputed メソッドを呼ぶ

【C 言語 API】

```
ER_BOOL ercd = jti_isi_thr(JNO thrno);
```

【パラメタ】

JNO *thrno* Java スレッドの識別番号

【戻り値】

ER_BOOL *ercd* メソッドの返却値もしくはエラーコード

【エラーコード】

TRUE 真
FALSE 偽
E_PAR パラメタの誤り (*thrno* が不正)

【API の機能】

thrno で指定された Java スレッドに対して Thread クラス中の **isInterrupted** メソッドを呼び、その結果を返す。

拡張仕様

jti_sus_thr

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);
```

【パラメタ】

JNO *thrno* Java スレッドの識別番号

【戻り値】

ER *ercd* エラーコード

【エラーコード】

E_OK 正常終了
E_PAR パラメタの誤り (*thrno* が不正)
E_OBJ Java メソッド実行中にセキュリティ例外が発生した

【API の機能】

thrno で指定された Java スレッドに対して **Thread** クラス中の **suspend** メソッドを呼ぶ。どういう条件でセキュリティ例外が発生したかはセキュリティマネジャの実装に依存する。

拡張仕様

jti_rsm_thr

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);
```

【パラメタ】

JNO *thrno* Java スレッドの識別番号

【戻り値】

ER *ercd* エラーコード

【エラーコード】

E_OK 正常終了
E_PAR パラメタの誤り (thrno が不正)
E_OBJ Java メソッド実行中にセキュリティ例外が発生した

【API の機能】

thrno で指定された Java スレッドに対して Thread クラス中の **resume** メソッドを呼ぶ。どういう条件でセキュリティ例外が発生したかはセキュリティマネジャの実装に依存する。

拡張仕様

jti_sta_thr

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);
```

【パラメタ】

JNO *thrno* Java スレッドの識別番号

【戻り値】

ER *ercd* エラーコード

【エラーコード】

E_OK 正常終了
E_PAR パラメタの誤り (*thrno* が不正)
E_OBJ Java メソッド実行中に状態違反が発生した

【API の機能】

thrno で指定された Java スレッドに対して **Thread** クラス中の **start** メソッドを呼ぶ。

拡張仕様

jti_thr_stp

Java の Thread クラス中の stop メソッドを呼ぶ

【C 言語 API】

```
ER ercd = jti_thr_stp(JNO thrno);
```

【パラメタ】

JNO *thrno* Java スレッドの識別番号

【戻り値】

ER *ercd* エラーコード

【エラーコード】

E_OK 正常終了
E_PAR パラメタの誤り (*thrno* が不正)
E_OBJ Java メソッド実行中にセキュリティ例外、 NULL ポインタ例外が発生した

【API の機能】

thrno で指定された Java スレッドに対して **Thread** クラス中の **stop** メソッドを呼ぶ。どういう条件でセキュリティ例外が発生したかはセキュリティマネジャの実装に依存する。

【補足説明】

オーバライドメソッド **stop(Throwable thrno)** 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出し可能なメソッドから除外する。

拡張仕様

jti_get_jpr

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);
```

【パラメタ】

JNO *thrno* Java スレッドの識別番号
INT **p_rslt* Java スレッドの優先度

【戻り値】

ER *ercd* エラーコード

【エラーコード】

E_OK 正常終了
E_PAR パラメタの誤り (*thrno* が不正)

【API の機能】

thrno で指定された Java スレッドに対して **Thread** クラス中の **getPriority** メソッドを呼び、その結果を *p_rslt* に返す。

【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。

拡張仕様

jti_set_jpr

Java の Thread クラス中のメソッド setPriority を呼ぶ

【C 言語 API】

```
ER ercd = jti_set_jpr(JNO thrno, INT newpri);
```

【パラメタ】

JNO *thrno* Java スレッドの識別番号
INT *newpri* Java スレッドの優先度

【戻り値】

ER *ercd* エラーコード

【エラーコード】

E_OK 正常終了
E_PAR パラメタの誤り (*thrno* が不正)
E_OBJ Java メソッド実行中にセキュリティ例外、引数誤り例外が発生した

【API の機能】

thrno で指定された Java スレッドに対して **Thread** クラス中の **setPriority** メソッドを呼ぶ。どういう条件でセキュリティ例外が発生したかはセキュリティマネジャの実装に依存する。

【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。

拡張仕様

jti_des_thr

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);
```

【パラメタ】

JNO *thrno* Java スレッドの識別番号

【戻り値】

ER *ercd* エラーコード

【エラーコード】

E_OK 正常終了
E_PAR パラメタの誤り (*thrno* が不正)
E_OBJ Java メソッド実行中にセキュリティ例外が発生した

【API の機能】

thrno で指定された Java スレッドに対して **Thread** クラス中の **destory** メソッドを呼ぶ。どういう条件でセキュリティ例外が発生したかはセキュリティマネジャの実装に依存する。

4.2.3 Java スレッドグループ操作のための ITRON API

以下の API の説明で Java の ThreadGroup クラスのメソッドに関しては [4] を参照.

API 名	概要	種別
jti_get_tgr	名前から Java 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

拡張仕様

jti_get_tgr

名前から Java 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(char *tgrnm, JNO *p_tgrno);
```

【パラメタ】

char **tgrnm* Java スレッドグループの名前
JNO **p_tgrno* Java スレッドグループの識別番号

【戻り値】

ER *ercd* エラーコード

【エラーコード】

E_OK 正常終了
E_OBJ スレッドグループが存在していない
E_PAR パラメタの誤り (*tgrnm* が NULL)

【API の機能】

tgrnm に対応する Java スレッドグループの識別番号を *p_tgrno* の指す領域に返す。 *tgrnm* 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 のスレッドグループの識別番号を返す。 *thgnm* に対応する Java オブジェクトが存在しない場合は、 **E_OBJ** を返す。 *thgnm* が NULL ポインタの場合は、 **E_PAR** を返す。実装は *tgrnm* を ASCII 文字列に限定してもよい。

