Cisco SCMS SCE Subscriber API プログラミング ガイド

Release 3.1
May 2007

Text Part Number: OL-8236-04-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。

CCSP, the Cisco Square Bridge logo, Follow Me Browsing, and StackWise are trademarks of Cisco Systems, Inc.; Changing the Way We Work, Live, Play, and Learn, and iQuick Study are service marks of Cisco Systems, Inc.; and Access Registrar, Aironet, ASIST, BPX, Catalyst, CCDA, CCDP, CCIE, CCIP, CCNA, CCNP, Cisco, the Cisco Certified Internetwork Expert logo, Cisco IOS, Cisco Press, Cisco Systems, Cisco Systems Capital, the Cisco Systems logo, Cisco Unity, Empowering the Internet Generation, Enterprise/Solver, EtherChannel, EtherFast, EtherSwitch, Fast Step, FormShare, GigaDrive, GigaStack, HomeLink, Internet Quotient, IOS, IP/TV, iQ Expertise, the iQ logo, iQ Net Readiness Scorecard, LightStream, Linksys, MeetingPlace, MGX, the Networkers logo, Networking Academy, Network Registrar, *Packet*, PIX, Post-Routing, Pre-Routing, ProConnect, RateMUX, ScriptShare, SlideCast, SMARTnet, StrataView Plus, SwitchProbe, TeleRouter, The Fastest Way to Increase Your Internet Quotient, TransPath, and VCO 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. (0501R)

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 SCE Subscriber API プログラミング ガイド
Copyright © 2007 Cisco Systems, Inc.
All rights reserved.

CONTENTS

このガイドについて

May 30, 2007, OL-8236-04-J

『*SCMS SCE Subscriber API プログラミング ガイド*』は、サブスクライバ プロビジョニングのために Service Control Engine（SCE）プラットフォームに直接アクセスする必要がある統合環境で使用します。

ここで説明する内容は次のとおりです。

対象読者

このガイドの対象読者は、サブスクライバ プロビジョニングを実行するポリシー サーバと SCE プラットフォームからなる統合環境で作業するネットワーク技術者またはコンピュータ技術者です。

マニュアルの変更履歴

Cisco Service Control のリリース	Part Number	発行日
リリース 3.1.0	OL-8236-04	2007 年 5 月

変更内容

- カスケード SCE の設定をサポートするために新たに RedundancyStateListener インターフェイスを追加しました。「SCE カスケード トポロジのサポート」（p.5-15）を参照してください。

Cisco Service Control のリリース	Part Number	発行日
リリース 3.0.5	OL-8236-03	2006 年 11 月

変更内容

- 「クォータ ステート復元イベント」（p.3-7）に関する新しいセクションを追加しました。
- 「SCAS_BB_Quota」（p.4-6）クラスをアップデートしました。
- QuotaListener インターフェイスが不適切なため、QuotaListenerEx インターフェイスを更新しました。「QuotaListenerEx インターフェイス クラス」（p.5-10）を参照してください。

Cisco Service Control のリリース	Part Number	発行日
リリース 3.0.3	OL-8236-02	2006 年 5 月

変更内容

- API コード例を更新しました。「API コードの例」（p.5-40）を参照してください。

Cisco Service Control のリリース	Part Number	発行日
リリース 3.0	OL-8236-01	2005 年 12 月

変更内容

- この文書の最初のバージョン

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」（p.1-1）	SCE Subscriber API を使用できるプラットフォーム、API のインストール方法、コンパイル方法、および起動方法について説明します。
第 2 章	「概念および用語」（p.2-1）	SCE Subscriber API を使用する際に役立つさまざまな用語および概念について説明します。
第 3 章	「API イベント」（p.3-1）	SCE Subscriber API. がアクセスするさまざまなイベントについて説明します。
第 4 章	「API データ型」（p.4-1）	各種 API データ タイプについて説明します。
第 5 章	「SCE Subscriber API を使用したプログラミング」（p.5-1）	API プログラミングの構造、クラス、メソッド、およびインターフェイスについて詳細に説明します。
第 6 章	「トラブルシューティング」（p.6-1）	API ロギング機能を使用した、API との統合に関するトラブルシューティング方法について説明します。API ロギングを使用すると、API クライアントと SCE 側の両方で受信されたパラメータを含めて、呼び出し中の処理をモニタできます。
付録 A	「エラー コードのリスト」（p.A-1）	API から戻されるエラー コードを示します。

関連資料

この API ガイドと併せて、次のシスコ マニュアルもご使用ください。

- 『*Cisco SCMS Subscriber Manager User Guide*』
- 『*Cisco Service Control Application for Broadband (SCA BB) User Guide*』
- 『*Cisco SCE 1000 2xGBE Installation and Configuration Guide*』
- 『*Cisco SCE 2000 4xGBE Installation and Configuration Guide*』

表記法

このガイドでは、次の表記法を使用しています。

- コマンド、キーワード、およびボタンは**太字**で示しています。
- ユーザが値を指定する引数は*イタリック体*で示しています。
- 画面に表示される情報の例は `screen` フォントで示しています。
- ユーザが入力する情報の例は**太字**の **`screen`** フォントで示しています。
- どれか 1 つを選ぶ必要のある選択要素は、縦棒（|）で区切られています。
- 角カッコ（[ ]）内の要素は省略可能です。
- 波カッコ（{}）内の要素は必須の選択肢です。
- 角カッコ内の波カッコ（[{}]）は省略可能な要素の中の必須選択肢を表します。

（注） 「*注釈*」です。役立つ情報や、このガイド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、推奨エイリアス、およびシスコ マニュアルに関する全般的な情報については、下記のサイトで毎月発行される『*What's New in Cisco Product Documentation*』を参照してください。この資料には、新規のまたは改訂されたシスコ技術資料がすべて掲載されています。

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、およびセキュリティ ガイドライン

CHAPTER 1

使用する前に

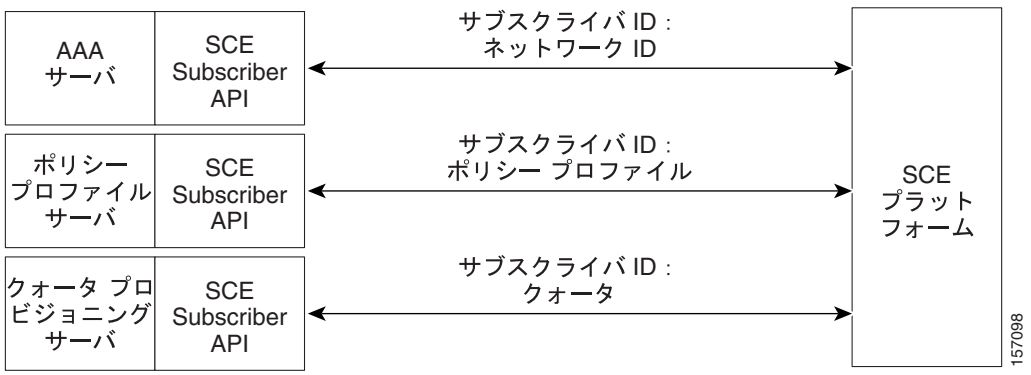
この章では、SCE Subscriber API を使用できるプラットフォーム、API のインストール方法、コンパイル方法、および起動方法について説明します。

