Cisco SCMS SM Java API プログラマ ガイド

Release 3.1
May 2007

Text Part Number: OL-7204-05-J

このマニュアルに記載されている仕様および製品に関する情報は、予告なしに変更されることがあります。このマニュアルに記載されている表現、情報、および推奨事項は、すべて正確であると考えていますが、明示的であれ黙示的であれ、一切の保証の責任を負わないものとします。このマニュアルに記載されている製品の使用は、すべてユーザ側の責任になります。

対象製品のソフトウェア ライセンスおよび限定保証は、製品に添付された『Information Packet』に記載されています。添付されていない場合には、代理店にご連絡ください。

シスコシステムズが採用している TCP ヘッダー圧縮機能は、UNIX オペレーティング システムの UCB（University of California, Berkeley）パブリック ドメイン バージョンの一部として、UCB が開発したプログラムを最適化したものです。All rights reserved. Copyright © 1981, Regents of the University of California.

ここに記載されている他のいかなる保証にもよらず、各社のすべてのマニュアルおよびソフトウェアは、障害も含めて「現状のまま」として提供されます。シスコシステムズおよびこれら各社は、商品性や特定の目的への準拠性、権利を侵害しないことに関する、または取り扱い、使用、または取引によって発生する、明示されたまたは黙示された一切の保証の責任を負わないものとします。

いかなる場合においても、シスコシステムズおよびその代理店は、このマニュアルの使用またはこのマニュアルを使用できないことによって起こる制約、利益の損失、データの損傷など間接的で偶発的に起こる特殊な損害のあらゆる可能性がシスコシステムズまたは代理店に知らされていても、それらに対する責任を一切負いかねます。

CCVP, the Cisco logo, and the Cisco Square Bridge logo are trademarks of Cisco Systems, Inc.; Changing the Way We Work, Live, Play, and Learn is a service mark of Cisco Systems, Inc.; and Access Registrar, Aironet, BPX, Catalyst, CCDA, CCDP, CCIE, CCIP, CCNA, CCNP, CCSP, Cisco, the Cisco Certified Internetwork Expert logo, Cisco IOS, Cisco Press, Cisco Systems, Cisco Systems Capital, the Cisco Systems logo, Cisco Unity, Enterprise/Solver, EtherChannel, EtherFast, EtherSwitch, Fast Step, Follow Me Browsing, FormShare, GigaDrive, HomeLink, Internet Quotient, IOS, iPhone, IP/TV, iQ Expertise, the iQ logo, iQ Net Readiness Scorecard, iQuick Study, LightStream, Linksys, MeetingPlace, MGX, Networking Academy, Network Registrar, *Packet*, PIX, ProConnect, ScriptShare, SMARTnet, StackWise, The Fastest Way to Increase Your Internet Quotient, and TransPath are registered trademarks of Cisco Systems, Inc. and/or its affiliates in the United States and certain other countries.

All other trademarks mentioned in this document or Website are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (0705R)

このマニュアルで使用している IP アドレスは、実際のアドレスを示すものではありません。マニュアル内の例、コマンド出力、および図は、説明のみを目的として使用されています。説明の中に実際のアドレスが使用されていたとしても、それは意図的なものではなく、偶然の一致によるものです。

Cisco SCMS SM Java API プログラマ ガイド
Copyright ©2007 Cisco Systems, Inc.
All rights reserved.

CONTENTS

はじめに

May 30, 2007, OL-7204-05-J

Java API は、SCMS Subscriber Manager（SM）のアップデート、クエリー、設定に使用できます。この API は次に示す 2 つの部分で構成されています。これらは、個別に使用できるだけでなく、制限なく一緒に使用することが可能です。

- SM Non-blocking Java API：高性能 API。エラーやその他の操作結果のわかりやすさは重視されていません。OSS/AAA システムとの自動統合をサポートしています。
- SM Blocking Java API：使いやすさを重視した API。SM のアクセスおよび管理用のユーザ インターフェイス アプリケーションをサポートしています。

（注） C/C++ 環境でも、まったく同じ機能を持つ API セットが用意されています。

この章では、次の内容について説明します。

対象読者

このマニュアルは、SM の設定を担当するネットワーク技術者またはコンピュータ技術者、および SCE プラットフォームを管理するオペレータを対象としています。

マニュアルの変更履歴

Cisco Service Control リリース	Part Number	発行日
Release 3.1.0	OL-7204-05	2007 年 5 月

変更の説明

- ドメイン間のサブスクライバ移動のサポート。「サブスクライバ ドメイン」（p.2-6）を参照してください。また、login メソッドについては、「パラメータ」（p.3-6）を参照してください。
- 「サブスクライバ名のフォーマット」（p.2-4）をアップデート。

Cisco Service Control リリース	Part Number	発行日
Release 3.0.5	OL-7204-04	2006 年 11 月

変更の説明

- Release 3.0.5 にアップデートしたマニュアル。

Cisco Service Control リリース	Part Number	発行日
Release 3.0.3	OL-7204-03	2006 年 5 月

変更の説明

- Release 3.0.3 にアップデートしたマニュアル。

Cisco Service Control リリース	Part Number	発行日
Release 3.0	OL-7204-02	2005 年 12 月

変更の説明

- マニュアルの再構成。このリリースには、主な変更や新しい機能は追加されませんでした。

Cisco Service Control リリース	Part Number	発行日
Release 2.5.7	OL-7204-01	2005 年 5 月

変更の説明

- このマニュアルの初版。

マニュアルの構成

このマニュアルの構成は、次のとおりです。

表 1

章	タイトル	説明
第 1 章	使用する前に	Java API を使用できるプラットフォーム、Java API コンポーネントのインストール方法、コンパイル方法、および実行の開始方法について説明します。
第 2 章	API の基礎知識	SM Java API を使用する際に役立つさまざまな事項について説明します。
第 3 章	ブロッキング API	ブロッキング API の機能と操作について説明し、コードの例をいくつか示します。
第 4 章	ノンブロッキング API	ノンブロッキング API の機能と操作について説明し、コードの例をいくつか示します。
付録 A	エラー コードのリスト	Java API で使用するコード エラー リストを提供します。

関連資料

このマニュアルは、SCMS Subscriber Manager のユーザ ガイド、API ガイド、リファレンス ガイドと併せてご利用ください。

表記法

このマニュアルでは、次の表記法を使用しています。

- **太字** は、コマンド、キーワード、およびボタンを示しています。
- *イタリック体*は、ユーザが値を指定する引数を示しています。
- `screen` フォントは、スクリーンに表示される情報の例を示しています。
- **太字**の **`screen`** フォントは、ユーザが入力する情報の例を示しています。
- 縦棒（|）は、選択要素の区切りを示しています。
- 角カッコ（[ ]）は、省略可能な要素を示します。
- 波カッコ（{}）は、必須の選択要素を示しています。
- 角カッコ内の波カッコおよび縦棒（[{}]）は、省略可能な要素の中の必須選択肢を示しています。

（注） 「*注釈*」です。役立つ情報や、このマニュアル以外の参照資料などを紹介しています。

ワンポイント・アドバイス 「*時間の節約に役立つ操作*」です。記述されている操作を実行すると時間を節約できます。

注意 「*要注意*」の意味です。機器の損傷またはデータ損失を予防するための注意事項が記述されています。

警告 **「*危険*」の意味です。人身事故を予防するための注意事項が記述されています。機器の取り扱い作業を行うときは、電気回路の危険性に注意し、一般的な事故防止対策に留意してください。**

マニュアルの入手方法、テクニカル サポート、およびシスコのセキュリティ ガイドライン

マニュアルの入手方法、テクニカル サポート、マニュアルに関するフィードバックの提供、セキュリティ ガイドラインに関する情報、および推奨するエイリアスと一般的なシスコのマニュアルに関する情報については、次の URL で、毎月更新される『*What's New in Cisco Product Documentation*』を参照してください。シスコの新規および改訂版の技術マニュアルの一覧が示されています。

http://www.cisco.com/en/US/docs/general/whatsnew/whatsnew.html

Japan TAC Web サイト

Japan TAC Web サイトでは、利用頻度の高い TAC Web サイト（http://www.cisco.com/tac）のドキュメントを日本語で提供しています。Japan TAC Web サイトには、次の URL からアクセスしてください。

http://www.cisco.com/jp/go/tac

サポート契約を結んでいない方は、「ゲスト」としてご登録いただくだけで、Japan TAC Web サイトのドキュメントにアクセスできます。

Japan TAC Web サイトにアクセスするには、Cisco.com のログイン ID とパスワードが必要です。ログイン ID とパスワードを取得していない場合は、次の URL にアクセスして登録手続きを行ってください。

http://www.cisco.com/jp/register/

■ マニュアルの入手方法、テクニカル サポート、およびシスコのセキュリティ ガイドライン

CHAPTER 1

使用する前に

ここでは、Java API を使用できるプラットフォームのほか、インストール、コンパイル、実行開始の手順について説明します。

Java API

Java API とは

Java API は、SCMS Subscriber Manager（SM）のアップデート、クエリー、設定に使用できます。この API は次に示す 2 つの部分で構成されています。これらは、個別に使用できるだけでなく、制限なく一緒に使用できます。

- SM Non-blocking Java API ― 高性能 API。エラーやその他の操作結果のわかりやすさは重視されていません。OSS/AAA システムとの自動統合をサポートしています。
- SM Blocking Java API ― 使いやすさを重視した API。SM のアクセスおよび管理用のユーザ インターフェイス アプリケーションをサポートしています。

プラットフォーム

SM Java API は、Windows プラットフォームで開発され試験されましたが、Java バージョン 1.4 をサポートしている任意のプラットフォームで動作可能です。

パッケージの内容

簡潔に示すため、以下ではインストレーション ディレクトリ **sm-java-api-vvv.bb** を **<installdir>** と示します。

The **<installdir>/javadoc** フォルダには、API JAVADOC マニュアルが入っています。

The **<installdir>lib** フォルダには、**smapi.jar** ファイルが入っています。このファイルは、API 実行可能ファイルです。このフォルダには、API 操作に必要なその他の jar ファイルも入っています。

表 1-1　インストレーション ディレクトリのレイアウト

パス	名前	説明
<installdir>		
	README	API readme ファイル
<installdir>/javadoc		
	index.html	すべての API 仕様のインデックス
	（API 仕様ファイルなど）	API 仕様書
<installdir>/lib		
	smapi.jar	SM API 実行可能ファイル
	asn1rt.jar	API で使用される jar ユーティリティ
	jdmkrt.jar	API で使用される jar ユーティリティ
	ilog4j.jar	API で使用される jar ユーティリティ
	log4j.properties	API ロギング機能に必要なプロパティ ファイル
	xerces.jar	API で使用される jar ユーティリティ

Java API のインストール

Java API ディストリビューションは、SCMS SM-LEG ディストリビューション ファイルの一部で、**sm_api** ディレクトリに配置されています。

Java SM API は、UNIX tar ファイルに実装されています。Java SM API は UNIX tar ユーティリティまたはほとんどの Windows 圧縮ユーティリティで展開できます。