拡張仕様

jti_des_tgr

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);
```

【パラメタ】

JNO *tgrno* Java スレッドグループの識別番号

【戻り値】

ER *ercd* エラーコード

【エラーコード】

E_OK 正常終了
E_PAR パラメタの誤り (*tgrno* が不正)
E_OBJ Java メソッド実行中にセキュリティ例外、状態違反が発生した。

【API の機能】

tgrno で指定された Java スレッドグループに対して **ThreadGroup** クラス中の **destroy** メソッドを呼ぶ。どういう条件でセキュリティ例外が発生したかはセキュリティマネジャの実装に依存する。

拡張仕様

jti_sus_tgr

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);
```

【パラメタ】

JNO *tgrno* Java スレッドグループの識別番号

【戻り値】

ER *ercd* エラーコード

【エラーコード】

E_OK 正常終了
E_PAR パラメタの誤り (*tgrno* が不正)
E_OBJ Java メソッド実行中にセキュリティ例外が発生した

【API の機能】

tgrno で指定された Java スレッドグループに対して **ThreadGroup** クラス中の **suspend** メソッドを呼ぶ。どういう条件でセキュリティ例外が発生したかはセキュリティマネジャの実装に依存する。

拡張仕様

jti_rsm_tgr

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);
```

【パラメタ】

JNO *tgrno* Java スレッドグループの識別番号

【戻り値】

ER ercd
エラーコード

【エラーコード】

E_OK 正常終了
E_PAR パラメタの誤り (*tgrno* が不正)
E_OBJ Java メソッド実行中にセキュリティ例外が発生した

【API の機能】

tgrno で指定された Java スレッドグループに対して **ThreadGroup** クラス中の **resume** メソッドを呼ぶ。どういう条件でセキュリティ例外が発生したかはセキュリティマネジャの実装に依存する。

拡張仕様

jti_stp_tgr

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);
```

【パラメタ】

JNO *tgrno* Java スレッドグループの識別番号

【戻り値】

ER *ercd* エラーコード

【エラーコード】

E_OK 正常終了
E_PAR パラメタの誤り (*tgrno* が不正)
E_OBJ Java メソッド実行中にセキュリティ例外が発生した

【API の機能】

tgrno で指定された Java スレッドグループに対して **ThreadGroup** クラス中の **stop** メソッドを呼ぶ。どういう条件でセキュリティ例外が発生したかはセキュリティマネジャの実装に依存する。

4.3 Java API

4.3.1 パッケージ構成

共有オブジェクトを提供するクラスは、jp.gr.itron.jtron.shared パッケージにまとめられている。以下のインタフェース、クラス、例外クラスから構成される。

インタフェース: Sharable

クラス: SharedObject,SharedObjectManager

例外クラス: ShmException,ShmIllegalStateException,ShmTimeoutException

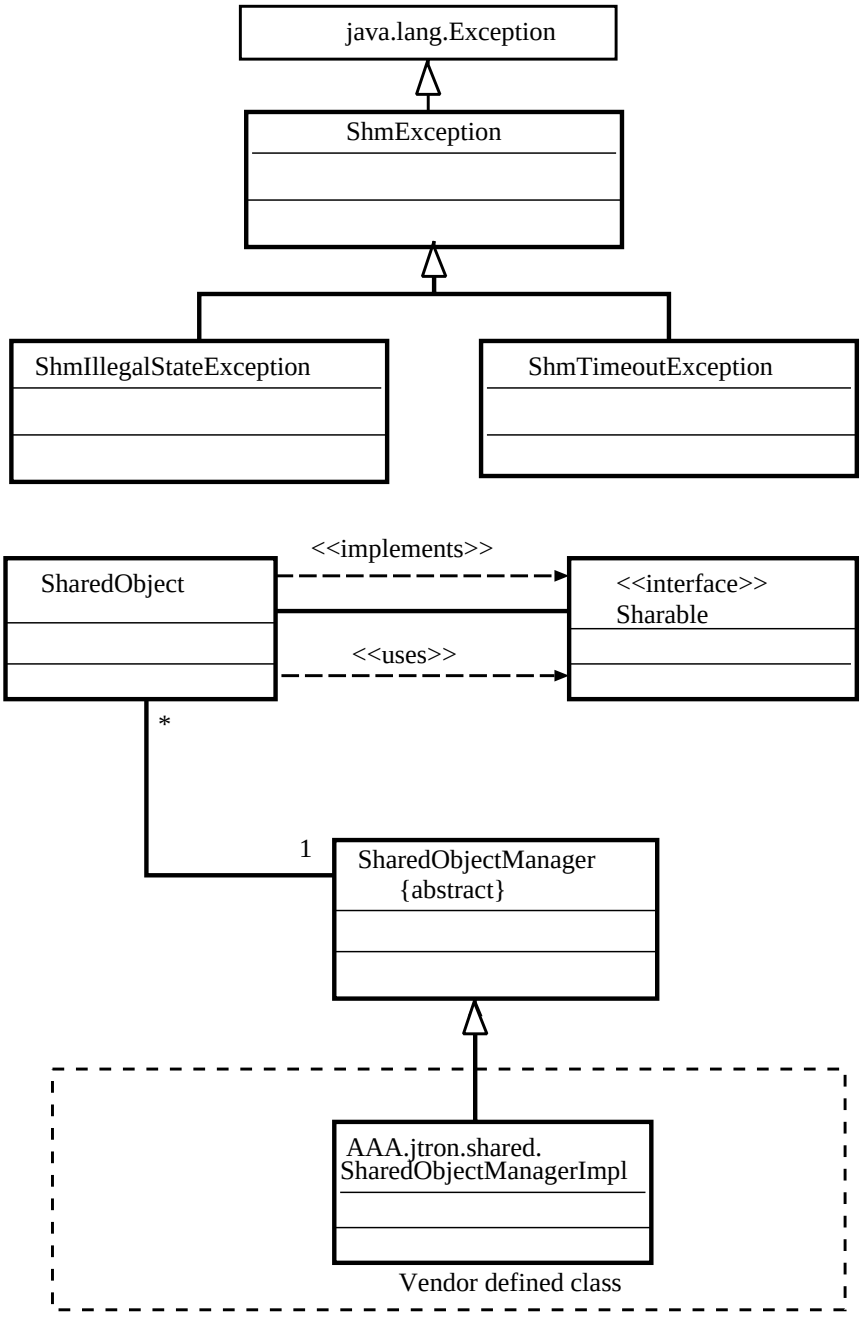


図 4.3: shared パッケージのクラス構成

4.3.2 インタフェース jp.gr.itron.jtron.shared.Sharable

public interface Sharable

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。

■ メソッド

public abstract void lock()
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。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 ShmTimeoutException
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。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 *timeout* (単位 ms) 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、 **ShmTimeoutException** を発生する。