SCMS SCE Subscriber API には、外部アプリケーション（ポリシー サーバ）を SCE に直接接続して、サブスクライバ プロビジョニングを行う機能があります。

*サブスクライバ プロビジョニング*は、サブスクライバ ID を相関付けに使用して、サブスクライバの*ネットワーク ID*、*ポリシー プロファイル*、および*クォータ*特性を更新するプロセスです。Service Control Application for Broadband（SCA BB）のサブスクライバの特性の詳細については、「サブスクライバの特性」（p.2-2）を参照してください。

API を複数のポリシー サーバにインストールし、同時に使用できます。また、次の図のように、API ごとにサブスクライバ プロビジョニング プロセスの異なるパートを実行できます。

図 1-1　複数のサーバにインストールされた SCE Subscriber API



API は Proprietary Remote Procedure Call（PRPC）プロトコルを使用して、SCE との接続上で転送を行います。PRPC はシスコが開発した独自の PRC プロトコルです。

（注）　API インスタンスごとに特定の SCE プラットフォームとの接続が確立されます。

SCMS SCE Subscriber API の制約事項

バージョン 3.0.5 の API は古いバージョンと下位互換性がありますが、バイナリ互換性はありません。新しいバージョンを使用するには、古いバージョンの API を使用するアプリケーションを再コンパイルする必要があります。この API は下位互換性があるため、アプリケーションのソース コードを変更する必要はありません。

(注)　SCE をバージョン 3.0.5 にアップグレードする場合は、API をバージョン 3.0.5 にアップグレードして、この API を使用するアプリケーションを再コンパイルする必要があります。

SCMS SCE Subsriber API

プラットフォーム

SCMS SCE Subscriber API は、Java バージョン 1.4 をサポートする任意のプラットフォームで実行できます。

パッケージの内容

簡潔にするため、ここではインストール ディレクトリ **sce-java-api-_<version>_<build-number>** を <*installdir*> と示します。

<*installdir*>**/javadoc** フォルダには、SCE Subscriber API JAVADOC マニュアルが格納されています。

<*installdir*>**/lib** フォルダには、**sceapi.jar** ファイルが格納されています。これは、API 実行可能ファイルです。さらに、API の実行に必要なその他の jar ファイルもこのフォルダに入っています。

表 1-1　インストール 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 仕様の索引
	（API 仕様ファイルなど）	API 仕様に関するマニュアル
<installdir>/lib		
	sceapi.jar	SCE Subscriber API 実行可能ファイル
	asn1rt.jar	API で使用されるユーティリティ jar
	log4j.jar	API で使用されるユーティリティ jar
	log4j.properties	ロギング機能で必要なプロパティ ファイル
	jdmkrt.jar	API で使用されるユーティリティ jar

パッケージの解凍とインストール

SCMS SCE Subscriber API は SCMS SM-LEG 配布ファイルとの一部として配布され、*sce_api* ディレクトリに格納されます。

SCMS SCE Subscriber API は UNIX tar ファイルにパッケージされています。SCMS SCE Subscriber API の解凍には、ほとんどの Windows 圧縮ユーティリティに含まれている UNIX tar ユーティリティを使用できます。

- UNIX プラットフォームへの配布ファイルのインストール（p.1-3）
- Windows プラットフォームへの配布ファイルのインストール（p.1-3）

UNIX プラットフォームへの配布ファイルのインストール

ステップ 1　SCMS SM-LEG 配布ファイルを解凍して、SCE Subscriber API 配布 tar ファイル **sce-java-api-dist.tar.gz** を特定します。

ステップ 2　配布ファイルを解凍します。

```
#>gunzip sce-java-api-dist.tar.gz
```

ステップ 3　SCE Subscriber API パッケージ tar ファイルを解凍します。

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

Windows プラットフォームへの配布ファイルのインストール

ステップ 1　zip 解凍ツール（WinZip など）を使用してパッケージを解凍します。

SCE プラットフォームの設定

ここでは、API を正常に機能させるために SCE プラットフォームで実行される設定について説明します。

前提条件

API は SCE プラットフォーム上の PRPC サーバに接続します。PRPC サーバはシスコが開発した独自の PRC プロトコルが動作するサーバです。詳細については、『*SCE User Guide*』を参照してください。

API を使用する前に、次の点を確認してください。

- SCE 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
- SCE 上の PRPC サーバが起動されていること

プル モードでの SCE の設定

プル モデル（「プル モデル」[p.2-3] を参照）で稼働している SCE プラットフォームから、サブスクライバ情報に関する要求を送信できるようにするには、SCE プラットフォームの CLI（コマンドライン インターフェイス）を使用して、次のように設定します。

SCE プラットフォームについての詳細は、『*Cisco SCE 1000 2xGBE Installation and Configuration Guide*』または『*Cisco SCE 2000 4xGBE Installation and Configuration Guide*』を参照してください。

ステップ 1　サブスクライバ テンプレートを設定します。

```
(config if)#>subscriber template import   CSV file
```

CSV ファイルのテンプレートおよびフォーマットの詳細については、『*Cisco Service Control Application for Broadband User Guide*』を参照してください。

ステップ 2　マッピングされていないサブスクライバ グループの範囲を設定します。

a. **subscriber anonymous group import** CLI を使用して、ファイルからアノニマス グループをインポートします。

```
(config if)#>subscriber anonymous group import   CSV file
```

b. あるいは、**subscriber anonymous group name** CLI を使用して、手動でアノニマス グループを定義します。

```
(config if)#>subscriber anonymous group name   NAMEIP-range   IP RANGE
```

RDR フォーマッタの設定

クォータ関連通知の送信に関する RDR フォーマッタの設定

RDR フォーマッタからクォータ関連通知を送信できるようにするには、SCE プラットフォーム上で次のように RDR フォーマッタを設定します。

ステップ 1　**RDR-formatter destination** CLI を使用します。

```
#>RDR-formatter destination 127.0.0.1 port 33001 category number 4 priority 100
```

異なるカテゴリへのクォータ RDR タグのマッピング

デフォルトでは、クォータ RDR タグはカテゴリ 4 にマッピングされています。別のカテゴリが必要な場合は、次のコマンドを使用します。

ステップ 1　**RDR-formatter rdr-mapping** CLI を使用します。

```
#>RDR-formatter rdr-mapping tag-ID   tag numbercategory-number   number
```