SM のセットアップ

API は SM 上の PRPC サーバに接続します。API を機能させるには、次の条件を満たす必要があります。

- SM が起動され稼働中であり、API のホスト マシンから到達できること。
- PRPC サーバが起動されていること。

PRPC サーバはシスコが開発した独自の PRC プロトコルです。PRPC サーバの詳細については、『*SCMS Subscriber Manager User Guide*』を参照してください。

Unix または Linux プラットフォームへのインストール

（注） 略語の **vvv** および **bb** は、Java SM API のバージョンとビルド番号を示しています。

ステップ 1　SCMS SM-LEG ディストリビューション ファイルを解凍します。

ステップ 2　Java SM API ディストリビューション tar **sm-java-api-dist.tar** を検索します。

ステップ 3　Java SM API ディストリビューション tar を解凍し、sm-java-api-vvv.bb.tar を取得します。

```
#>tar -xvf sm-java-api-dist.tar
```

ステップ 4　Java SM API パッケージ tar を解凍します。

```
#>tar -xvf sm-java-api-vvv.bb.tar
```

Windows プラットフォームへのインストール

ステップ 1　zip 圧縮 / 解凍ユーティリティ（WinZip など）を使用します。

コンパイルと実行

SM Java API を使用するプログラムをコンパイルして実行するには、**smapi.jar** を CLASSPATH に入れる必要があります。

たとえば、プログラム ソースが **SMApiProgram.java** にある場合は、次のコマンドラインを使用してプログラムをコンパイルします。

```
#>javac -classpath smapi.jar SMApiProgram.java
```

プログラムをコンパイルしたあと、次のコマンドラインを使用してプログラムを実行します。

```
#>java -cp .;<installdir>/lib/smapi.jar SMApiProgram
```

CHAPTER 2

API の基礎知識

この章では、SM Java API を使用する際に役立つさまざまな事項を紹介します。

ブロッキング API とノンブロッキング API

ここでは、ブロッキング API とノンブロッキング API の操作の相違点について説明します。

ブロッキング API

一般的な API であるブロッキング API 操作では、どのメソッドも、操作の*完了後に*戻ります。

SM Blocking Java API は広範な操作を提供します。この API には、ノンブロッキング API の機能のほとんどに加え、ノンブロッキング API にはない機能も多く含まれています。ブロッキング API は、信頼性機能や自動再接続機能には対応しません。

ノンブロッキング API

ノンブロッキング Java API 操作では、メソッドはたとえ操作が完了していなくても*ただちに*戻ります。操作結果は、Observer オブジェクト（リスナ）に戻るか、またはまったく戻りません。

ノンブロッキング API メソッドは、入出力が含まれていて時間がかかるような操作の場合に有効です。別のスレッドで操作を実行すれば、呼び出しプログラムはほかのタスクを続行できるので、全体のシステム パフォーマンスが向上します。

SM Non-blocking Java API には、ノンブロッキング操作がいくつか含まれています。この API は、結果リスナによる操作結果の取得をサポートしています。

SM Non-blocking Java API は、信頼と非信頼という 2 種類のモードで使用できます。信頼モードについては、「信頼性のサポート」（p.4-2）を参照してください。

API の初期化

API の初期化には、主に次の 3 つの手順があります。

- いずれかのコンストラクタを使用し、API を構築します。
- API 固有のセットアップ操作を実行します。
- API を SM に接続します。

以下では、上記の 3 つの手順について詳しく説明します。

初期化の例は、各 API のコード例を参照してください。

- API の構築（p.2-3）
- セットアップ操作（p.2-3）
- SM への接続（p.2-4）

API の構築

ブロッキング API とノンブロッキング API には、共通のコンストラクタが 2 つあります。

- 空のコンストラクタ
- パラメータとして LEG 名を受け入れるコンストラクタ

LEG 名を受け入れるコンストラクタ

SM で SM-LEG 障害ハンドリング オプションを有効にするには、LEG 名を設定します。『*Cisco SCMS Subscriber Manager User Guide*』の LEG ソフトウェア コンポーネントと SM-LEG 障害ハンドリングについての説明を参照してください。

LEG 名は、接続障害から回復するときに SM が使用します。LEG 名には、次のように API を識別する文字列定数が付加されます。

- ブロッキング API：**.B.SM-API.J**
- ノンブロッキング API：**.NB.SM-API.J**

例（ブロッキング API）

設定した LEG 名が **my-leg.10.1.12.45-version-1.0** の場合、実際の LEG 名は **my-leg.10.1.12.45-version-1.0.B.SM-API.J** になります。

LEG 名を設定しないと、名前のプレフィックスとして、そのマシンのホスト名が使用されます。

LEG-SM 障害ハンドリングに関する詳細は、『*Cisco SCMS Subscriber Manager User Guide* 』の「Configuration File Options」を参照してください。

ノンブロッキング API では、その他のコンストラクタも使用できます。詳細は、「ノンブロッキング API の構築」（p.4-5）を参照してください。

セットアップ操作

セットアップ操作は、2 つの API で異なります。どちらの API も、「切断リスナ インターフェイス」に説明されている切断リスナの設定をサポートしています。

- ブロッキング API のセットアップ（p.2-4）
- ノンブロッキング API のセットアップ（p.2-4）

ブロッキング API のセットアップ

ブロッキング API をセットアップするには、操作タイムアウト値を設定する必要があります。詳細は、「ブロッキング API」（p.3-1）を参照してください。

ノンブロッキング API のセットアップ

ノンブロッキング API をセットアップするには、切断リスナを設定する必要があります。詳細は、「ノンブロッキング API」（p.4-1）を参照してください。

SM への接続

SM に接続するには、以下のいずれかの **connect** メソッドを使用します。

- 次の方法で、SM への接続にデフォルトの RPC TCP ポート（14374）を使用します。

  ```
  connect(String host)
  ```

- 次の方法で、API の接続に使用する TCP ポートを呼び出し側が設定できます。

  ```
  connect(String host, int port)
  ```

いずれの方法でも、host パラメータは、IP アドレスまたは到達可能なホスト名で指定できます。

isConnected メソッドにより、API の操作中いつでも、API が接続されているかどうかを確認できます。

API の終了

サーバとクライアントの両方のリソースを解放するには、**disconnect** メソッドを使用します。

たとえば、次のようにメイン クラスで **finally** ステートメントを使用することを推奨します。

```
public static void main(String [] args) throws Exception {
SMNonBlockingApi smnbapi = new SMNonBlockingApi();
try {
...
}
finally {
smnbapi.disconnect();
}}
```

サブスクライバ名のフォーマット

どちらの API も、ほとんどのメソッドは入力パラメータとしてサブスクライバ名を必要とします。ここでは、サブスクライバ名のフォーマット ルールについて説明します。

使用できる文字数は最大 64 文字です。ASCII コード 32 ～ 126 の印刷可能な文字が使用できますが、34（"）、39（'）、および 96（`）は除きます。

ネットワーク ID マッピング

ネットワーク ID マッピングは、SCE デバイスが特定のサブスクライバ レコードと関連付けることのできるネットワーク識別子です。たとえば IP アドレスは、ネットワーク ID マッピング（または単純にマッピング）の典型的な例です。現在のところ、Cisco Service Control ソリューションでは、IP アドレス、IP 範囲、VLAN のマッピングがサポートされています。

ブロッキング API とノンブロッキング API のいずれにも、パラメータとしてマッピングを受け入れる操作が含まれています。例：

- The **addSubscriber** 操作（ブロッキング API）
- **login** メソッド（ブロッキング API またはノンブロッキング API）

API メソッドでマッピングを行う場合、呼び出し側は次に示す 2 つのパラメータを要求されます。

- **java.lang.String** マッピング識別子またはマッピング型の配列
- short マッピング型またはマッピング型の配列

配列を渡す場合、**mappingTypes** 配列に含まれる要素の数は、**mappings** 配列の要素数と同じであるか、または 1 つです。**mappingTypes** 配列内の要素が 1 つの場合、すべてのマッピングがこの単一要素と同じ型になります。

API は、次に示すサブスクライバ マッピング型をサポートしています。

- IP アドレスまたは IP 範囲
- VLAN タグ

詳細については、『*Cisco SCMS Subscriber Manager User Guide*』を参照してください。

- IP アドレス マッピングの指定（p.2-5）
- IP 範囲マッピングの指定（p.2-6）
- VLAN タグ マッピングの指定（p.2-6）

IP アドレス マッピングの指定

IP アドレスの文字列フォーマットには、一般的に次のような 10 進表記法が使用されています。

```
IP-Address=[0-255].[0-255].[0-255].[0-255]
```

例

- `216.109.118.66`
- IP アドレスのマッピングの型は、**com.pcube.management.api.SMApiConstants** インターフェイスで指定します。
  - **com.pcube.management.api.SMApiConstants.MAPPING_TYPE_IP** は、マッピング識別子がマッピング識別子配列内の同じインデックスと一致する単一の IP マッピングを指定します。
  - **com.pcube.management.api.SMApiConstants.ALL_IP_MAPPINGS** は、マッピング識別子配列内のすべてのエントリが IP マッピングであることを指定します。

IP 範囲マッピングの指定

IP 範囲の文字列フォーマットは、10 進表記法の IP アドレスおよびビット マスク内の 1 の数を表す 10 進数です。

```
IP-Range=[0-255].[0-255].[0-255].[0-255]/[0-32]
```

例

- **10.1.1.10/32** は、フル マスクの IP 範囲、つまり正規の IP アドレスです。
- **10.1.1.0/24** は、24 ビット マスクの IP 範囲で、**10.1.1.0** から **10.1.1.255** までの範囲のアドレスすべてを表します。

(注) IP 範囲のマッピング型は、IP アドレスのマッピング型と同じです。

VLAN タグ マッピングの指定

VLAN タグ マッピングの文字列フォーマットは VLAN-tag = 0-4095 です。

値は指定範囲の 10 進数です。

マッピングの型は、**com.pcube.management.api.SMApiConstants** インターフェイスで指定します。

- **com.pcube.management.api.SMApiConstants.MAPPING_TYPE_VLAN** は、マッピング識別子がマッピング識別子配列内の同じインデックスと一致する単一の VLAN マッピングを指定します。
- **com.pcube.management.api.SMApiConstants.ALL_VLAN_MAPPINGS** は、マッピング識別子配列内のすべてのエントリが VLAN マッピングであることを指定します。

サブスクライバ ドメイン

The ドメインの詳細については、『*Cisco SCMS Subscriber Manager User Guide*』を参照してください。簡単にいえば、ドメインとは、どの SCE デバイスのサブスクライバ レコードをアップデートするかを SM に伝える識別子です。

ドメイン名は、**String** 型です。システム ドメイン名は、システム導入時にネットワーク管理者が決めるので、導入システムによってさまざまです。API には、サブスクライバが所属するドメインを指定するメソッドやシステムのドメイン名に関するクエリーを許可するメソッドが含まれています。ある API 操作で SM ドメイン リポジトリにないドメイン名が指定された場合、その操作はエラーとみなされ、**RpcErrorException** が戻ります。

SM の自動ドメインローミング機能により、アップデートしたドメイン パラメータを持つサブスクライバ向けの方法を利用して、サブスクライバはドメイン間を移動することができます。

(注) 自動ドメイン ローミングは、サブスクライバのドメイン変更が出来なかった SM API の旧バージョンとの下位互換性はありません。

サブスクライバ プロパティ

サブスクライバ プロパティを扱う操作もいくつかあります。サブスクライバ プロパティは、サブスクライバによって生成されたネットワーク トラフィックに対する SCE の分析や反応に影響する一組のキー値です。