public abstract void unlock() throws ShmIllegalStateException
同一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。すでにロックが解除されている場合は何もしない。
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
ShmIllegalStateException 例外が発生する。

public abstract void forceUnlock()
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。

public abstract void unshare() throws ShmIllegalStateException
リアルタイムタスク側とのオブジェクトの共有を終了する。すでにスレッド、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共有を終了する。すでに共有が終了されている場合は ShmIllegalStateException 例外が発生する。

public abstract void unshare(int timeout) throws ShmTimeoutException, ShmIllegalStateException
リアルタイムタスク側とのオブジェクトの共有を終了する。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 *timeout* (単位 ms) 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、 **ShmTimeoutException** 例外が発生する。ロックを解除した後共有を終了する。すでに共有していない場合は **ShmIllegalStateException** 例外が発生する。

public abstract Object getContent()
共有するオブジェクトを返す。 **SharedObjectManager** は、このメソッドを使って、実際に共有するオブジェクトを得る。

4.3.3 クラス jp.gr.itron.jtron.shared.SharedObject

```
java.lang.Object
   |
   +--- jp.gr.itron.jtron.shared.SharedObject
```

---

public class SharedObject
extends Object
implements Sharable

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。

■ 変数

JVvarprotected Sharable shm コンストラクタの引数 *shm* で指定されたオブジェクトを保持する。 *shm* の指定がないコンストラクタが呼ばれたときは、 this が設定される。

■ コンストラクタ

public SharedObject(String name) throws ShmIllegalStateException
name という名前で共有オブジェクトを生成する。生成時に、共有オブジェクトマネジャに登録され、リアルタイムタスクからアクセスできる状態になる。登録に失敗したときは、 **ShmIllegalStateException** 例外が発生する。

public SharedObject(Sharable shm, String name) throws ShmIllegalStateException
Sharable を実装したクラスのオブジェクト *shm* を *name* という名前で 21 共有オブジェクトにする。生成時に、マネジャに登録され、リアルタイムタスクから参照できる状態になる。登録に失敗したときは、 **ShmIllegalStateException** 例外が発生する。

■ メソッド

public void lock()
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。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 ShmTimeoutException
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。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 *timeout*(単位 ms) 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、 **ShmTimeoutException** 例外が発生する。

public vpid unlock() throws ShmIllegalStateException
同一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。すでにロックが解除されている場合は何もしない。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 **ShmIllegalStateException** 例外が発生する。

public void forceUnlock()
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。

public void unshare() throws ShmIllegalStateException
リアルタイムタスク側とのオブジェクトの共有を終了する。すでにスレッド、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共有を終了する。すでに共有していない場合は **ShmIllegalStateException** 例外が発生する。

public void unshare(int timeout) throws ShmTimeoutException, ShmIllegalStateException

リアルタイムタスク側とのオブジェクトの共有を終了する。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 *timeout* (単位 ms) 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、 **ShmTimeoutException** が発生する。ロックを解除した後共有を終了する。すでに共有していない場合は **ShmIllegalStateException** 例外が発生する。

public Object getContent()

共有するオブジェクトを返す。 **SharedObjectManager** は、このメソッドを使って、実際に共有するオブジェクトを得る。 **SharedObject** では、このメソッドは、コンストラ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 を返す実装になっている)。

SharedObject のサブクラスで別のオブジェクト (配列など) を共有対象にしたい場合は、このメソッドをオーバライドする。以下に例を示す。

```
public class SharedData extends SharedObject {
    protected int data[];
    public SharedData(String name) {
        super(name);
        data = new int[10];
    }

    public Object getContent() {
        return data;
    }
    .....
}
```

4.3.4 クラス jp.gr.itron.jtron.shared.SharedObjectManager

```
java.lang.Object
   |
   +--- jp.gr.itron.jtron.shared.SharedObjectManager
```

public abstract class SharedObjectManager

共有オブジェクトを管理するクラス。リアルタイムタスク側とのインタフェースを受け持つ。このクラスはアブストラクトクラスであり、ベンダはこのクラスを継承したサブクラスを提供する。

■ コンストラクタ

protected SharedObjectManager()
共有オブジェクトマネージャを生成する。

■ メソッド

public static SharedObjectManager getSharedObjectManager() throws ShmIllegalStateException
デフォルトの共有オブジェクトの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、 **ShmIllegalStateException** 例外が発生する。

public abstract void share(Sharable obj, String name) throws ShmIllegalStateException
obj を *name* という名前で登録する。すでに登録されている場合、不正な名前の場合は、 **ShmIllegalStateException** 例外が発生する。

public abstract void unshare(String name) throws ShmIllegalStateException
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 **ShmIllegalStateException** 例外が発生する.

public abstract void unshare(String name, int timeout) throws ShmIllegalStateException, ShmTimeoutException
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 **ShmIllegalStateException** 例外が発生する。

public abstract void lock(Sharable obj)
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。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(Sharable obj, int timeout) throws ShmTimeoutException
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。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。タイムアウト (単位: ms) が指定でき、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。タイムアウト時間が過ぎると **ShmTimeoutException** 例外を発生する。

public abstract void unlock(Sharable obj) throws ShmIllegalStateException
同一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。すでにロックが解除されている場合は何もしない。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 **ShmIllegalStateException** 例外が発生する。

public absract void forceUnlock(Sharable obj)
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。すでにロックが解除されている場合は何もしない。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死んだ場合に使用する。