（注）　クォータ RDR タグ ID については、『*Cisco Service Control Application for Broadband User Guide*』を参照してください。

アプリケーションからクォータ関連通知を送信できるようにするには、『*Cisco Service Control Application for Broadband GUI*』の説明に従って、アプリケーションをイネーブルにする必要があります。設定方法については、『*Cisco Service Control Application for Broadband User Guide*』を参照してください。

RDR サーバの設定

クォータ通知を API が受信できるようにするには、RDR サーバをイネーブルにして、RDR フォーマッタで設定されているポートと同じポート上で待ち受ける必要があります。

RDR サーバの設定の確認

RDR サーバの設定を確認するには、次の手順を実行します。

ステップ 1　**show RDR-server** CLI を使用します。

```
#>show RDR-serverRDR server is ONLINE
RDR server port is 33001
```

RDR サーバのイネーブル化

RDR サーバをイネーブルにするには、次の手順を実行します。

ステップ 1　**RDR-server** コンフィギュレーション CLI を使用します。

```
#>configure(config)#>RDR-server Default RDR server port is 33001
```

RDR サーバ ポートの変更

RDR サーバ ポートを変更するには、次の手順を実行します。

ステップ 1　**RDR-server port** CLI を使用します。

```
#>configure(config)#>RDR-server port  port
```

API 切断タイムアウトの設定方法

SCE プラットフォームでは、切断した API を SCE プラットフォームに再接続するためのタイムアウトを設定できます。SCE はタイムアウト中にリソースを解放しないため、データは失われません。タイムアウトが経過しても、API が再接続しなかった場合、SCE は API が切断したとみなし、すべてのリソースを解放します。デフォルトのタイムアウト値は 5 分です。

API 切断タイムアウトの設定

API 切断タイムアウトを設定するには、次の手順を実行します。

ステップ 1　**management-agent sce-api timeout** CLI を使用します。

```
(config)# management-agent sce-api timeout    timeout-in-sec
```

切断タイムアウトのデフォルト値へのリセット

API 切断タイムアウトをデフォルト値にリセットするには、次の手順を実行します。

ステップ 1　**default management-agent sce-api　timeout** CLI を使用します。

```
(config)# default management-agent sce-api timeout
```

タイムアウト値の表示

タイムアウト値を表示するには、次の手順を実行します。

ステップ 1　**show management-agent sce-api** CLI を使用します。

```
# show management-agent sce-api
```

CHAPTER 2

概念および用語

この章では、SCMS SCE Subscriber API を使用する際に役立つさまざまな用語および概念について説明します。

サブスクライバの特性

Service Control Application for Broadband（SCA BB）ソリューションの基本的なエンティティの 1 つがサブスクライバです。サブスクライバは、SCAS BB ソリューションでサービス コンフィギュレーションを個別にモニタ、アカウンティング、および実施できるエンティティです。ここでは、SCA BB のサブスクライバの特性について簡単に説明します。サブスクライバ特性の形式および使用方法に関する詳細は、「API データ型」（p.4-1）を参照してください。

サブスクライバ ID

サブスクライバ ID はサブスクライバ固有の ID です。ユーザ名、IMSI、サブスクライバを一意に識別するその他のコードなどが該当します。

アノニマス サブスクライバ ID

プルモデル統合環境で作業する場合は、ポリシー サーバから実際にサブスクライバ ID が受信されるまで、不明なサブスクライバの IP アドレスにはそれぞれ一時的なサブスクライバ ID（アノニマス サブスクライバ ID）が割り当てられます。

プル モデル統合についての詳細は、「サブスクライバ統合モデル」（p.2-3）を参照してください。

ネットワーク ID

SCE はネットワーク ID（IP アドレス、IP 範囲、VLAN [仮想 LAN] など）をサブスクライバ エンティティにマッピングして、特定のトラフィック フローとサブスクライバを関連付けます。

ポリシー プロファイル

ポリシー プロファイルには、サブスクライバで実行されるポリシーを定義するために SCA BB ソリューションで使用されるパラメータ セットが含まれています。

クォータ

クォータには、サービス クォータ、またはサブスクライバを使用するために利用可能なクォータのクォータバケット値が含まれています。

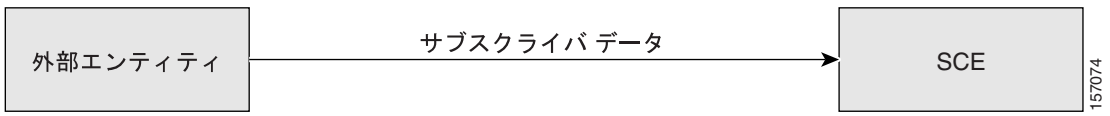
サブスクライバ統合モデル

次の用語は、SCE プラットフォームでサポートされている 2 つの動的なサブスクライバ統合モデルを示します。

プッシュ モデル

プッシュ モデル統合環境では、外部サーバから SCE プラットフォームにサブスクライバが導入（プッシュ）されます。この処理が実行されるのは、新しいサブスクライバがネットワークにログインした場合、または外部サーバがすべてのサブスクライバを認識していて、サブスクライバの接続時にサブスクライバを SCE ボックスに導入する場合です。

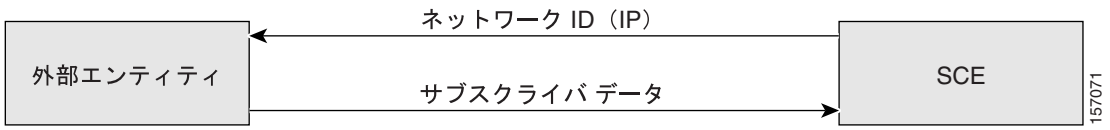
図 2-1　プッシュ モデルの概略図



プル モデル

プル モデル統合環境では、SCE プラットフォームは不明なサブスクライバ（アノニマス サブスクライバ）のトラフィックを検出すると、外部エンティティに対してサブスクライバ データを要求します。外部エンティティは要求されたサブスクライバ情報を取得して、SCE プラットフォームに戻します。