プロパティの詳細については、『*Cisco SCMS Subscriber Manager User Guide*』および『*Cisco Service Control Application for Broadband User Guide*』を参照してください。アプリケーション ユーザ ガイドには、アプリケーション固有の情報が記載されているので、ご使用のシステムで稼働しているアプリケーション内のサブスクライバ プロパティ、許可値、各プロパティ値の重要性を知ることができます。

Java API 操作のサブスクライバ プロパティをフォーマットするには、文字列配列 **propertyKeys** および **propertyValues** を使用します。

(注)　この配列は*同じ長さ*にする必要があり、また NULL エントリは許可されません。キー配列内の各キーには値の配列内に一致するエントリがあります。たとえば、**propertyKeys[j]** の値は **propertyValues[j]** にあります。IP 範囲のマッピング型は IP アドレスのマッピング型と同じです。

例

プロパティ キー配列が **{"packageId","monitor"}** でプロパティ値配列が **{"5","1"}** である場合、プロパティは **packageId=5** , **monitor=1** となります。

カスタム プロパティ

カスタム プロパティを扱う操作もあります。カスタム プロパティはサブスクライバ プロパティと似ていますが、サブスクライバのトラフィックに対する SCE の分析や操作には影響しません。アプリケーション管理モジュールはカスタム プロパティを使用して各サブスクライバに関する追加情報を保存します。

カスタム プロパティをフォーマットするには、「サブスクライバ プロパティ」(p.2-7) のフォーマットと同様に、**文字列**配列の **customPropertyKeys** および **customPropertyValues** を使用します。

切断リスナ インターフェイス

ブロッキングとノンブロッキングのいずれの API も、切断リスナの設定が可能です。切断リスナは、単一のメソッドによるインターフェイスです。

```
public interface DisconnectListener {
/**
* called when the connection with the server is down.
*/
public void connectionIsDown();
}
```

API が SM から切断されたときに通知を受けるようにするには、このインターフェイスを実装します。

切断リスナの設定には、**setDisconnectListener** メソッドを使用します。

DisconnectListener インターフェイス例

次の例では、**stdout** にメッセージを出力して終了する簡単な切断リスナの実装です。

```
import com.pcube.management.framework.rpc.DisconnectListener;
public class MyDisconnectListener implements DisconnectListener {
public void connectionIsDown(){
System.out.println("Message: connection is down.");
System.exit(0);
}
}
```

例外

SM Java API のすべての関数エラーは、同じ Java クラスの **com.pcube.management.framework.rpc.RpcErrorException** で提供されます。これは、通常の Java の使い方とは対照的です。このような「対照的」なアプローチが採用されたのは、SM API の「言語横断的」な特性のためです。このようなアプローチを採ることによって、SM API の実装（Java、C、C++）は、概観上の統一性を保つことができます。

各例外は、次の情報を提供します。

- 固有のエラー コード（**long** ）
- 情報メッセージ（**java.lang.String** ）
- サーバ側スタック トレース（**java.lang.String** ）

エラー コードは、**com.pcube.management.api.SMApiConstants** を使用して解釈できます。エラー コードとその重要性についての詳細は、「エラー コードのリスト」（p.A-1）を参照してください。

（注） ブロッキング API 使用時**のみ**に発生するエラーもいくつかあります。また、操作タイムアウト処理に関連した操作エラーもあります。これらについての詳細は、「ブロッキング API」（p.3-1）を参照してください。

実用的なヒント

API とアプリケーションを統合するコードを実装する場合、次の実用的なヒントを考慮する必要があります。

- SM に接続したら、API を何度も使用して SM へのオープンな API 接続を常時維持します。接続を確立することは、SM 側と API クライアント側にリソースを割り当てるタイムリーな手順です。
- スレッドの間で API 接続を共有します。LEG ごとに 1 つの接続を持つことを推奨します。複数の接続では、SM 側とクライアント側でより多くのリソースを必要とします。
- API に対するコールの同期化を実行しないでください。クライアントが API に対するコールを自動的に同期化します。
- API クライアント（LEG）を SM マシン プロセッサ番号と同じ順で配置することを推奨します。
- LEG アプリケーションにログオン操作のバーストがある場合、このバーストを保持するため内部バッファ サイズを適宜拡大します（ノンブロッキング式）。
- 統合中、問題が発生したときに統合をトラブルシューティングするため、SM ログ内に API 操作を表示するよう SM **logon_logging_enabled** 設定パラメータを設定します。
- ノンブロッキング操作の戻り値をログ / 出力する LEG アプリケーションのデバッグ モードを使用します。
- SM への接続の復元力を改善するには、自動再接続機能を使用します。
- クラスタのセットアップでは、1 つのマシンの管理 IP アドレスではなく、クラスタの仮想 IP アドレスを使用して API を接続します。

■ 実用的なヒント

CHAPTER 3

ブロッキング API

この章では、ブロッキング API 固有の機能である応答タイムアウトについて説明します。

さらに、ブロッキング API のすべての操作を紹介し、コードの例も示します。

(注) *自動統合*の開発のみを行う場合は、この章を飛ばして、第 4 章「ノンブロッキング API」に進んでください。

マルチスレッドのサポート

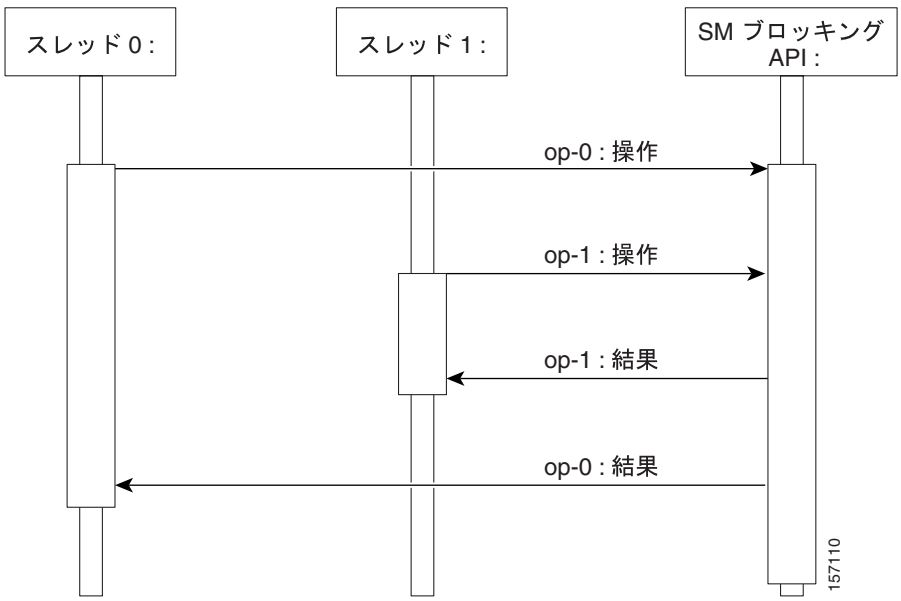
ブロッキング API では、メソッドを同時に呼び出すことのできるスレッド数に制限がありません。

(注)　ブロッキング API でマルチスレッドにする場合、呼び出しの順序は保証**されません**。

例

スレッド -0 が操作 -0 をタイム -0 で呼び出し、スレッド -1 が操作 -1 をタイム -1 で呼び出します。タイム -1 はタイム -0 よりあとです。この例では、次の図（縦が時間）のように操作 -1 が操作 -0 の**前に**実行される可能性があります。

図 3-1　マルチスレッドのサポート



SM は各 API インターフェイスを処理するために 5 つのスレッドを割り当てます。5 つのスレッド順のスレッド番号のある API を使用するマルチスレッド アプリケーションを開発することを推奨します。より多くのスレッドを実装すると、スレッドの呼び出し遅延が長くなる場合があります。

応答タイムアウトと操作タイムアウトの例外

ブロッキング操作が戻るのは、SM から操作結果が取得されたときだけです。ネットワークの異常やその他のエラーによって操作結果を取得できない場合、呼び出し側はいつまでも待機することになります。SM API には、こうした状況を避ける手段が用意されています。

呼び出し側は、応答タイムアウト機能（**setReplyTimeout** メソッド）を使用してタイムアウトを設定できます。タイムアウト期間内に応答が戻らないと、**com.pcube.management.framework.rpc.OperationTimeoutException** が発生します。

応答タイムアウトを設定するには、long 値を指定して **setReplyTimeout** メソッドを呼び出します。応答タイムアウトはミリ秒単位です。ゼロの値を指定すると、結果が届くまで操作が待機（フリーズまたは停止）状態になります。つまり、結果が届かなければ永久に待機状態のままになります。

結果が戻るまで待機状態になって、呼び出し側をブロックしているメソッド呼び出しを別の方法で解放することも可能です。呼び出しているスレッドから **interrupt** メソッドを呼び出します。呼び出し側に **java.lang.InterruptedException** が戻ります。

ブロッキング API メソッド

ここでは、ブロッキング API のメソッドについて順番に説明します。各メソッドの構文のあとに、その入力パラメータと戻り値を記載します。

ブロッキング API は、ノンブロッキング API の上位集合です。戻り値と結果の処理が異なる点を除けば、同じ操作の機能と構文構造はどちらの API でも同じです。

メソッドはどれも、SM との接続が確立される前に呼び出されると、**java.lang.IllegalStateException** を生じます。

ブロッキング API メソッドは、次のカテゴリに分類されます。

- **IP およびプロパティの動的割り当て** ― AAA システムとの統合に SM API を使用する場合は、次のようなメソッドを使用します。これらのメソッドは、データベースに対するサブスクライバの追加や削除ではなく、既存のサブスクライバの動的パラメータ（IP アドレスなど）の変更に使用されます。
  - ― login（p.3-5）
  - ― logoutByName（p.3-8）
  - ― logoutByNameFromDomain（p.3-9）
  - ― logoutByMapping（p.3-10）
  - ― loginCable（p.3-11）
  - ― logoutCable（p.3-13）
- **サブスクライバの静的 / 手動設定** ― GUI を利用する場合は、次のメソッドを使用します。
  - ― addSubscriber（p.3-14）
  - ― removeSubscriber（p.3-16）
  - ― removeAllSubscribers（p.3-17）
  - ― setPropertiesToDefault（p.3-27）
  - ― removeCustomProperties（p.3-28）
- サブスクライバ認識モードで独立して実行される単純な読み取りのみの操作の場合は、次のメソッドを使用します。
  - ― getNumberOfSubscribers（p.3-18）
  - ― getNumberOfSubscribersInDomain（p.3-18）
  - ― getSubscriber（p.3-19）
  - ― subscriberExists（p.3-20）
  - ― subscriberLoggedIn（p.3-21）
  - ― getSubscriberNameByMapping（p.3-22）
  - ― getSubscriberNames（p.3-23）
  - ― getSubscriberNamesInDomain（p.3-24）
  - ― getSubscriberNamesWithPrefix（p.3-25）
  - ― getSubscriberNamesWithSuffix（p.3-26）
  - ― getDomains（p.3-27）