4.3.5 クラス jp.gr.itron.jtron.shared.ShmException

```
java.lang.Object
   |
   +---- java.lang.Throwable
            |
            +---- java.lang.Exception
                     |
                     +---- jp.gr.itron.jtron.shared.ShmException
```

---

public class ShmException
extends Exception

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。

■ コンストラクタ

public ShmException()
詳細なメッセージなしで **ShmException** を生成する。

public ShmException(String msg)
指定された詳細メッセージ *msg* をもつ **ShmException** を生成する。

4.3.6 クラス jp.gr.itron.jtron.shared.ShmIllegalStateException

```
java.lang.Object
   |
   +--- java.lang.Throwable
           |
           +--- java.lang.Exception
                   |
                   +---- jp.gr.itron.jtron.shared.ShmException
                           |
                           +---- jp.gr.itron.jtron.shared.ShmIllegalStateException
```

---

pubic class ShmIllegalStateException
extends ShmException

あるメソッドを発行したときに、オブジェクトがそのメソッドが実行できない不正な状態であるときに発生する。

■ 変数

public static final int **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、マネージャが存在しない。

public static final int **OBJECT_IN_USE** = 2
すでにオブジェクトが登録されている。

public static final int **OBJECT_NOEXIST** = 3
そのオブジェクトは存在しない (すでに削除されている)。

public static final int **ILLEGAL_NAME** = 4
不正な名前である。

public static final int **OBJECT_UNSHARED** = 5
そのオブジェクトは共有されていない。

public static final int **OBJECT_LOCKED** = 6
そのオブジェクトが他のスレッド，あるいはタスクによってロックされている。

■ コンストラクタ

public ShmIllegalStateException(int cause)
指定された原因の例外オブジェクトを生成する。パラメタ *cause* には例外の詳細原因を渡す。

public ShmIllegalStateException(int cause, String msg)
指定された原因で、詳細なメッセージをもつ例外オブジェクトを生成する。通常、 *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()
例外の詳細原因を返す。

4.3.7 クラス jp.gr.itron.jtron.shared.ShmTimeoutException

```
java.lang.Object
   |
   +--- java.lang.Throwable
           |
           +--- java.lang.Exception
                   |
                   +--- jp.gr.itron.jtron.shared.ShmException
                           |
                           +---- jp.gr.itron.jtron.shared.ShmTimeoutException
```

---

public class ShmTimeoutException
extends ShmException

タイムアウト時間がすぎたことを通知する.

■ コンストラクタ

public ShmException()
詳細なメッセージなしで **ShmTimeoutException** を生成する。

public ShmTimeoutException(String msg)
指定された詳細メッセージ *msg* をもつ **ShmTimeoutException** を生成する。通常、 *msg* にはオブジェクト名が指定される。

第 5 章

ストリームインタフェース

5.1 概要

5.1.1 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ものである。

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。

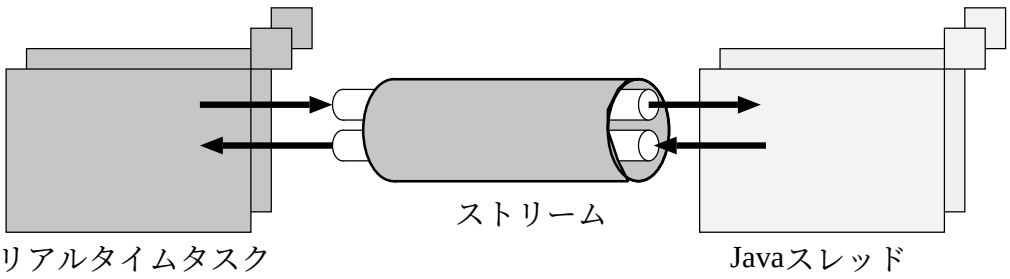


図 5.1: ストリーム

5.1.2 ストリームとチャネルの状態

ストリームは識別子番号で識別する。

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にある。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識別子番号を指定してオープンすることができる (**JtiDataStream**)。

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クローズすると (Java プログラムからは **outputStream** を **close**、リアルタイムタスクからは **jti_sht_stm**)、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だし、正常クローズされたこと検出し (Java プログラムは **inputStream.read** 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)。

JtiDataStream の **close** は、 **outputStream** と **inputStream** の両方の **close** と等価である.

【補足説明】

jti_rea_stm が 0 を返す (ないしは **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 (ないしは強制クローズ) 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** (ないしは **jti_wri_stm** か **jti_sht_stm**) を呼びだしても、 0 (ないしは **E_CLS** は返らないので注意が必要である。

ストリームの状態は、「未生成」と「未接続」に加え、両チャネルの状態によって 11 通りの状態をとる。具体的には、 Java プログラムからリアルタイムタスクへのチャネルが「接続」、「送信終了」、「接続」の 3 状態、リアルタイムタスクから Java プログラムへのチャネルが「接続」、「送信終了」、「強制切断」、「接続」の 4 状態を持つが、両チャネルが切断状態になるとストリームは未接続状態に戻るため、ストリームがとる状態は 13 $(= 2 + 3 \times 4 - 1)$ 通りとなる.