図 2-2　プル モデルの概略図



ノンブロッキング モデル

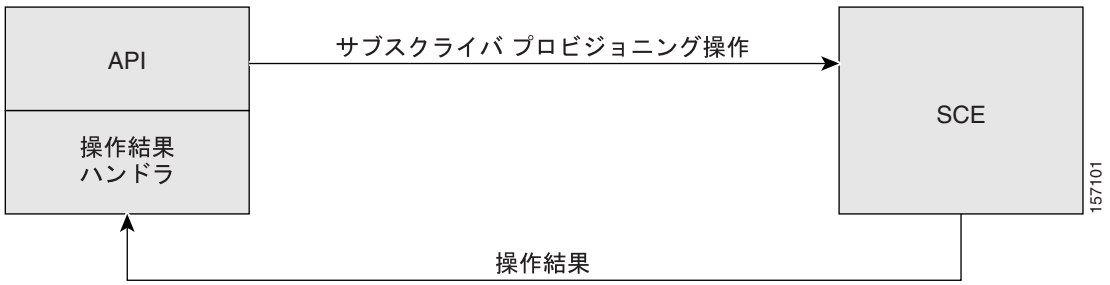
SCE Subscriber API はノンブロッキング モデルを使用して実装されます。ノンブロッキング メソッドは、サブスクライバ プロビジョニング操作が完了する前であっても、即座に戻されます。ノンブロッキング モデル メソッドは、入出力が含まれていて時間のかかる操作の場合に有利です。別のスレッドで操作を実行すれば、呼び出し側はほかのタスクを続行できるので、システム全体のパフォーマンスが向上します。

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。

API は操作結果ハンドラを使用して、操作結果を取得できます（「結果処理」[p.5-17] を参照）。

次の図に、サブスクライバ プロビジョニング操作中のノンブロッキング モデル メソッドを示します。

図 2-3　ノンブロッキング モデル

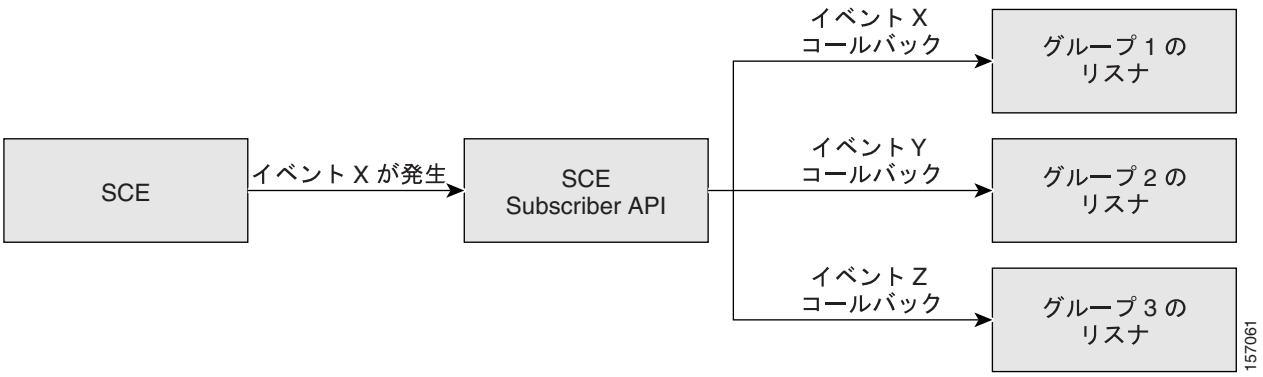


操作結果は、操作結果エラーのロギングや、操作で使用したパラメータの検査に使用できます。

通知リスナ

API には、SCE プラットフォームに特定のイベントが発生した場合に、通知を受信する機能があります。API は関連エンティティ（リスナ）のコールバック メソッドをアクティブにして、SCE から受信した通知を目的のリスナにディスパッチします。通知グループごとに定義できるリスナが **1 つ**のみの場合、通知は複数の論理グループに分割されます。

図 2-4　通知リスナ



特定の通知を受信するには、必要なコールバック関数を実装する API にリスナを登録する必要があります。リスナが登録されると、API は要求された通知をリスナにディスパッチできます。各リスナが次の 3 つのタイプの通知に登録されている場合、SCMS SCE Subscriber API はこれらのタイプの通知を送信します。

- ログインプル通知
- ログアウト通知
- クォータ通知

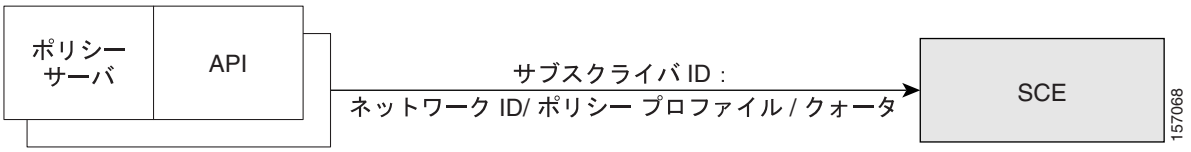
リスナの通知についての詳細は、「API イベント」（p.3-1）を参照してください。

サポート対象トポロジ

SCMS SCE Subscriber API では、次のトポロジを使用することを推奨します。

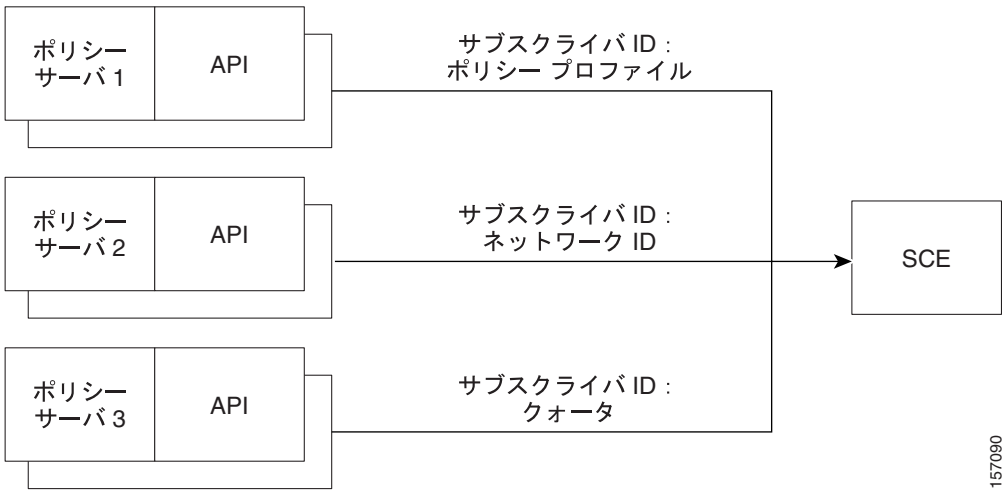
- サブスクライバ プロビジョニング プロセスの要素をすべて実行するポリシー サーバ（または 2 ノード クラスタ）× 1