異なるカテゴリのメソッドを 1 つのアプリケーションに組み合わせて使用できます。この分類は、メソッドの分類のみを目的としたものです。

- login（p.3-5）
- logoutByName（p.3-8）
- logoutByNameFromDomain（p.3-9）
- logoutByMapping（p.3-10）

login

構文

```
public void login(String subscriberName,
String[] mappings,
short[] mappingTypes,
String[] propertyKeys,
String[] propertyValues,
String domain,
boolean isMappingAdditive,
int autoLogoutTime)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

説明

The **login** メソッドは、SM データベース内にすでに存在しているサブスクライバのドメインおよびマッピング（場合によってはプロパティ）の追加や変更を実行します。このメソッドは部分データで呼び出すことができます。たとえば、マッピングだけ、またはプロパティだけを与えたり、未変更フィールドに NULL を入力することが可能です。

同じドメイン内に同じ（つまり衝突する）マッピングを持つ別のサブスクライバがすでに存在している場合、既存のサブスクライバから衝突しているマッピングが削除され、新しいサブスクライバにそのマッピングが割り当てられます。

指定したサブスクライバが SM データベース内に存在しない場合は、与えられたデータでサブスクライバが作成されます。

パラメータ

subscriberName ― 「サブスクライバ名のフォーマット」（p.2-4）の説明を参照してください。

mappings ― 「ネットワーク ID マッピング」（p.2-5）のマッピングおよびマッピング型に関する説明を参照してください。マッピングを指定せず、**isMappingAdditive** フラグが TRUE の場合、前のマッピングが保持されます。そのようなマッピングがない場合、操作はエラーとなります。

mappingTypes ― 「ネットワーク ID マッピング」（p.2-5）のマッピングおよびマッピング型に関する説明を参照してください。

propertyKeys ― 「サブスクライバ プロパティ」（p.2-7）のプロパティのキーおよび値に関する説明を参照してください。

propertyValues ― 「サブスクライバ プロパティ」（p.2-7）のプロパティのキーおよび値に関する説明を参照してください。

domain ― 「サブスクライバ ドメイン」（p.2-6）の説明を参照してください。

domain が NULL であっても、そのサブスクライバにすでにドメインがある場合は、既存のドメインが保持されます。

ドメインが、サブスクライバに事前に割り当てられたドメインと異なる場合、サブスクライバは事前に割り当てられたドメインの SCE から自動的に削除され、新しいドメインの SCE へ移動されます。

`isMappingAdditive`

- **TRUE** ― この呼び出しに与えられたマッピングをサブスクライバ レコードに追加します。
- **FALSE** ― この呼び出しに与えられたマッピングで、サブスクライバ レコードの既存のマッピングを上書きします。

autoLogoutTime ― このメソッドに引数として与えられたマッピングのみに適用されます。

- 正の値（N）― N 秒後に自動的にマッピングのログアウトを実行します（logout メソッドの呼び出しに似ています）。
- 0 の値 ― 指定されたマッピングのその時点での有効時間が維持されます。
- 負の値 ― 指定されたマッピングに設定されている有効時間を無効にします。

RPC 例外エラー コード

このメソッドで戻る可能性のあるエラー コードは次のとおりです。

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE_BAD_SUBSCRIBER_MAPPING
- ERROR_CODE_SUBSCRIBER_DOMAIN_ASSOCIATION
- ERROR_CODE_DATABASE_EXCEPTION
- ERROR_CODE_UNKNOWN

  このエラーは次の場合に発生する可能性があります。

  ― 存在しないサブスクライバまたはドメインのないサブスクライバのドメイン　パラメータが NULL 値の場合

  ― propertyValues パラメータの値が無効である場合

エラー コードの説明については、「エラー コードのリスト」（p.A-1）を参照してください。

戻り値

なし

例

john という名前の既存のサブスクライバに IP アドレス 192.168.12.5 を追加し、既存のマッピングを変更しない場合は、次のようにします。

```
login(
"john",                          // subscriber name
new String[]{"192.168.12.5"},
SMApiConstants.ALL_IP_MAPPINGS,
null, null,
"subscribers",                   // domain
true,                            // isMappingAdditive is true
-1);                             // autoLogoutTime set to infinite
```

IP アドレス 192.168.12.5 を追加して、今までのマッピングを上書きするには、次のようにします。

```
login(
"john",                          // subscriber name
new String[]{"192.168.12.5"},
SMApiConstants.ALL_IP_MAPPINGS,
null, null,
"subscribers",                   // domain
false,                           // isMappingAdditive is false
-1);                             // autoLogoutTime set to infinite
```

以前 *john* へ割り当てられた 192.168.12.5 の自動ログアウト時間を延長するには、次にようにします。

```
login(
"john",                          //the previously assigned IP
new String[]{"192.168.12.5"},
SMApiConstants.ALL_IP_MAPPINGS,
null, null,
"subscribers",                   // domain
false,                           // isMappingAdditive
300);                            // autoLogoutTime set to 300 seconds
```

john の動的プロパティ（package ID など）を変更するには、次のようにします。

```
login(
"john",
null, null,
new String[]{"packageId"}, // property key
new String[]{"10"},     // property value
"subscribers",          // domain
false, -1);
```

john という名前の既存のサブスクライバに IP アドレス 192.168.12.5 を追加し、既存のマッピングは変更せず、*john* の動的プロパティ（package ID など）を変更するには、次のようにします。

```
login(
"john",
new String[]{"192.168.12.5"},
SMApiConstants.ALL_IP_MAPPINGS,
new String[]{"packageId"}, // property key
new String[]{"10"}, // property value
"subscribers", // domain
true,       // isMappingAdditive is set to true
-1);
```

logoutByName

構文

```
public boolean logoutByName(String subscriberName,
String[] mappings,
short[] mappingTypes)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

説明

データベース内でサブスクライバを検索し、マッピングを削除します。指定したサブスクライバが存在しなければ、何も実行されません。

パラメータ

subscriberName ― 「サブスクライバ名のフォーマット」（p.2-4）の説明を参照してください。

mappings ― 「ネットワーク ID マッピング」（p.2-5）のマッピングおよびマッピング型に関する説明を参照してください。マッピングを指定しないと、そのサブスクライバのすべてのマッピングが削除されます。

mappingTypes ― 「ネットワーク ID マッピング」（p.2-5）のマッピングおよびマッピング型に関する説明を参照してください。

戻り値

- TRUE ― サブスクライバが見つかり、そのサブスクライバのマッピングがサブスクライバ データベースから削除された場合
- FALSE ― サブスクライバ データベースでそのサブスクライバが見つからなかった場合

RPC 例外エラー コード

このメソッドで戻る可能性のあるエラー コードは次のとおりです。

- ERROR_CODE_SUBSCRIBER_DOES_NOT_EXIST
- ERROR_CODE_BAD_SUBSCRIBER_MAPPING
- ERROR_CODE_SUBSCRIBER_DOMAIN_ASSOCIATION
- ERROR_CODE_DOMAIN_NOT_FOUND
- ERROR_CODE_NOT_A_SUBSCRIBER_DOMAIN
- ERROR_CODE_DATABASE_EXCEPTION

エラー コードの説明については、「エラー コードのリスト」（p.A-1）を参照してください。

例

サブスクライバ *john* から IP アドレス 192.168.12.5 を削除するには、次のようにします。

```
boolean isExist = logoutByName(
"john",
new String[]{"192.168.12.5"},
SMApiConstants.ALL_IP_MAPPINGS);
```

サブスクライバ *john* のすべての IP アドレスを削除するには、次のようにします。

```
boolean isExist = logoutByName("john", null, null);
```

logoutByNameFromDomain

構文

```
public boolean logoutByNameFromDomain(String subscriberName,
String[] mappings,
short[] mappingTypes,
String domain)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

説明

logoutByName と似ていますが、呼び出し側はサブスクライバが所属するドメイン名も指定できます。サブスクライバのドメインがわかっている場合は、このメソッドを使うとパフォーマンスが向上します。

パラメータ

subscriberName ― 「サブスクライバ名のフォーマット」（p.2-4）の説明を参照してください。

mappings ― 「ネットワーク ID マッピング」（p.2-5）のマッピングおよびマッピング型に関する説明を参照してください。マッピングを指定しないと、そのサブスクライバのすべてのマッピングが削除されます。

mappingTypes ― 「ネットワーク ID マッピング」（p.2-5）のマッピングおよびマッピング型に関する説明を参照してください。

domain ― 「サブスクライバ ドメイン」（p.2-6）の説明を参照してください。

次の条件の *いずれか* に該当する場合、操作はエラーとなります。

- ドメインが NULL であるが、そのサブスクライバはデータベース内に存在し、ドメインに所属している場合
- 指定されたドメインが間違っている場合

戻り値

- TRUE ― そのサブスクライバが見つかり、サブスクライバ データベースから削除された場合
- FALSE ― サブスクライバ データベースでそのサブスクライバが見つからなかった場合

RPC 例外エラー コード

このメソッドで戻る可能性のあるエラー コードは次のとおりです。

- ERROR_CODE_SUBSCRIBER_DOES_NOT_EXIST
- ERROR_CODE_BAD_SUBSCRIBER_MAPPING
- ERROR_CODE_SUBSCRIBER_DOMAIN_ASSOCIATION
- ERROR_CODE_DOMAIN_NOT_FOUND
- ERROR_CODE_NOT_A_SUBSCRIBER_DOMAIN
- ERROR_CODE_DATABASE_EXCEPTION

エラー コードの説明については、「エラー コードのリスト」（p.A-1）を参照してください。

例

ドメイン *subscribers* から、サブスクライバ *john* の IP アドレス 192.168.12.5 を削除するには、次のようにします。

```
boolean isExist = logoutByNameFromDomain(
"john",
new String[]{"192.168.12.5"},
SMApiConstants.ALL_IP_MAPPINGS,
"subscribers");
boolean isExist = logoutByNameFromDomain(
"john",
null,
null,
"subscribers");
```

logoutByMapping

構文

```
public boolean logoutByMapping(String mapping,
short mappingType,
String domain)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

説明

ドメインとマッピングに基づいてサブスクライバを探し、サブスクライバがデータベース内にあれば、そのマッピングを削除します。

パラメータ

mappings ― 「ネットワーク ID マッピング」（p.2-5）のマッピングおよびマッピング型に関する説明を参照してください。

mappingTypes ― 「ネットワーク ID マッピング」（p.2-5）のマッピングおよびマッピング型に関する説明を参照してください。

ドメイン ― logoutByNameFromDomain 操作については、「パラメータ」（p.3-9）を参照してください。

戻り値

- TRUE ― そのサブスクライバが見つかり、サブスクライバ データベースから削除された場合
- FALSE ― サブスクライバ データベースでそのサブスクライバが見つからなかった場合

RPC 例外エラー コード

このメソッドで戻る可能性のあるエラー コードは次のとおりです。

- ERROR_CODE_SUBSCRIBER_DOES_NOT_EXIST
- ERROR_CODE_BAD_SUBSCRIBER_MAPPING
- ERROR_CODE_SUBSCRIBER_DOMAIN_ASSOCIATION
- ERROR_CODE_DOMAIN_NOT_FOUND
- ERROR_CODE_NOT_A_SUBSCRIBER_DOMAIN
- ERROR_CODE_DATABASE_EXCEPTION