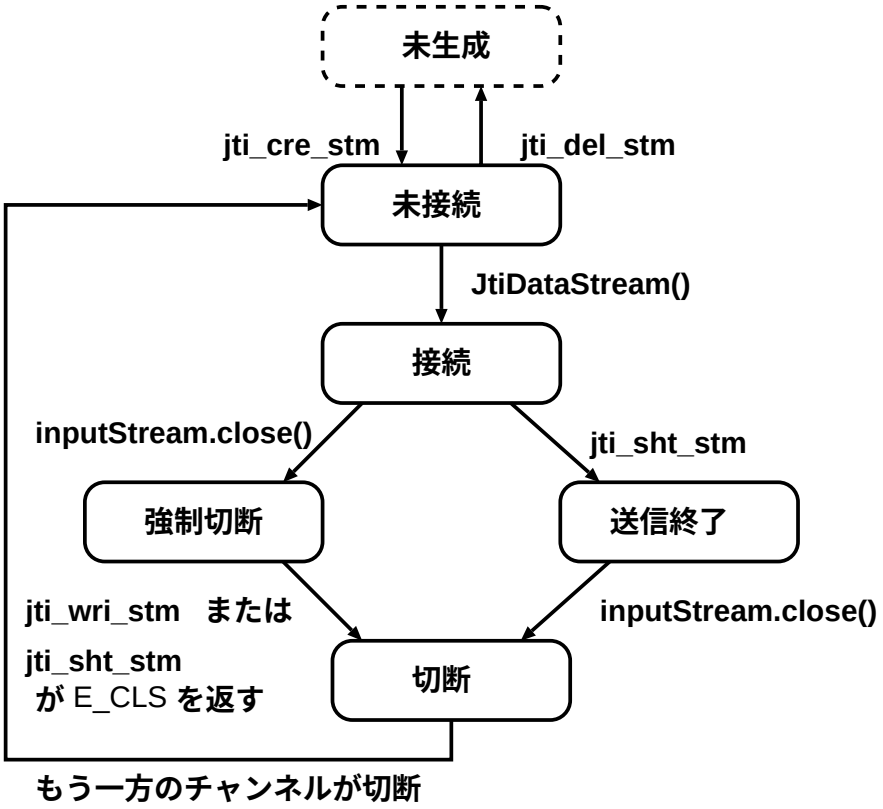


図 5.2: リアルタイムタスクから Java プログラムへのチャネルの状態遷移

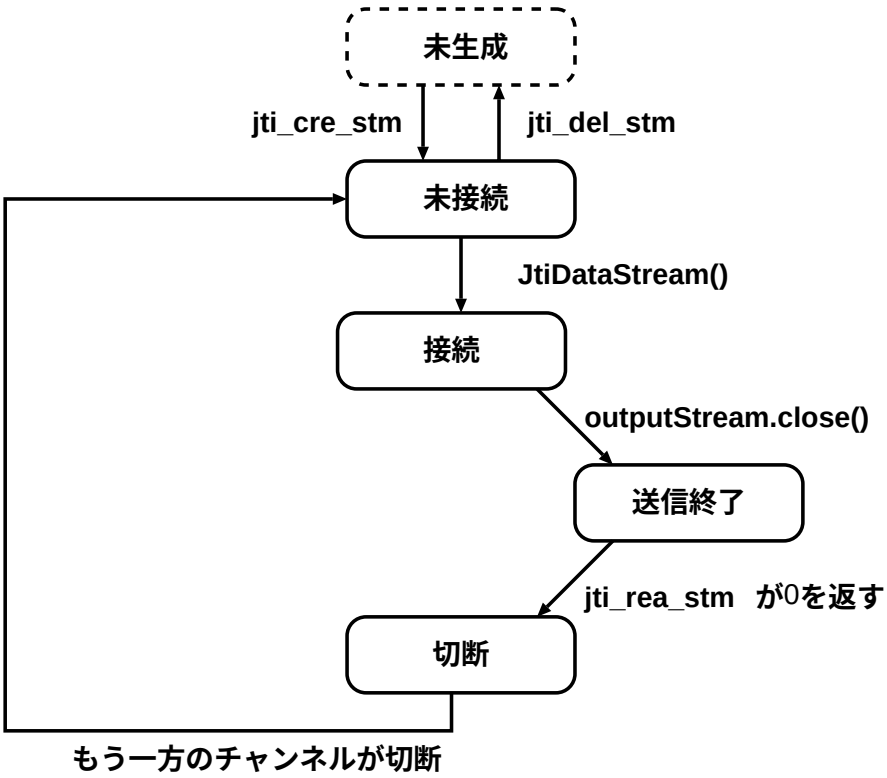


図 5.3: Java プログラムからリアルタイムタスクへのチャネルの状態遷移

5.2 ITRON API

5.2.1 ストリームの生成 / 削除

API 名	概要	種別
jti_cre_stm, JTI_CRE_STM	ストリームの生成	標準仕様
jti_del_stm	ストリームの削除	標準仕様

標準仕様

jti_cre_stm, JTI_CRE_STM

ストリームの生成

【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 });
```

【パラメタ】

ID	*stmid*	ストリーム識別子
T_JTI_CSTM	**pk_cstm*	ストリーム生成情報

pk_cstm **の内容**

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

(他に実装依存のパラメタがあってもよい)

【戻り値】

ER	*ercd*	エラーコード

【エラーコード】

E_OK	正常終了
E_ID	不正 ID 番号
E_RSATR	予約属性
E_PAR	パラメタエラー (*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)

【API の機能】

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。ストリーム属性に **TA_WRITE**、 **TA_READ** 以外を指定した場合は、 **E_RSATR** となる。なお、 **TA_WRITE** と **TA_READ** の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。送受信バッファのサイズが負の場合は、 **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)。

バッファを内部で確保する実装では、バッファの先頭アドレスとして **NADR**(= −1) を指定させるものとする。その場合にも、バッファのサイズの指定は有効である。また、 **NADR** が指定された場合にはバッファを内部で確保し、それ以外の場合には与えられたバッファを用いる実装も可能である。

標準仕様

jti_del_stm

ストリームの削除

【C 言語 API】

```
ER ercd = jti_del_stm(ID stmid);
```

【パラメタ】

ID *stmid* ストリーム識別子

【戻り値】

ER *ercd* エラーコード

【エラーコード】

E_OK	正常終了
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)

【API の機能】

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、ECDE_DLT を返す。

5.2.2 データの送受信と送信終了

API 名	概要	種別
jti_wri_stm	データの送信	標準仕様
jti_rea_stm	データの受信	標準仕様
jti_sht_stm	データ送信の終了	標準仕様

標準仕様

jti_wri_stm

データの送信

【C 言語 API】