図 2-5　サポート対象のトポロジ — ポリシー サーバ× 1



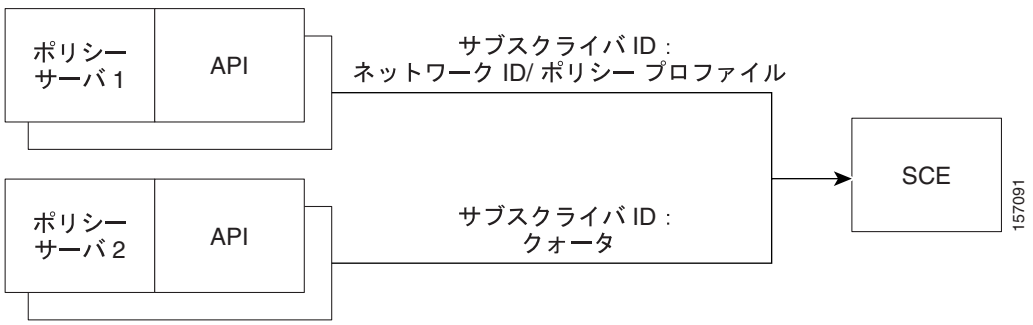
- ポリシー サーバ× 3（または 2 ノード クラスタ× 3）— サーバごとにサブスクライバ プロビジョニング プロセスの異なる要素を実行します。

図 2-6　サポート対象のトポロジ — ポリシー サーバ× 3



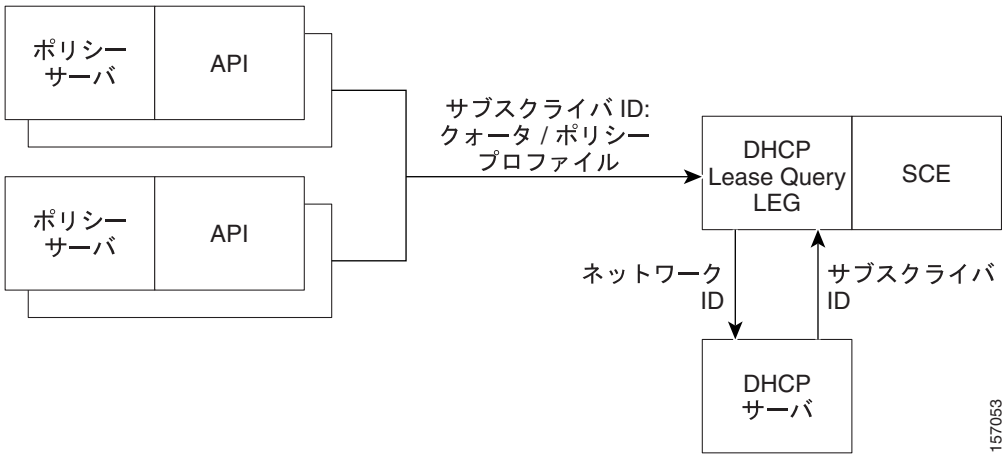
- ポリシー サーバ× 2（または 2 ノード クラスタ× 2）— 1 つのサーバでサブスクライバ プロビジョニング プロセスの要素を 2 つ実行し、もう 1 つのサーバでプロセスの要素を 1 つのみ実行します（任意の組み合わせが可能）。次に例を示します。

図 2-7　サポート対象のトポロジ — ポリシー サーバ× 2



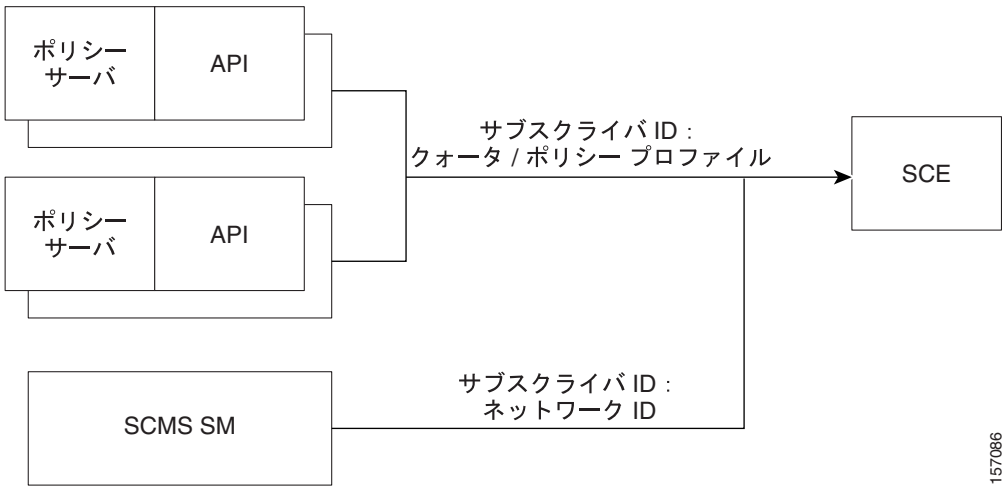
- ネットワーク ID とサブスクライバ ID をマッピングする DHCP Lease Query LEG と、1 台以上のポリシー サーバ（ポリシー サーバを 3 台配置した上図を参照）。次の図に、DHCP Lease Query LEG を示します。

図 2-8　サポート対象のトポロジ — DHCP Lease Query LEG



- ネットワーク ID とサブスクライバ ID をマッピングする SCMS SM と、1 台以上のポリシー サーバ。ポリシー サーバ数は、SM をネットワーク ID だけでなくポリシー プロファイル プロビジョニングにも使用するかどうかによって異なります。

図 2-9　サポート対象のトポロジ — SM



（注） API によってトポロジの使用が制限されることはありません。ただし、SCE プラットフォームでは、サブスクライバ プロビジョニングを実行するエントリ（ポリシー サーバ）の一部が関連付けられないことがあります。したがって、**同じプロビジョニング目的**にポリシー サーバを複数使用する場合は、**特に**注意する必要があります。そうししないと、SCE プラットフォームは、同じサブスクライバ プロビジョニング プロセスを実行する 2 台のポリシー サーバから、同じサブスクライバに対して異なる情報を受信することがあります。この場合、少なくとも 1 台のポリシー サーバとの同期が失われることがあります。たとえば、同じサブスクライバに 2 台のポリシー サーバを使用して、サブスクライバ ID/ ネットワーク ID の関連付けを行っている場合、SCE は、このサブスクライバに最後の更新を実行したポリシー サーバと常に同期します。

マルチスレッドのサポート

API では、メソッドを同時に呼び出すスレッドの数に制限がありません（ただし、使用可能メモリによってスレッド数は制限されます）。

(注) マルチスレッド シナリオでは、呼び出しの順番が**保証**されます。API は、呼び出された順番と同じ順序で操作を実行します。

自動再接続のサポート

接続障害が発生した場合、API は SCE との自動再接続をサポートします。このオプションがアクティブな場合、API は SCE との接続が切断された時期を判別できます。接続が切断されると、API は再接続タスクをアクティブにして、再接続に成功するまで、設定可能なインターバル内で SCE との再接続を試みます。