エラー コードの説明については、「エラー コードのリスト」（p.A-1）を参照してください。

例

ドメイン *subscribers* から IP アドレス 192.168.12.5 を削除するには、次のようにします。

```
boolean isExist = logoutByMapping(
"192.168.12.5",
SMApiConstants. MAPPING_TYPE_IP,
"subscribers");
```

loginCable

構文

```
public void loginCable(String CPE,
String CM,
String IP,
int lease,
String domain,
String[] propertyKeys,
String[] propertyValues)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

説明

ケーブル環境に適応した login メソッド（SM 内のケーブル対応モジュールへの呼び出し）。このメソッドは CPE および CM の SM へのログイン用に設計されています。CPE ログインの場合は、CM 引数に CM MAC を、CPE 引数に CPE MAC を指定します。CM のログインでは、CPE と CM の両方の引数に CM MAC アドレスを指定します。CM が SM データベースに存在していない CPE のログインは無視されます。インポートまたは CM login 操作のいずれかによって、あらかじめ CM がデータベース内に存在している必要があります。詳細については、『*Cisco SCMS Subscriber Manager User Guide* 』の「CPE as Subscriber in Cable Environment」の章を参照してください。

（注） SM データベース内の CPE の名前は、CPE と CM の値を 2 つの下線（_）で連結したものとなります。呼び出し側で CPE と CM の値の長さが合計 **38** 文字を超えていないことを必ず確認してください。

パラメータ

CPE ― CPE 固有の識別子（通常、MAC アドレス）

CM ― ケーブル モデム固有の識別子（通常、MAC アドレス）

IP ― CPE の IP アドレス

lease ― CPE のリース時間

domain ―「サブスクライバ ドメイン」（p.2-6）の説明を参照してください。ドメインは通常、CMTS IP になります。

（注） CMT SIP がドメイン名として正しく解釈されるためには、SM にドメインの別名を設定する必要があります。エイリアスの設定については、『*Cisco SCMS Subscriber Manager User Guide* 』の「Configuring Domains」の章を参照してください。

propertyKeys ―「サブスクライバ プロパティ」（p.2-7）のプロパティのキーおよび値に関する説明を参照してください。

CPE のアプリケーション プロパティが部分的に指定されたり、まったく指定されない場合は、失われたプロパティの値は、その CPE が属している CM のアプリケーション プロパティからコピーされます。CM の各アプリケーション プロパティは、そこに属す CPE のデフォルト値として使用されます。

propertyValues ―「サブスクライバ プロパティ」（p.2-7）のプロパティのキーおよび値に関する説明を参照してください。

戻り値

なし

RPC 例外エラー コード

なし

例

CM1 という名前の CM に IP アドレス 192.168.12.5 を追加し、リース時間を 2 時間にするには、次のようにします。

```
loginCable(
"CM1",
"CM1",
"192.168.12.5",
7200,          // lease time in seconds
"subscribers", null, null);
```

CM1 にある *CPE1* という名前の CPE に IP アドレス 192.168.12.50 を追加し、リース時間を 1 時間にするには、次のようにします。

```
loginCable(
"CPE1",
"CM1",
"192.168.12.50",
3600,          // lease time in seconds
"subscribers", null, null);
```

logoutCable

構文

```
public boolean logoutCable(String CPE,
String CM,
String IP,
String domain)
```

説明

SM ケーブル対応モジュールにログアウト イベント（CPE のオフライン）を示します。

パラメータ

CPE ― 説明については、loginCable メソッドの項「パラメータ」（p.3-12）を参照してください。

CM ― 説明については、loginCable メソッドの項「パラメータ」（p.3-12）を参照してください。

IP ― 説明については、loginCable メソッドの項「パラメータ」（p.3-12）を参照してください。

domain ― 説明については、loginCable メソッドの項「パラメータ」（p.3-12）を参照してください。

戻り値

- TRUE ― その CPE が見つかり、サブスクライバ データベースから削除された場合
- FALSE ― サブスクライバ データベースでその CPE が見つからなかった場合

RPC 例外エラー コード

なし

例

CM1 に属す *CPE1* から IP アドレス 192.168.12.5 を削除するには、次のようにします。

```
boolean isExist = logoutCable(
"CPE1",
"CM1",
"192.168.12.5",
"subscribers");
```

addSubscriber

構文

```
public void addSubscriber(String subscriberName,
String[] mappings,
short[] mappingTypes,
String[] propertyKeys,
String[] propertyValues,
String[] customPropertyKeys,
String[] customPropertyValues,
String domain)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

説明

メソッド パラメータに準じて新規サブスクライバのレコードを作成し、SM データベースにレコードを追加します。

この名前のサブスクライバがすでに存在する場合、そのサブスクライバは新しいサブスクライバが追加される前に削除されます。**login** では指定フィールドが変更され、未指定フィールドは変更されませんが、**addSubscriber** では、渡されたパラメータによって、指定どおりにサブスクライバが設定されます。

(注) 既存のサブスクライバには、**addSubscriber** ではなく **login** メソッドを呼び出すことを推奨します。動的なマッピングおよびプロパティは **login** を使用して設定する必要があります。静的なマッピングおよびプロパティは、**addSubscriber** を使用してサブスクライバの初回作成時に設定します。

(注) **addSubscriber** では、auto-logout 機能は常に無効になります。auto-logout を有効にするには、**login** を使用します。

例

サブスクライバ データベース内にすでにセットアップされているサブスクライバ *AB* には、*IP1* を保持します。

AB の **addSubscriber** 操作がマッピングの指定なしに呼び出される場合（**mappings** および **mappingTypes** フィールドのいずれでも NULL）、*AB* はマッピングされません。

ただし、これらの NULL 値パラメータの **login** 操作を呼び出しても、*AB* のマッピングは変更しません。*AB* は *IP1* の前の IP マッピングを保持します。

パラメータ

subscriberName ― 「サブスクライバ名のフォーマット」（p.2-4）の説明を参照してください。

mappings ― 「ネットワーク ID マッピング」（p.2-5）のマッピングおよびマッピング型に関する説明を参照してください。

mappingTypes ― 「ネットワーク ID マッピング」（p.2-5）のマッピングおよびマッピング型に関する説明を参照してください。

propertyKeys ― 「サブスクライバ プロパティ」（p.2-7）のプロパティのキーおよび値に関する説明を参照してください。

propertyValues ― 「サブスクライバ プロパティ」（p.2-7）のプロパティのキーおよび値に関する説明を参照してください。

customPropertyKeys ― 「カスタム プロパティ」（p.2-7）のプロパティのキーおよび値に関する説明を参照してください。

customPropertyValues ― 「カスタム プロパティ」（p.2-7）のプロパティのキーおよび値に関する説明を参照してください。

domain ― 「サブスクライバ ドメイン」（p.2-6）の説明を参照してください。

NULL 値はサブスクライバにドメインがないことを示します。

domain が NULL であるが、そのサブスクライバには既にドメインがある場合、既存のドメインが保持されます。

戻り値

なし

RPC 例外エラー コード

このメソッドで返される可能性のあるエラー コードは次のとおりです。

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE_BAD_SUBSCRIBER_MAPPING
- ERROR_CODE_DOMAIN_NOT_FOUND
- ERROR_CODE_SUBSCRIBER_ALREADY_EXISTS
- ERROR_CODE_SUBSCRIBER_DOMAIN_ASSOCIATION
- ERROR_CODE_UNKNOWN

  このエラー コードは、**propertyValues** パラメータに無効な値が指定されたことを示しています。
- ERROR_CODE_DATABASE_EXCEPTION

エラー コードの説明については、「エラー コードのリスト」（p.A-1）を参照してください。

例

カスタム プロパティをいくつか持つ新しいサブスクライバ、*john* を追加するには、次のようにします。

```
addSubscriber("john",
null, null,    // dynamic mappings will be set by login
null, null      // dynamic properties will be set by login
new String[]{  // custom property keys
        "work phone",
        "home phone"},
new String[]{  // custom property values
        "6543212"
        "5059927"},
"subscribers");// default domain
```

removeSubscriber

構文

```
public boolean removeSubscriber(String subscriberName)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

説明

SM データベースから 1 つのサブスクライバを完全に削除します。

パラメータ

subscriberName ― 「サブスクライバ名のフォーマット」（p.2-4）の説明を参照してください。

戻り値

- TRUE ― データベースでそのサブスクライバが見つかり、正常に削除された場合
- FALSE ― TRUE の条件が満たされなかった場合。つまり、データベースでそのサブスクライバが見つからなかったか、または見つかったが正常に削除されなかった場合

RPC 例外エラー コード

このメソッドで返される可能性のあるエラー コードは次のとおりです。

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE_SUBSCRIBER_DOES_NOT_EXIST
- ERROR_CODE_DATABASE_EXCEPTION

エラー コードの説明については、「エラー コードのリスト」（p.A-1）を参照してください。

例

サブスクライバ *john* をデータベースから完全に削除するには、次のようにします。

```
boolean isExist = removeSubscriber("john");
```

removeAllSubscribers

構文

```
public void removeAllSubscribers()
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

説明

SM からすべてのサブスクライバを削除し、データベースにサブスクライバが 1 つもない状態にします。

（注） このメソッドは実行に時間がかかる場合があります。操作タイムアウトの例外条件が発生しないようにするため、このメソッドを呼び出す前に操作タイムアウトを大きな値（最大 5 分）に設定してください。

戻り値

なし

RPC 例外エラー コード

なし

getNumberOfSubscribers

構文

```
public int getNumberOfSubscribers()
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

説明

SM データベース内のサブスクライバの合計数を取得します。

戻り値

SM 内のサブスクライバ数

RPC 例外エラー コード

なし

getNumberOfSubscribersInDomain

構文

```
public int getNumberOfSubscribersInDomain(String domain)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

説明

ある 1 つのサブスクライバ ドメイン内のサブスクライバ数を取得します。

パラメータ

domain ― SM のドメイン リポジトリにあるサブスクライバ ドメインの名前

戻り値

指定されたドメイン内のサブスクライバ数

RPC 例外エラー コード

このメソッドで戻る可能性のあるエラー コードは次のとおりです。

- ERROR_CODE_NOT_A_SUBSCRIBER_DOMAIN
- ERROR_CODE_DOMAIN_NOT_FOUND

エラー コードの説明については、「エラー コードのリスト」（p.A-1）を参照してください。

getSubscriber

構文

```
public Object[] getSubscriber(String subscriberName)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

説明

サブスクライバ レコードを取得します。各フィールドは、整数、文字列、文字列配列としてフォーマットされます（このメソッドの「戻り値」を参照）。

SM データベース内にそのサブスクライバが存在しない場合は、例外が発生します。

パラメータ

subscriberName ― 「サブスクライバ名のフォーマット」（p.2-4）の説明を参照してください。

戻り値

9 つの要素を持つオブジェクト配列。次の表にインデックス値を示します。NULL 値となる配列要素はありません。

インデックス 0	サバスクライバ名（**java.lang.String**）
インデックス 1	マッピングの配列（**java.lang.String[]**）
インデックス 2	マッピング型の配列（**short[]**）

インデックス 3	ドメイン名（**java.lang.String**）
インデックス 4	プロパティ名の配列（**java.lang.String[]**）
インデックス 5	プロパティ値の配列（**java.lang.String[]**）
インデックス 6	カスタム プロパティ名の配列（**java.lang.String[]**）
インデックス 7	カスタム プロパティ値の配列（**java.lang.String[]**）
インデックス 8	その時点からの秒数で表される自動ログアウト時間 （設定されていない場合は -1）（**long[]**）

RPC 例外エラー コード

このメソッドで戻る可能性のあるエラー コードは次のとおりです。

- ERROR_CODE_SUBSCRIBER_DOES_NOT_EXIST
- ERROR_CODE_DATABASE_EXCEPTION

エラー コードの説明については、「エラー コードのリスト」（p.A-1）を参照してください。

例

john のサブスクライバ レコードを取得するには、次のようにします。