```
ER ercd = jti_wri_stm(ID stmid, VP data, INT len, TMO tmout);
```

【パラメタ】

ID	*stmid*	ストリーム識別子
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	*tmout*	タイムアウト指定

【戻り値】

ER	*ercd*	バッファに入れたデータの長さ／エラーコード

【エラーコード】

正の値	正常終了 (送信バッファに入れたデータの長さ)
E_ID	不正 ID 番号
E_NOEXS	オブジェクト未生成
E_PAR	パラメタエラー (*data*、 *len*、 *tmout* 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	送信用のチャネルが強制切断された

【API の機能】

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、送信バッファが一杯になるまで送信バッファにデータを入れ、送信バッファに入れたデータの長さを返す。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の時は、 **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** を呼びだ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。

標準仕様

jti_rea_stm

データの受信

【C 言語 API】

```
ER ercd = jti_rea_stm(ID stmid, VP data, INT len, TMO tmout);
```

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	*tmout*	タイムアウト指定

【戻り値】

ER	*ercd*	取り出したデータの長さ／エラーコード

【エラーコード】

正の値	正常終了 (取り出したデータの長さ)
0	データ終結 (接続が正常切断された)
E_ID	不正 ID 番号
E_NOEXS	オブジェクト未生成
E_PAR	パラメタエラー (*data*、 *len*、 *tmout* 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

【API の機能】

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ると、 API から 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、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_rea_stm** を呼びだ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。

標準仕様

jti_sht_stm

データ送信の終了

【C 言語 API】

```
ER ercd = jti_sht_stm(ID stmid);
```

【パラメタ】

ID *stmid* ストリーム識別子

【戻り値】

ER *ercd* エラーコード

【エラーコード】

E_OK	正常終了
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状態ないしは未接続状態、**jti_wri_stm** がペンディング中)
E_CLS	送信用のチャネルが強制切断された

【API の機能】

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。

5.2.3 ストリームの状態参照

API 名	概要	種別
jti_ref_stm	ストリームの状態参照	標準仕様

標準仕様

jti_ref_stm

ストリームの状態参照

【C 言語 API】

```
ER ercd = jti_ref_stm(ID stmid, T_JTI_RSTM *pk_rstm);
```

【パラメタ】

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

【戻り値】

ER	*ercd*	エラーコード

pk_rstm の内容

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)

(他に実装依存のパラメタがあってもよい)

【エラーコード】

E_OK	正常終了
E_ID	不正 ID 番号
E_NOEXS	オブジェクト未生成
E_PAR	パラメタエラー (*pk_rstm* のアドレスが不正)

【API の機能】

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、待たず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。

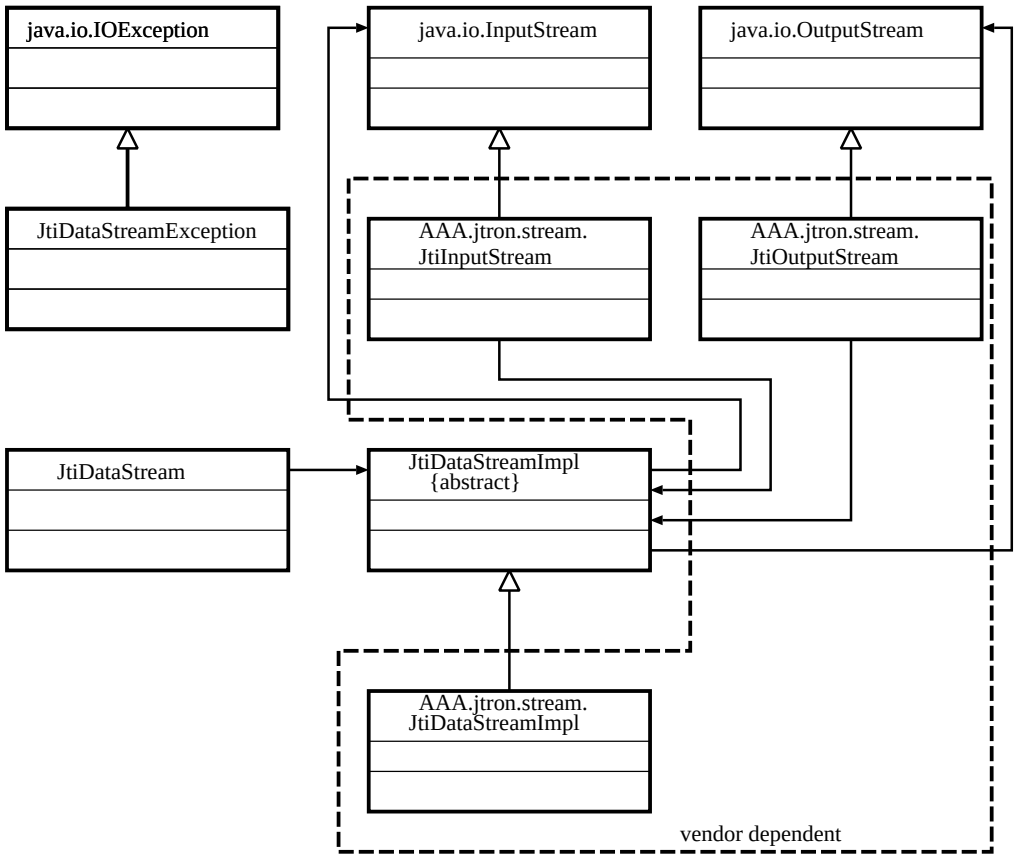
5.3 Java API

5.3.1 パッケージ構成

ストリームを提供するクラスは、jp.gr.itron.jtron.stream パッケージにまとめられている。以下のクラス、例外クラスから構成される。

クラス: JtiDataStream

例外クラス: JtiDataStreamException



JtiDataStreamImpl と各ストリームクラスとの関連は、実装依存である。ここに示した関連は実装の 1 例である。ある実装では JtiDataStreamImpl のサブクラス (この図では AAA.jtron.stream.JtiDataStreamImpl) とストリームクラスに関連があるかもしれない。

図 5.4: stream パッケージのクラス構成

stream パッケージのクラス構成を図 5.4に示す。**JtiDataStream** を実装するにあたって、ブリッジ [5][6] と呼ばれるデザインパターンを使って、**JtiDataStream** とその実装である **JtiDataStreamImpl** を分離した。これによりベンダ間で異なるストリーム実装を容易に交換できるようになる。ブリッジは **java.io.Socket** でも使用されている。

5.3.2 クラス jp.gr.itron.jtron.stream.JtiDataStream