信頼性のサポート

SCMS SCE Subscriber API は*信頼できる* API として実装されます。この API を使用すると、SCE に対する要求や SCE からの通知が失われなくなります。また、SCE に送信されたすべての API 要求が内部ストレージに保存されます。要求が処理された SCE から確認応答を受信した場合のみ、要求は**コミットした**とみなされ、API は内部ストレージから要求を削除できます。API と SCE 間の接続に障害が発生すると、API は SCE との接続が再確立されるまで、内部ストレージにすべての要求を蓄積します。再接続されると、API は**コミットされていない**すべての要求を SCE に再送信して、要求が失われないようにします。

(注) 要求を再送信する順序は**保証**されています。API は、呼び出された順番と同じ順序で要求を再送信します。

ハイ アベイラビリティのサポート

API はハイ アベイラビリティをサポートしています。ポリシー サーバのハイ アベイラビリティ方式には、2 ノード クラスタ タイプが使用されます。この場合、同時にアクティブにできるサーバは 1 台のみです。別のサーバ（スタンバイ）は、SCE に接続されません。詳細は、「ハイ アベイラビリティの実装」（p.5-39）を参照してください。

同期

SCE が内部パラメータを処理しているサブスクライバに関して、SCE およびポリシー サーバを常に同期させる必要があります。そうしないと、SCE は一方のサブスクライバのトラフィックを別のサブスクライバのトラフィックと混同したり、SCE に到達しなかったポリシーが変更されて、サブスクライバの SLA（サービス レベル契約）が実行されなくなることがあります。詳細は、「SCE-API の同期」（p.5-33）を参照してください。

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は、次のヒントを考慮する必要があります。

- SCE に接続したら、API を複数回使用して、API と SCE との接続を維持します。接続は適切なタイミングで確立され、これによって、SCE 側と API クライアント側でリソースが割り当てられます。
- API 接続はスレッド間で共有されます。ポリシー サーバごとに接続を 1 つ確立することを推奨します。複数の接続を確立する場合は、SCE 側およびクライアント側でより多くのリソースが必要となります。
- API への呼び出しを同期しないでください。API への呼び出しはクライアントによって自動的に同期されます。
- ポリシー サーバ アプリケーションにログオン処理がバースト的に発生した場合は、これらのバーストを保持できるように、内部バッファ サイズを拡張します（ノンブロッキング方式）。
- 統合中に問題が発生した場合は、「SCE ロギング」（p.6-2）および「API クライアント ロギング」（p.6-6）に記載されているロギング機能を使用して、SCE クライアント ログ内の API 操作を表示し、トラブルシューティングを実行してください。
- ポリシー　サーバ　アプリケーションで処理の戻り値を記録または出力する場合は、デバッグ モードを使用します。
- SCE との接続の復元力を向上させるには、自動再接続機能を使用します。

■ ヒント

CHAPTER 3

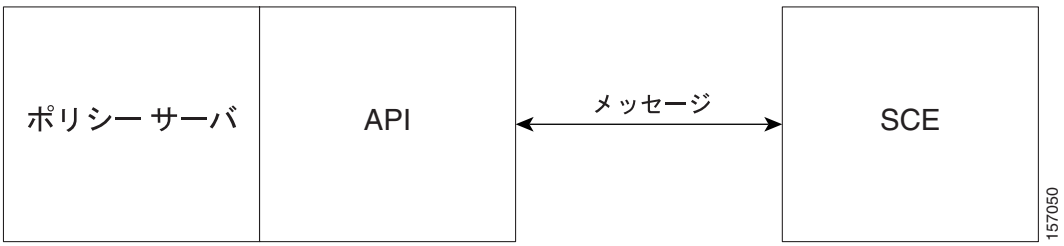
API イベント

この章では、SCMS SCE Subscriber API がアクセスする各イベントについて説明します。

API イベント

API は一連のイベント、つまり、ポリシー サーバと SCE プラットフォーム間で送受信された一連の定義済みメッセージにアクセスします。

図 3-1　API イベントの概要



すべてのメッセージには、メッセージの目的に従ってタイプを割り当てることができます。

- *要求* ― 情報または実行するアクションを要求します。要求のあとに応答が発生しないことがあります。
- *応答* ― 直前の要求に応答します。
- *通知* ― イベントが発生したことを反対側に通知します。

ほとんどのイベントは**プッシュ** モデルおよび**プル** モデルで使用できます。「サブスクライバ統合モデル」（p.2-3）を参照してください。

イベントは次のサブスクライバ プロビジョニング プロセス グループに分割できます。

- *ネットワーク管理イベント* ― サブスクライバ ネットワーク ID マッピングの変更に関連するイベントが含まれます。
- *ポリシー プロファイル管理イベント* ― サブスクライバ ポリシー プロファイルパラメータの変更に関連するイベントが含まれます。
- *クォータ管理イベント* ― サブスクライバ クォータの管理に関連するイベントが含まれます。
- *SCE 同期管理イベント* ― SCE 同期プロセスの管理に関連するイベントが含まれます。

バルク操作を実行して、複数のサブスクライバ上の同じイベントに関する複数のトリガーを 1 つのグローバル イベントにバンドルできます。

ここでは、各タイプのイベントの概要を示します。

ネットワーク ID 管理イベント

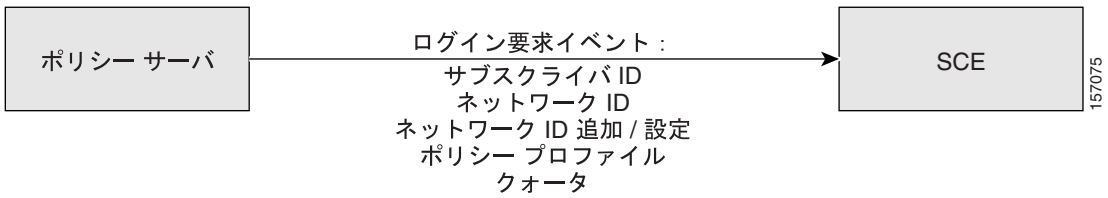
ログイン イベント

サブスクライバがネットワークに接続すると、ログイン イベントが発生します。ログイン イベントは、プル モデルおよびプッシュ モデルで異なります。

プッシュ モデル

プッシュ統合モデルでは、ポリシー サーバによって SCE にサブスクライバが導入されます。たとえば、ポリシー サーバは Authentication, Authorization, Accounting（AAA; 認証、認可、アカウンティング）などの外部エンティティからサブスクライバ ログイン通知を受信し、必要なサブスクライバ属性を抽出して、情報を SCE プラットフォームに「プッシュ」します。