```
Object[] subRecord = getSubscriber("john");
String[] mappings = (String[])subRecord[1]
short[] mappingTypes = {short[])subRecord[2];
String domainName = (String)subRecord[3];
String[] propertyNames = (String[])subRecord[4];
String[] propertyValues = (String[])subRecord[5];
String[] customPropertyName = (String[])subRecord[6];
String[] customPropertyValues = (String[])subRecord[7];
long[] autoLogoutTime = (long[])subRecord[8];
```

subscriberExists

構文

```
public boolean subscriberExists(String subscriberName)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

説明

あるサブスクライバが SM データベース内に存在していることを確認します。

パラメータ

subscriberName ― 「サブスクライバ名のフォーマット」（p.2-4）の説明を参照してください。

戻り値

- TRUE ― SM データベースでそのサブスクライバが見つかった場合
- FALSE ― そのサブスクライバが見つからなかった場合

RPC 例外エラー コード

なし

subscriberLoggedIn

構文

```
public boolean subscriberLoggedIn(String subscriberName)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

説明

SM データベース内の既存のサブスクライバがログインしているかどうかを確認します。つまり、そのサブスクライバが SCE データベースに存在しているかどうかを確認します。

SM が Pull モードで機能するように設定されている場合、このメソッドで TRUE の値が戻っても、そのサブスクライバが実際に SCE データベース内に存在しているとは**限りません**。確かなのは、必要に応じてそのサブスクライバに SCE による pull が実行可能であることだけです。

SM データベース内にそのサブスクライバが存在しない場合は、例外が発生します。

パラメータ

subscriberName ― 「サブスクライバ名のフォーマット」（p.2-4）の説明を参照してください。

戻り値

- TRUE ― そのサブスクライバがログインしている場合
- FALSE ― そのサブスクライバがログインしていない場合

RPC 例外エラー コード

このメソッドで戻る可能性のあるエラー コードは次のとおりです。

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE_DATABASE_EXCEPTION

エラー コードの説明については、「エラー コードのリスト」（p.A-1）を参照してください。

getSubscriberNameByMapping

構文

```
public String getSubscriberNameByMapping(String mapping,
short mappingType,
String domain)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

説明

マッピングおよびドメインに基づいてサブスクライバを検索します。

パラメータ

mappings ― 「ネットワーク ID マッピング」（p.2-5）のマッピングおよびマッピング型に関する説明を参照してください。

mappingTypes ― 「ネットワーク ID マッピング」（p.2-5）のマッピングおよびマッピング型に関する説明を参照してください。

domain ― そのサブスクライバが属しているドメインの名前。次の条件のいずれかに該当する場合、操作はエラーとなります。

- ドメインが NULL であるが、そのサブスクライバはデータベース内に存在し、ドメインに所属している場合
- 指定されたドメインが間違っている場合

戻り値

- サブスクライバ名 ― サブスクライバ レコードが見つかった場合
- NULL ― サブスクライバ レコードが見つからなかった場合

RPC 例外エラー コード

このメソッドで戻る可能性のあるエラー コードは次のとおりです。

- ERROR_CODE_DOMAIN_NOT_FOUND
- ERROR_CODE_BAD_SUBSCRIBER_MAPPING
- ERROR_CODE_NOT_A_SUBSCRIBER_DOMAIN
- ERROR_CODE_DATABASE_EXCEPTION

エラー コードの説明については、「エラー コードのリスト」（p.A-1）を参照してください。

getSubscriberNames

構文

```
public String[] getSubscriberNames(String lastBulkEnd,
int numOfSubscribers)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

説明

SM データベースから、ある 1 まとまり（バルク）のサブスクライバ名を取得します。**lastBulkEnd** から始まり、アルファベット順に **numOfSubscribers** だけサブスクライバが取得されます。

lastBulkEnd が NULL の場合、SM データベース内に存在するサブスクライバ名のうちアルファベット順で最初のサブスクライバ名が使用されます。

（注） **getNumOfSubscribers** からの戻り値が必ず（全バルクの）サブスクライバの合計数であるとは**限りません**。たとえば、取得中にいくつかのサブスクライバの追加や削除が実行された場合は合計数と異なる可能性があります。

パラメータ

lastBulkEnd ― 最後のバルクの最後のサブスクライバ名。アルファベット順で最初のサブスクライバから開始する場合は、NULL を使用します。

numOfSubscribers ― 取得するサブスクライバの数を限定します。この値が SM の制限値（1000）より大きい場合は、SM の制限値が使用されます。

（注） このパラメータに 500 を超える値を指定することは**推奨できません**。

戻り値

アルファベット順のサブスクライバ名の配列

このメソッドは、要求されたサブスクライバから開始して、SM データベース内で見つかったすべてのサブスクライバを戻します。配列のサイズは、**numOfSubscribers** と SM 制限値（1000）のうち小さい方の値に制限されます。

RPC 例外エラー コード

このメソッドで返される可能性のあるエラー コードは次のとおりです。

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE_DATABASE_EXCEPTION

エラー コードの説明については、「エラー コードのリスト」（p.A-1）を参照してください。

例

```
boolean hasMoreSubscribers;
String lastBulkEnd = null;
int bulkSize = 100;
do {
String[] subscribers = smApi.getSubscriberNames(lastBulkEnd, bulkSize);
hasMoreSubscribers = false;
if (subscribers != null) {
for (int i = 0; i <subscribers.length; i++) {
// do something with subscribers[i]
}
if (subscribers.length == bulkSize) {
hasMoreSubscribers = true;
lastBulkEnd = subscribers[bulkSize - 1];
}
}
} while (hasMoreSubscribers);
```

getSubscriberNamesInDomain

構文

```
public String[] getSubscriberNamesInDomain(String lastBulkEnd,
int numOfSubscribers,
String domain)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

説明

指定されたドメインに関連付けられているサブスクライバを SM データベースから取得します。

この操作の意味は、「getSubscriberNames」（p.3-23）操作の意味と同じです。

パラメータ

lastBulkEnd ― getSusbcriberNames 操作の「パラメータ」（p.3-23）の項を参照してください。

numOfSubscribers ― getSusbcriberNames 操作の「パラメータ」（p.3-23）の項を参照してください。

domain ― SM のドメイン リポジトリにあるサブスクライバ ドメインの名前

戻り値

指定されたドメインに属しているサブスクライバ名のアルファベット順配列

getSubscriberNames 操作の「戻り値」（p.3-23）の項を参照してください。

RPC 例外エラー コード

このメソッドで戻る可能性のあるエラー コードは次のとおりです。

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE_DOMAIN_NOT_FOUND
- ERROR_CODE_DATABASE_EXCEPTION

エラー コードの説明については、「エラー コードのリスト」（p.A-1）を参照してください。

getSubscriberNamesWithPrefix

構文

```
public String[] getSubscriberNamesWithPrefix(String lastBulkEnd,
int numOfSubscribers,
String prefix)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

説明

指定されたプレフィックスから名前が始まるサブスクライバを SM データベースから取得します。

この操作の意味は、「getSubscriberNames」（p.3-23）操作の意味と同じです。

パラメータ

lastBulkEnd ― getSusbcriberNames 操作の「パラメータ」（p.3-23）の項を参照してください。

numOfSubscribers ― getSusbcriberNames 操作の「パラメータ」（p.3-23）の項を参照してください。

prefix ― 要求対象のサブスクライバ名のプレフィックスを示す文字列（大文字と小文字は区別されます）

戻り値

要求されたプレフィックスから始まるサブスクライバ名のアルファベット順配列

getSubscriberNames 操作の「戻り値」（p.3-23）の項を参照してください。

RPC 例外エラー コード

このメソッドで戻る可能性のあるエラー コードは次のとおりです。

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE_DATABASE_EXCEPTION

エラー コードの説明については、「エラー コードのリスト」（p.A-1）を参照してください。

getSubscriberNamesWithSuffix

構文

```
public String[] getSubscriberNamesWithSuffix(String lastBulkEnd,
int numOfSubscribers,
String suffix)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

説明

指定されたサフィックスで名前が終わるサブスクライバを SM データベースから取得します。

この操作の意味は、「getSubscriberNames」（p.3-23）操作の意味と同じです。

パラメータ

lastBulkEnd ― getSusbcriberNames 操作の「パラメータ」（p.3-23）の項を参照してください。

numOfSubscribers ― getSusbcriberNames 操作の「パラメータ」（p.3-23）の項を参照してください。

suffix ― 要求対象のサブスクライバ名のサフィックスを示す文字列（大文字と小文字は区別されます）

戻り値

要求されたサフィックスで終わるサブスクライバ名のアルファベット順配列

getSubscriberNames 操作の「戻り値」（p.3-23）の項を参照してください。

RPC 例外エラー コード

このメソッドで戻る可能性のあるエラー コードは次のとおりです。

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE_DATABASE_EXCEPTION

エラー コードの説明については、「エラー コードのリスト」（p.A-1）を参照してください。

getDomains

構文

```
public String[] getDomains()
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

説明

SM ドメイン リポジトリ内の最新のサブスクライバ ドメインのリストを提供します。

戻り値

SM 内のサブスクライバ ドメイン名の完全なリスト

RPC 例外エラー コード

なし

setPropertiesToDefault

構文

```
public void setPropertiesToDefault(String subscriberName,
String[] properties)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

説明

1 つのサブスクライバの指定されたアプリケーション プロパティをリセットします。アプリケーションがインストールされている場合、該当するアプリケーション プロパティは、その時点でロードされているアプリケーション情報に基づいて、そのプロパティのデフォルト値に設定されます。アプリケーションがインストールされていない場合は、**java.lang.IllegalStateException** が戻ります。

パラメータ

subscriberName ― 「サブスクライバ名のフォーマット」（p.2-4）の説明を参照してください。

properties ― 「サブスクライバ プロパティ」（p.2-7）のプロパティのキーおよび値に関する説明を参照してください。

戻り値

なし

RPC 例外エラー コード

このメソッドで戻る可能性のあるエラー コードは次のとおりです。

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE_BAD_SUBSCRIBER_MAPPING
- ERROR_CODE_DOMAIN_NOT_FOUND
- ERROR_CODE_SUBSCRIBER_DOES_NOT_EXIST
- ERROR_CODE_NOT_A_SUBSCRIBER_DOMAIN
- ERROR_CODE_DATABASE_EXCEPTION

エラー コードの説明については、「エラー コードのリスト」（p.A-1）を参照してください。

removeCustomProperties

構文

```
public void removeCustomProperties(String subscriberName,
String[] customProperties)
throws InterruptedException, OperationTimeoutException, RpcErrorException
```