```
java.lang.Object
   |
   +--- jp.gr.itron.jtron.stream.JtiDataStream
```

public class JtiDataStream

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。

■ 変数

public static final int **MAIN_STREAM** = 1
リアルタイムタスクと Java プログラムの間で用いる標準的なストリームの識別子

■ コンストラクタ

public JtiDataStream(int stmid)throws JtiDataStreamException
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が接続状態になる)。指定した識別子が既に使用されている場合は **JtiDataStreamException** が発生する。

public JtiDataStream(int stmid, int timeout)throws IOException,InterruptedException
タイムアウト時間を指定して、指定した識別子のストリームをオープンする。タイムアウト時間の単位はミリ秒。指定した識別子が既に使用されている場合は **JtiDataStreamException** が発生する。タイムアウト発生時は、例外 **InterruptedException** が発生する。

protected JtiDataStream(JtiDataStreamImpl impl, int stmid, int timeout) throws IOException, InterruptedException
ユーザ定義の実装を用いて、タイムアウト時間を指定して、指定した識別子のストリームをオープンする。タイムアウト時間の単位はミリ秒。指定した識別子が既に使用されている場合は **JtiDataStreamException** が発生する。タイムアウト発生時は、例外 **InterruptedException** が発生する。

■ メソッド

public synchronized InputStream getInputStream() throws IOException
受信ストリームを取得する。

public synchronized OutputStream getOutputStream() throws IOException
送信ストリームを取得する。

public synchronized void setIDSTimeOut(int timeout) throws IOException
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を設定する。タイムアウト時間の単位はミリ秒。 *timeout* は 0 以上でなければならない。 *timeout* に 0 を指定した場合は無限待ちとなる。タイムアウト発生時は **java.io.InterruptedException** 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と同様にタイムアウト指定はできない。

public synchronized int getIDSTimeOut() throws IOException
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を取得する。タイムアウト時間の単位はミリ秒である。 0 が返却された場合は無限待ちを意味する。

public synchronized void close() throws IOException
ストリームをクローズする。具体的には、送信用のチャネルが接続状態なら正常クローズ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強制クローズ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には、切断状態にする。

5.3.3 クラス jp.gr.itron.jtron.stream.JtiDataStreamImpl

```
java.lang.Object
   |
   +--- jp.gr.itron.jtron.stream.JtiDataStreamImpl
```

public abstract class JtiDataStreamImpl

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。

■ コンストラクタ

public JtiDataStreamImpl() throws JtiDataStreamException

■ メソッド

説明のないものは、 jp.gr.iton.jtron.stream.JtiDataStream の対応するメソッドと同じ仕様である。

public abstract void setTimeout(int timeout)
タイムアウト時間を設定する。

public abstract void setStreamId(int stmid)
識別子を設定する。

public abstract int getTimeout(int timeout)
タイムアウト時間を取得する。

public abstract int getStreamId(int stmid)
識別子を取得する。

public abstract InputStream getInputStream() throws IOException

public abstract OutputStream getOutputStream() throws IOException

public abstract void setIDSTimeOut(int timeout) throws IOException

public abstract int getIDSTimeOut() throws IOException

public abstract void close() throws IOException

5.3.4 クラス jp.gr.itron.jtron.stream.JtiDataStreamException

```
java.lang.Object
   |
   +--- java.lang.Throwable
           |
           +--- java.lang.Exception
                   |
                   +--- java.io.IOException
                           |
                           +--- jp.gr.itron.jtron.stream.JtiDataStreamException
```

---

public class JtiDataStreamException extends IOException

ストリーム通信関連の例外が発生したことを通知する。

■ コンストラクタ

public JtiDataStreamException(int cause)
詳細なメッセージなしで **JtiDataStreamException** を生成する。 パラメタ *cause* には、例外の詳細原因を渡す。

public JtiDataStreamException(int cause, String msg)
指定された詳細メッセージ *msg* をもつ **JtiDataStreamException** を生成する。パラメタ *cause* には、例外の詳細原因を渡す。

■ メソッド

public int getCause()
例外の詳細原因を返す。

■ 変数

public static final int **STREAM_NOT_FOUND** = 1
指定した識別子のストリームは存在しない (未生成)。

public static final int **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。

public static final int **STREAM_CLOSED** = 3
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。

public static final int **STREAM_ILLEGAL_ARGUMENT** = 4
指定した引数が不正。

付録 A

付録

A.1 アタッチクラス

TBD

A.2 共有オブジェクトインタフェース

A.2.1 定義例

(1) 継承する場合

他のクラスのサブクラスでない場合は, SharedOject クラスを継承する方法を推奨する.

- 例 1

```
public class MyObject extends SharedObject {
    private int x;
    private int y;
    private int z;

    public MyObject(String name) {
      super(name);
      ...
    }
    ....
}
```

- 例 2

```
public class SharedData extends SharedObject {
    int data[];
    ....
    public SharedData(String name) {
      super(name);
      data = new int[10];
      ...
    }
    public Object getContent() {
        return data;
    }
    ....
}
```

(2) Sharable インタフェースを利用する場合

SharedObject クラスをメンバとしてもたせる.