図 3-2　ログイン イベント — プッシュ モデル



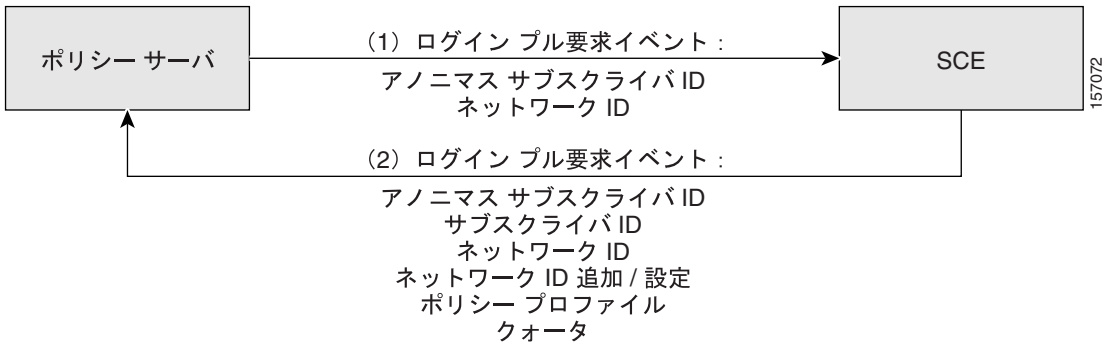
サブスクライバ ログイン操作を実行すると、新しいサブスクライバ レコードが SCE に作成されたり、既存サブスクライバが更新されたりすることがあります。たとえば、ケーブル モデム ネットワークの場合、サブスクライバはケーブル モデムです。このケーブル モデムに接続された CPE は、IP アドレスのリスト（あるいは範囲）として設定されます。この場合、同じモデムに接続された新しい CPE にログインすると、CPE の IP アドレスがサブスクライバのネットワーク ID リストに追加されます。

プル モデル

プル統合モデルでは、SCE は着信データ トラフィックから新しいサブスクライバを検出すると想定されます。新しいサブスクライバはアノニマス サブスクライバとしてシステムに入力され、いずれかのデフォルト ポリシーが割り当てられます。SCE は外部システムに要求（ログインプル要求）を送信し、サブスクライバ ログイン情報（ログインプル応答）を提供します。この IP に対応する情報がない場合は、要求が省略されます。SCE に提供されるログイン情報によって、アノニマス サブスクライバは実際のサブスクライバに置き換えられ、正しいポリシーが適用されます。

外部システムがログインを拒否して、アノニマス サブスクライバからトラフィックが着信し続けている場合は、プル要求が再試行されます。

図 3-3　ログイン イベント — プル モデル

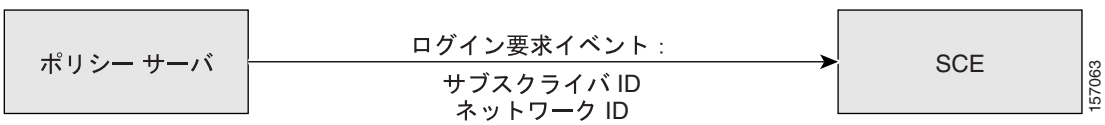


（注） ログイン要求イベントおよびログイン プル応答イベントは「ネットワーク ID 管理イベント」として分類されているにもかかわらず、SCE にサブスクライバ情報をすべて送信できるように最適化されています。単一のポリシー サーバでサブスクライバ プロビジョニングのすべてのパートを実行する場合は、これらのイベントを使用して、ポリシー プロファイルおよびクォータを更新することを推奨します。複数のポリシー サーバが配置されたトポロジの場合は、ポリシー プロファイルおよびクォータの情報を更新するために、別々のイベントを使用します（以下を参照）。トポロジについての詳細は、「サポート対象トポロジ」（p.2-6）を参照してください。

ログアウト イベント

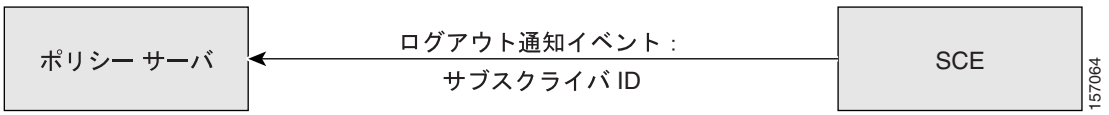
ログアウト イベントは、サブスクライバが特定のネットワーク ID を使用しないことを示します。ログアウト イベントのあとに、SCE からサブスクライバ レコードが削除されるとはかぎりません。たとえば、ケーブル モデム ネットワークで複数の CPE が同じモデムに接続されている場合は、1 つの CPE からログアウトしても、別の CPE が接続されたままであれば、サブスクライバは削除されないことがあります。すべての CPE（サブスクライバのネットワーク ID）を切断すると、サブスクライバが実際に削除されます。

図 3-4　ログアウト要求イベント



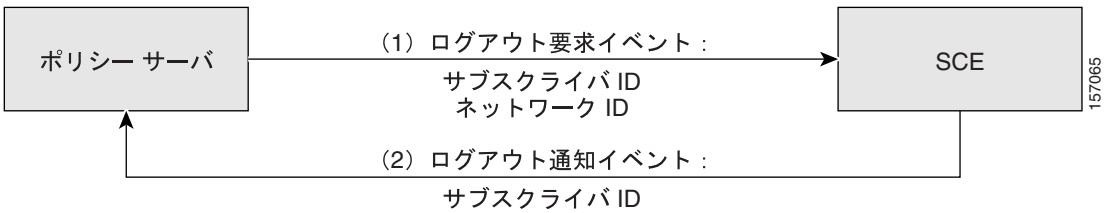
プル モデルでは、特定の期間中にサブスクライバが非アクティブであることを SCE が識別した場合などに、ログアウト イベントが発生することがあります。SCE はサブスクライバを「ログアウト」し、ログアウト通知イベントを送信します。

図 3-5　ログアウト通知イベント



ログアウト処理に続いてログアウト通知イベントが発生することもあります。このようになるのは、実際にサブスクライバが削除された場合です。たとえば、このサブスクライバに関係付けられている有効なネットワーク マッピング（IP）がこれ以上ない場合に、この処理が発生します。

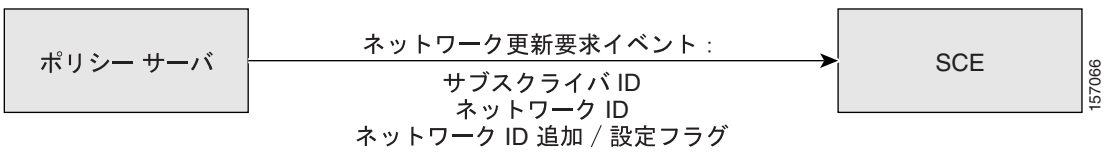
図 3-6　ログアウト要求イベント



ネットワーク ID 更新イベント

このイベントは、SCE プラットフォームの既存のサブスクライバのネットワーク ID を更新するために、ポリシー サーバから SCE に送信される要求です。このイベントに応答は不要です。