説明

1 つのサブスクライバの指定されたカスタム プロパティをリセットします。

パラメータ

subscriberName ― 「サブスクライバ名のフォーマット」（p.2-4）の説明を参照してください。

CustomProperties ― 「カスタム プロパティ」（p.2-7）のプロパティのキーおよび値に関する説明を参照してください。

戻り値

なし

RPC 例外エラー コード

このメソッドで戻る可能性のあるエラー コードは次のとおりです。

- ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME
- ERROR_CODE_SUBSCRIBER_DOES_NOT_EXIST
- ERROR_CODE_DATABASE_EXCEPTION

エラー コードの説明については、「エラー コードのリスト」（p.A-1）を参照してください。

ブロッキング API のコード例

ここでは、次に示す 2 つのコード例を紹介します。

- サブスクライバ数の取得
- サブスクライバの追加、サブスクライバ情報の出力、サブスクライバの削除
- サブスクライバ数の取得（p.3-30）
- サブスクライバの追加、情報の出力、サブスクライバの削除（p.3-31）

サブスクライバ数の取得

次の例では、SM データベース内のサブスクライバ総数と各サブスクライバ ドメイン内のサブスクライバ数を **stdout** に出力します。

```
package blocking;
import com.pcube.management.api.SMBlockingApi;
public class PrintInfo {
public static void main (String args[]) throws Exception {
SMBlockingApi bapi = new SMBlockingApi();
try {
//initiation
bapi.setReplyTimeout(300000);  //set timeout for 5 minutes
bapi.connect(args[0]);         // connect to the SM
//operations
String[] domains=bapi.getDomains();
int totalSubscribers=bapi.getNumberOfSubscribers();
System.out.println(
"number of susbcribers in the database:\t\t "+
totalSubscribers);
for (int i=0; i<domains.length; i++) {
int numberOfSusbcribersInDomain=
bapi.getNumberOfSubscribersInDomain(domains[i]);
System.out.println(
"number of susbcribers domain "+domains[i]+
":\t\t "+numberOfSusbcribersInDomain);
}
} finally {
//finalization
bapi.disconnect();
}
}
}
```

サブスクライバの追加、情報の出力、サブスクライバの削除

次に示すのは、サブスクライバ データベースにサブスクライバを追加し、その情報を取得して**stdout** に出力し、最後にそのサブスクライバをサブスクライバ データベースから削除するプログラムです。

```
package blocking;
import com.pcube.management.api.SMBlockingApi;
import com.pcube.management.api.SMApiConstants;
public class AddPrintRemove {
public static void main (String args[]) throws Exception {
checkArguments(args);
SMBlockingApi bapi = new SMBlockingApi();
try {
//initiation
bapi.setReplyTimeout(10000);   //set timeout for 10 seconds
bapi.connect(args[0]);         // connect to the SM
//add subscriber
System.out.println("+ adding subscriber to SM");
bapi.addSubscriber(
args[1],                     //name
new String[]{args[2]},       //mapping`
SMApiConstants.ALL_IP_MAPPINGS,
new String[]{args[3]},       //property key
new String[]{args[4]},       //property value
new String[]{"custom-key"}, //custom property key
new String[]{"10"},          //custom property value
args[5]);                    //domain
//Print subscriber
System.out.println("+ Printing subscriber");
Object[] subfields = bapi.getSubscriber(args[1]);
System.out.println("\tname:\t\t"+subfields[0]);
System.out.println("\tmapping:\t"+
((String[])subfields[1])[0]);
System.out.println("\tdomain:\t\t"+subfields[3]);
System.out.println("\tautologout:\t"+subfields[8]);
//Remove subscriber
System.out.println("+ removing subscriber from SM");
bapi.removeSubscriber(args[1]);
} finally {
//finalization
bapi.disconnect();
}
}
static void checkArguments(String[] args)  throws Exception{
if (args.length != 6) {
System.err.println(
"usage: java AddPrintRemove <SM-address>"+
" <subscriber-name><IP mapping><property-key>"+
" <property-value><domain>");
System.exit(1);
}
}
}
```

CHAPTER 4

ノンブロッキング API

この章では、ノンブロッキング API 固有の性能を紹介します。また、ノンブロッキング API のすべての操作を説明し、コードの例をいくつか示します。

信頼性のサポート

ノンブロッキング API は、信頼モードと非信頼モードという 2 つの異なるモードで動作可能です。これらのモードについては以下で詳しく説明します。モードを指定しない場合、デフォルト値として信頼モードが使用されます。

信頼モード

信頼モードでは、API は SM への要求が失われないように保証します。API は、SM に送信されたすべての API 要求を内部ストレージに維持しています。SM からの応答が受信されるまで、要求は確定したとはみなされず、API はその要求を内部ストレージから削除できません。API と SM の間に接続エラーが生じた場合、SM への接続が確立されるまで、API はすべての要求をその内部ストレージに蓄積します。再接続時に、API は確定していないすべての要求を SM に再送信するため、要求が失われることはありません。

(注)　信頼モードでは、要求の再送信順序が保証されます。API は、呼び出された順番に要求を再送信します。

非信頼モード

非信頼モードでは、API は SM に送信された要求の実行を保証しません。さらに、外部の信頼性メカニズムを実装しないかぎり、SM への接続が切断されると、API によって送信される要求はすべて失われます。

自動再接続のサポート

ノンブロッキング API は、接続エラー時の SM への自動再接続をサポートしています。このオプションが有効になっていると、API は SM への接続の切断を知ることができます。API は、接続が切断されると、再接続タスクを起動して、接続されるまで SM への再接続を試行します。

(注)　自動再接続サポートのオプションは信頼性モードとは関係なく設定できます。

マルチスレッドのサポート

ノンブロッキング API では、メソッドを同時に呼び出すスレッド数に制限はありません。

(注)　ノンブロッキング API でマルチスレッドにする場合、呼び出しの順序は保証されます。API は、呼び出された順番に操作を実行します。

ResultHandler インターフェイス

ノンブロッキング API では、結果ハンドラを設定できます。結果ハンドラは、2 つのメソッド、**handleSuccess** と **handleError** を持つインターフェイスです。以下のコードを参照してください。

```
public interface ResultHandler {
/**
* handle a successful result
*/
public void handleSuccess(long handle, Object result);
/**
* handle a failure result
*/
public void handleError(long handle, Object result);
}
```

API を通じて実行された操作結果（成功またはエラー）について通知を受けたい場合は、このインターフェイスを実装する必要があります。

（注） これは、結果を取得する **唯一の**インターフェイスです。結果は、API メソッドが呼び出し側に戻った直後に戻すことは**できません**。

（注） 操作結果を受信できるようにするには、結果を受信する API メソッドを呼び出す前に API の結果ハンドラを設定する必要があります。以下の例のように、API の接続後に結果ハンドラを設定することを推奨します。

handleSuccess と **handleError** のメソッドは、次に示す 2 つのパラメータを受け入れます。

- **Handle** ― 各 API の戻り値は、**long** 型のハンドルです。このハンドルによって、操作の呼び出しとその結果を関連付けることができます。値 *X* のハンドルを指定して **handle...** 操作が呼び出された場合、その結果は同じハンドル値（*X*）を呼び出し側に返した操作の結果と一致します。
- **Result** ― 実際の操作結果。NULL の結果を返す操作もあります。

ResultHandler インターフェイス例

以下の例は、**stdout**（結果が成功の場合）または **stderr**（結果がエラーの場合）にメッセージを出力する単純な結果ハンドラの実装です。この main メソッドは API を開始し、結果ハンドラを割り当てます。

結果ハンドラを正しく機能させるためには、以下の例に示されているコード シーケンスを守る必要があります。

（注） この例は、コールバック ハンドルの使用方法を示すものでは**ありません**。

```
import com.pcube.management.framework.rpc.ResultHandler;
import com.pcube.management.api.SMNonBlockingApi;
public class ResultHandlerExample implements ResultHandler{
public void handleSuccess(long handle, Object result) {
System.out.println("success: handle="+handle+", result="+result);
}
public void handleError(long handle, Object result) {
System.err.println("error: handle="+handle+", result="+result);
}
public static void main (String args[]) throws Exception{
if (args.length != 1) {
System.err.println("usage: ResultHandlerExample <sm-ip>");
System.exit(1);
}
//note the order of operations!
SMNonBlockingApi nbapi = new SMNonBlockingApi();
nbapi.connect(args[0]);
nbapi.setResultHandler(new ResultHandlerExample());
nbapi.login(...);
}
}
```

ノンブロッキング API の構築

ノンブロッキング API には、「API の構築」（p.2-3）で説明したコンストラクタに加えて、再接続期間や信頼性モードを設定できるコンストラクタもあります。

ノンブロッキング API の構文

ノンブロッキング API の追加コンストラクタの構文については、次のコード ブロックを参照してください。

```
public SMNonBlockingApi(long autoReconnectInterval)
public SMNonBlockingApi(boolean reliable, long autoReconnectInterval)
public SMNonBlockingApi(String legName, long autoReconnectInterval)
public SMNonBlockingApi(String legName,
    boolean reliable, long autoReconnectInterval)
```

ノンブロッキング API の引数

ノンブロッキング API の追加コンストラクタ用のコンストラクタ引数は、次のとおりです。

- **autoReconnectInterval**

  次のように、再接続タスクの再接続試行間隔（ミリ秒単位）を決めます。

  - 0 以下の値：再接続タスクは起動されません（自動再接続は試行されません）。
  - 0 より大きい値：接続エラーの場合、**autoReconnectInterval** ミリ秒ごとに再接続タスクが起動されます。
- デフォルト値：-1（自動再接続は試行されません）

（注） 自動再接続サポートを有効にするには、API の connect メソッドが少なくとも 1 回起動される必要があります。詳細は、「ノンブロッキング API のコード例」（p.4-11）を参照してください。

- **reliable**

  API が信頼モードで動作するかどうかを決定するフラグです。

  - TRUE ― API は信頼モードで動作します。
  - FALSE ― API は非信頼モードで動作します。
- デフォルト値：TRUE（API は信頼モードで動作します）
- **legName**

  LEG の名前。「API の構築」（p.2-3）を参照してください。

ノンブロッキング API 例

次のコードで構築される API は、信頼モードで動作し、自動再接続間隔は 10 秒です。

```
SMNonBlockingAPI nbapi = SMNonBlockingAPI(10000);
nbapi.connect(<SM IP address>);
```

次のコードで構築される API は、信頼モードで動作し、自動再接続は試行しません。

```
// API construction
SMNonBlockingAPI nbapi = SMNonBlockingAPI();
// Connect to the API
nbapi.connect(<SM IP address>);
```

次のコードで構築される API は、非信頼モードで動作し、自動再接続を試行します。

```
// API construction
SMNonBlockingAPI nbapi = SMNonBlockingAPI(false,10000);
// Initial connection - to enable the reconnect task
nbapi.connect(<SM IP address>);
```

ノンブロッキング API の初期化

ノンブロッキング API では、一部の内部プロパティを初期化することによって、API をカスタマイズできます。この初期化は、**init** メソッドを使用して実行されます。