```
public class FooObject implements Sharable {
    private SharedObject shm;
    int x;
    int y;
    int z;

    public FooObject(String name) {
        shm = new SharedObject(this, name);
    }
    public void lock() {
        shm.lock();
    }
    public void lock(int timeout) {
        shm.locj(timeout);
    }
    public void unlock() {
        shm.unlock();
    }
    public void forceUnlock() {
        stm.forceUnlock();
    }
    public void unshare() {
        shm.unshare();
    }
   public void unshare(int timeout) {
        shm.unshare(timeout);
   }
   public void getContent() {
      return this;
   }
}
```

A.2.2 リアルタイムタスク，Java プログラムによる通信例

Java 側 (JtiSharedSample.java)

List A.1 JtiSharedSample.java

```
1   /**
2    * Shared Object sample
3    */
4   import jp.gr.itron.jtron.shared.*;
5
6   class SharedData extends SharedObject {
7       private int data;
8
9       SharedData(String name) throws ShmIllegalStateException {
10          super(name);
11          data = 0;
12      }
13
14      public int getData() {
15          return data;
16      }
17  }
18
19  public
20  class JtiSharedSample {
21      private SharedData data = null;
22      private int sum;
23
24      JtiSharedSample() {
25          sum = 0;
26          try {
27              data = new SharedData("Shared");
28          } catch (ShmIllegalStateException ex) {
29              System.out.println("error: code =" + ex.getCause());
30              System.exit(1);
31          } catch (ShmException ex) {
32              System.out.println("error:" + ex);
33              system.exit(1);
34          }
35      }
36
37      public void dispose() {
38          try {
39              data.unshare(); // unshare する
40          } catch (ShmIllegalStateException ex) {
41              System.out.println("error: already unshared.");
42          }
43      }
44
45      public void startSharedSample() {
46          int c;
47
48          try {
49              while(true) {
50                  c = 0;
51                  try {
52                      data.lock(10);      // リアルタイムタスクからのデータを待つ
53                      c = data.getData(); // データを取得
54                      data.unlock();
55                      if (c == -1) {      // -1 がおくられてきたら終了.
56                          break;
57                      }
58                      sum += c;           // データを処理する
59                  } catch (ShmTimeoutException ex) {
```

```
60                      /* timeout のときの処理: ここでは、 10ms 寝る */
61                      try {
62                          Thread.sleep(10);
63                      } catch (InterruptedException e) {
64                          /* nop */
65                      }
66                  }
67              }
68              System.out.println("sum      = " + sum);
69          } catch (ShmIllegalStateException ex) {
70              System.out.println("internal error:" + ex);
71          }
72      }
73
74      public static void main(String args[]) {
75          JtiSharedSample app = new JtiSharedSample();
76          app.startSharedSample();
77          app.dispose();
78      }
79  }
```

ITRON 側 (jtron.c)

List A.2 jtron.c

```
1   #include "jti_shared.h"
2   #define  WAIT_TIME  10 /* ロックの最大待ち時間 */
3
4   struct JSharedObj *p;  /* 共有されるオブジェクトの javah による構造体 */
5
6   void maintask() {
7       JNO  shoid;
8       ER  ercd;
9       /* jti_get_obj で名前から共有オブジェクト id を獲得 */
10      ercd = jti_get_obj("Shared", &shoid);
11
12      while(1) {
13          /* jti_loc_obj で共有オブジェクトをロックしようとする．待ち時間は 10 ミリ秒  */
14          ercd = jti_loc_obj(shoid, MAX_TIME);
15          if (ercd == E_OK) { /* ロックできた場合 */
16              /* jti_get_mem で共有オブジェクトをアドレスを獲得 */
17              jti_get_mem(&p, shoid);
18              /* Java プログラム側に渡すデータを準備 */
19              /* p の指す領域にデータを設定する */
20
21              /* jti_unl_obj で共有オブジェクトをアンロック */
22              ercd = jti_unl_obj(shoid);
23          }
24      }
25  }
```

A.3 ストリームインタフェース

A.3.1 リアルタイムタスク，Java プログラムによる通信例

ITRON 側 (jtron.c)

List A.3 jtron.c

```
1   #include "jti_stream.h"
2   #define SIZE_WBUF 100
3   char WBUF[SIZE_WBUF];
4
5   #define N_DATA 100
6
7   /* ITRON => Java */
8
9   void maintask() {
10      ER ercd;
11      T_JTI_CSTM pk_cstm;
12      int writedata, readdata;
13      int i;
14
15      /* ストリームの生成（ライト側のみ）*/
16      pk_cstm.exinf = 0;
17      pk_cstm.stmatr = TA_WRITE;
18      pk_cstm.wbuf = WBUF;
19      pk_cstm.wbufsz = SIZE_WBUF;
20      pk_cstm.rbuf = 0;
21      pk_cstm.rbufsz = 0;
22
23      ercd = jti_cre_stm(JTI_MAIN_STREAM, &pk_ctsm);
24
25      /* データの送信 */
26      /* Java 側で ItronDataStream() が呼ばれるまで待ち状態になる */
27      /* N_DATA 個のデータを ITRON 側から送る */
28
29      for (i = 0; i < N_DATA; i++) {
30          writedata = i;
31          ercd = jti_wri_stm(JTI_MAIN_STREAM, &writedata, sizeof(int), TMO_FEVR);
32          /* エラーのときには何をするべきか？ */
33          if (ercd != E_OK) {
34              /* エラー処理 */
35          }
36      }
37      /* データ送信の終了 */
38      ercd = jti_sht_stm(JTI_MAIN_STREAM);
39      /* ストリームの削除 */
40      ercd = jti_del_stm(JTI_MAIN_STREAM);
41
42      ext_tsk();
43  }
```

Java 側 (JtiStreamSample.java)

List A.4 JtiStreamSample.java

```
1   import java.net.*;
2   import java.io.*;
3   import java.util.*;
4
5   import jp.gr.itron.jtron.*;
6
7   public class JtiStreamSample {
8       public static void main(String args[]) {
9           JtiSteamSample jtiss = new JtiStreamSample();
10          jtiss.startStreamSample();
11      }
12
13      public void startStreamSample() {
14          JtiDataStream ids = null;
15          InputStream is = null;
16          int c = 0;
17
18          try {
19              // ITRON 側と接続
20              ids = new JtiDataStream(MAIN_STREAM);
21
22              // InputStream を得る
23              is = ids.getInputStream();
24              while (true) {
25                  // ITRON 側から送られてくるデータを得る
26                  c = is.read();
27
28                  if (c == -1) {
29                      // EOF が送られてきたらクローズして処理終了
30                      is.close();
31                      return;
32                  }
33                  // リードしたデータに対する処理
34              }
35              is.close();
36              return;
37          } catch (IOException ioe) {
38              System.out.println("Exception:" + ioe.getMessage());
39              return;
40          }
41      }
42  }
```

索引

【A】～【Z】