(注)　この設定を有効にするには、**connect** メソッドの**前**に **init** メソッドを呼び出す必要があります。

次のプロパティを設定できます。

- 出力キュー サイズ — 内部バッファ サイズ。これによって、SM に送信されるまで API が蓄積できる最大要求数が決まります。デフォルトは 1024 です。
- 操作タイムアウト — 応答のない PRPC プロトコル接続の望ましいタイムアウトに関するヒント（ミリ秒）。デフォルトは 45 秒です。

ノンブロッキング API の初期化構文

ノンブロッキング API の init メソッドの構文は、次のとおりです。

```
public void init(Properties properties)
```

ノンブロッキング API の初期化パラメータ

ノンブロッキング API の **init** メソッドのパラメータは次のとおりです。

- **properties (java.util.Properties)**
  前述のプロパティを設定できます。
  - 出力キュー サイズを設定するには、**prpc.client.output.machinemode.recordnum** を使用します。
  - 操作タイムアウトを設定するには、**prpc.client.operation.timeout** を使用します。

ノンブロッキング API の初期化例

ノンブロッキング API では、次のコード例のような方法で、初期化時にプロパティをカスタマイズできます。**connect** メソッドの**前**に **init** メソッドが呼び出されている点に注意してください。

```
// API construction
SMNonBlockingAPI nbapi = SMNonBlockingAPI(10000);
// API initialization
java.util.Properties p = new java.util.Properties();
p.setProperty("prpc.client.output.machinemode.recordnum", 2048);
p.setProperty("prpc.client.operation.timeout", 60000);// 1 minute
nbapi.init(p);
// initial connect to the API to enable the reconnect task
nbapi.connect(<SM API address>);
```

ノンブロッキング API メソッド

ここでは、ノンブロッキング API のメソッドについて説明します。

すべてのメソッドが **long** 型のハンドルを返します。これは、操作呼び出しとその結果を関連付けるために使用できます。（「ResultHandler インターフェイス」[p.4-3] を参照）。

結果ハンドラに渡される操作結果は、次の点を除けば「ブロッキング API」（p.3-1）の同じメソッドで説明した戻り値と同じです。

- 基本の型は Java クラスの表現に変換されます。たとえば、**int** は **java.lang.Integer** に変換されます。
- **Void** の戻り値は NULL に変換されます。

(注)　エラーと一緒に結果ハンドラが渡されるのは、ブロッキング API の一致する操作が、呼び出し時の SM データベースの状態に応じて、同じ引数を持つ例外を生成する場合 **だけ**です。

メソッドはどれも、SM との接続が確立される前に呼び出されると、**java.lang.IllegalStateException** を生成します。

ここでは、次のメソッドについて説明します。

- login（p.4-8）
- logoutByName（p.4-9）
- logoutByNameFromDomain（p.4-9）
- logoutByMapping（p.4-9）
- loginCable（p.4-9）
- logoutCable（p.4-10）

login

構文

```
public long login(String subscriberName,
String[] mappings,
short[] mappingTypes,
String[] propertyKeys,
String[] propertyValues,
String domain,
boolean isMappingAdditive,
int autoLogoutTime)
```

操作機能は、一致するブロッキング API 操作と同じです。詳細は、第 3 章「ブロッキング API のコード例」の「login」（p.3-5）を参照してください。

logoutByName

構文

```
public long logoutByName(String subscriberName,
String[] mappings,
short[] mappingTypes)
```

操作機能は、一致するブロッキング API 操作と同じです。詳細は、第 3 章「ブロッキング API のコード例」の「logoutByName」（p.3-8）を参照してください。

logoutByNameFromDomain

構文

```
public long logoutByNameFromDomain(String subscriberName,
String[] mappings,
short[] mappingTypes,
String domain)
```

操作機能は、一致するブロッキング API 操作と同じです。詳細は、第 3 章「ブロッキング API のコード例」の「logoutByNameFromDomain」（p.3-9）を参照してください。

logoutByMapping

構文

```
public long logoutByMapping(String mapping,
short mappingType,
String domain)
```

操作機能は、一致するブロッキング API 操作と同じです。詳細は、第 3 章「ブロッキング API のコード例」の「logoutByMapping」（p.4-9）を参照してください。

loginCable

構文

```
public long loginCable(String CPE,
String CM,
String IP,
int lease,
String domain,
String[] propertyKeys,
String[] propertyValues)
```

操作機能は、一致するブロッキング API 操作と同じです。詳細は、第 3 章「ブロッキング API のコード例」の「loginCable」（p.3-11）を参照してください。

logoutCable

構文

```
public long logoutCable(String CPE,
String CM,
String IP,
String domain)
```

操作機能は、一致するブロッキング API 操作と同じです。詳細は、第 3 章「ブロッキング API のコード例」の「logoutCable」（p.3-13）を参照してください。

ノンブロッキング API のコード例

ここでは、サブスクライバのログインおよびログアウトのコード例を紹介します。

ログインおよびログアウト

次に例では、事前定義された数のサブスクライバが SM にログインしたのち、ログアウトします。切断リスナと結果ハンドラが実装されている点に注意してください。

```
package nonblocking;
import com.pcube.management.framework.rpc.DisconnectListener;
import com.pcube.management.framework.rpc.ResultHandler;
import com.pcube.management.api.SMNonBlockingApi;
import com.pcube.management.api.SMApiConstants;
class LoginLogoutDisconnectListener implements  DisconnectListener {
public void connectionIsDown() {
System.err.println("disconnect listener:: connection is down");
}
}
class LoginLogoutResultHandler implements ResultHandler {
int count = 0;

//prints a success result every 100 results
public synchronized void handleSuccess(long handle, Object result) {
Object tmp = null;
if (++count%100 == 0) {
tmp = result instanceof Object[] ?
((Object[])result)[0] : result;
System.out.println("\tresult "+count+":\t"+tmp);
}
}
//prints every error that occurs
public synchronized void handleError(long handle, Object result) {
System.err.println("\terror: "+count+":\t"+ result);
++count;
}

//waits for result number 'last result' to arrive
public synchronized void waitForLastResult(int lastResult) {
while (count<lastResult) {
try {
wait(100);
} catch (InterruptedException ie) {
ie.printStackTrace();
}
}
}
}
public class LoginLogout {
public static void main (String args[]) throws Exception{
//check arguments
checkArguments(args);
int numSubscribersToLogin = Integer.parseInt(args[2]);
//instantiation
SMNonBlockingApi nbapi = new SMNonBlockingApi();
try {
//initation
nbapi.setDisconnectListener(
new LoginLogoutDisconnectListener());
nbapi.connect(args[0]);
LoginLogoutResultHandler resultHandler =
new LoginLogoutResultHandler();
nbapi.setResultHandler(resultHandler);
//login
```

```
System.out.println("login of "+numSubscribersToLogin
+" subscribers");
for (int i=0; i<numSubscribersToLogin; i++) {
nbapi.login("subscriber"+i,    //subscriber name
getMappings(i),    //a single ip mapping
new short[]{
SMApiConstants.MAPPING_TYPE_IP
},
null,          //no properties
null,
args[1],       //domain
false,         //mappings are not additive
-1);           //disable auto-logout
}
resultHandler.waitForLastResult(numSubscribersToLogin);
//logout
System.out.println("logout of "+numSubscribersToLogin
+" subscribers");
for (int i=0; i<numSubscribersToLogin; i++) {
nbapi.logoutByMapping(getMappings(i)[0],
SMApiConstants.MAPPING_TYPE_IP,
args[1]);
}
resultHandler.waitForLastResult(numSubscribersToLogin*2);
} finally {
nbapi.disconnect();
}
}
static void checkArguments(String[] args)  throws Exception{
if (args.length != 3) {
System.err.println("usage: java LoginLogout "+
"<SM-address><domain><num-susbcribers>");
System.exit(1);
}
}
//'automatic' mapping generator
private static String[] getMappings(int i) {
return new String[]{ "10." +((int)i/65536)%256 + "." +
((int)(i/256))%256 + "." + (i%256)};
}
}
```

APPENDIX A

エラー コードのリスト

この章では、Java API で使用するエラー コードのリストを示します。

エラー コードのリスト

エラー コードは、実際にどのようなエラーが原因で **RpcErrorException** が戻ったのかを知るために役立ちます。エラー コードを取得するには、**getErrorCode** メソッドを使用します。

エラーコード列挙体は、**com.pcube.management.api.SMApiConstants** インターフェイスで提供されます。エラー コードのリストとその説明を以下の表に示します。

表 A-1　エラー コードのリスト

エラー コード	説明
ERROR_CODE_BAD_SUBSCRIBER_MAPPING	マッピングのフォーマット不良またはサブスクライバへのマッピングの不正割り当て
ERROR_CODE_DOMAIN_NOT_FOUND	操作で指定されたドメインは SM ドメイン リポジトリに存在していません。
ERROR_CODE_ILLEGAL_ARGUMENT	メソッドで指定された引数の 1 つが無効です。
ERROR_CODE_ILLEGAL_SUBSCRIBER_NAME	指定されたサブスクライバ名が 40 文字を超えているか、または使用できない文字が含まれています。
ERROR_CODE_NOT_A_SUSBCRIBER_DOMAIN	操作で指定されたドメインは SM ドメイン リポジトリに存在していますが、サブスクライバ ドメインではありません。
ERROR_CODE_NUMBER_FORMAT	その API に与えられた VLAN マッピング文字列は 10 進数ではありません。
ERROR_CODE_SUBSCRIBER_DOES_NOT_EXIST	操作の実行対象であるサブスクライバは、SM データベース内に存在していません。
ERROR_CODE_SUBSCRIBER_DOMAIN_ASSOCIATION	サブスクライバは SM データベースに存在していますが、操作で指定されたドメインとは異なるドメインに関連付けられています。
ERROR_CODE_SUBSCRIBER_MAPPING_CONGESTION	操作でそのサブスクライバに指定されたマッピングはすでに別のサブスクライバに属しています。
ERROR_CODE_SUSBSCRIBER_ALREADY_EXISTS	操作の実行対象となったサブスクライバは、すでに SM データベース内に存在しています。

表 A-1　エラー コードのリスト（続き）

エラー コード	説明
ERROR_CODE_DATABASE_EXCEPTION	SM の内部エラー ― 操作中にデータベース エラーが発生しました。
ERROR_CODE_ARRAY_ACCESS	SM の内部エラー
ERROR_CODE_ATTRIBUTE_NOT_FOUND	SM の内部エラー
ERROR_CODE_CLASS_CAST	SM の内部エラー
ERROR_CODE_CLASS_NOT_FOUND	SM の内部エラー
ERROR_CODE_CLIENT_INTERNAL_ERROR	内部エラー
ERROR_CODE_CLIENT_OUT_OF_THREADS	内部エラー
ERROR_CODE_ILLEGAL_STATE	SM の内部エラー
ERROR_CODE_OBJECT_NOT_FOUND	SM の内部エラー
ERROR_CODE_OPERATION_NOT_FOUND	SM の内部エラー
ERROR_CODE_OUT_OF_MEMORY	SM の内部エラー
ERROR_CODE_RUNTIME	SM の内部エラー
ERROR_CODE_NULL_POINTER	SM の内部エラー
ERROR_CODE_SE_ERROR	SM の内部エラー。SM はその SCE デバイスに対して操作を実行できませんでした。
ERROR_CODE_UNKNOWN	SM または API の内部エラー