図 3-7　ネットワーク ID 更新イベント

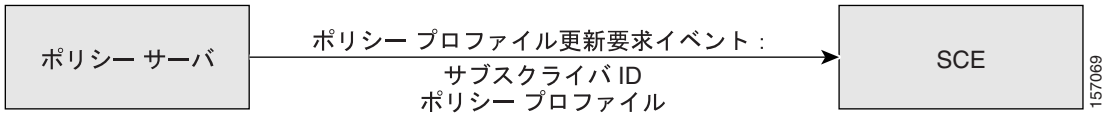


ポリシー プロファイル管理イベント

プロファイル更新イベント

このイベントは、SCE プラットフォームの既存のサブスクライバのポリシー プロファイルを更新するために、ポリシー サーバから SCE に送信される要求です。このイベントに応答は不要です。

図 3-8　プロファイル更新イベント



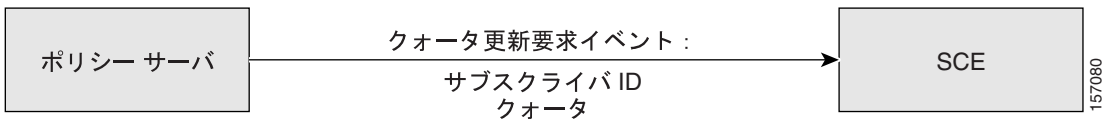
（注） 上記のように、ログイン要求イベントおよびログイン プル応答イベントによって、ポリシー プロファイルが更新されることもあります。

クォータ管理イベント

クォータ更新イベント

クォータ更新イベントは、SCE プラットフォームの既存のサブスクライバのクォータを更新するために、ポリシー サーバから SCE に送信される要求です。このイベントに応答イベントは不要です。

図 3-9　クォータ更新イベント

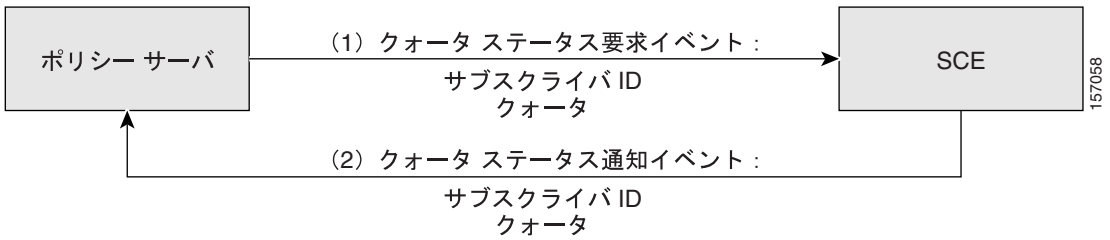


（注） 上記のように、ログイン要求イベントおよびログイン　プル応答イベントによって、クォータが更新されることもあります。

クォータ ステータス取得イベント

クォータ ステータス取得イベントは、SCE プラットフォームの既存のサブスクライバのクォータ情報を報告するために、ポリシー サーバから SCE に送信される要求です。このイベントのあとに、クォータ ステータス通知イベントが続きます。

図 3-10　クォータ ステータス取得イベント

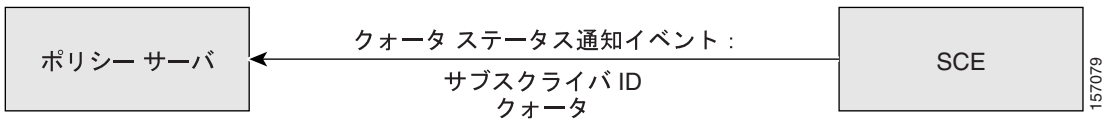


（注） ポリシー サーバから特定の要求がない場合でも、SCE からクォータ ステータス通知イベント定期的に送信されることがあります。「クォータ ステータス イベント」（p.3-6）を参照してください。

クォータ ステータス イベント

SCE はクォータ ステータス通知イベントを使用して、残りのクォータをポリシー サーバに通知します。このイベントは、事前定義されたインターバルで定期的に呼び出されます。

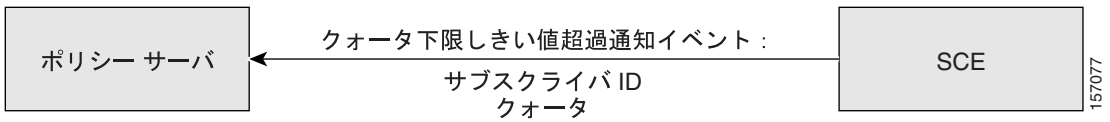
図 3-11　クォータ ステータス イベント



クォータ下限しきい値超過イベント

SCE はクォータ下限しきい値超過通知イベントを使用して、特定のサブスクライバの特定のサービスの残りのクォータが、事前定義されたしきい値を下回っていることを、ポリシー サーバに通知します。ポリシー サーバから SCE に送信されるクォータ更新要求イベントのあとに、このイベントが続くことがあります。ただし、この処理が必須なわけではありません。

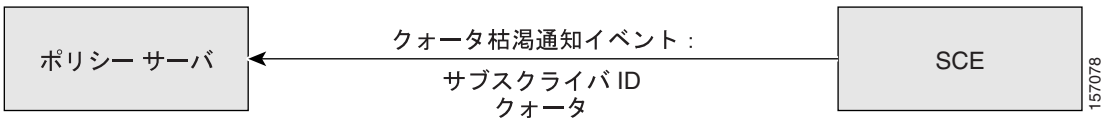
図 3-12　クォータ下限しきい値超過イベント



クォータ枯渇イベント

SCE はクォータ枯渇通知イベントを使用して、特定のサブスクライバの特定のサービスのクォータが枯渇していることをポリシー サーバに通知します。ポリシー サーバから SCE に送信されるクォータ更新要求イベントのあとに、このイベントが続くことがあります。

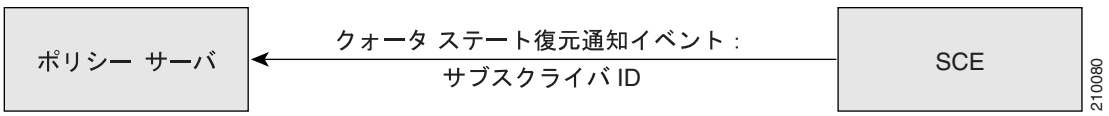
図 3-13　クォータ枯渇イベント



クォータ ステート復元イベント

クォータ ステート復元イベントは、SCE プラットフォームの既存のサブスクライバのクォータを復元するために、SCE からポリシー サーバに送信される通知です。このイベントは、サブスクライバが SCE にログインした直後に呼び出されます。このイベントのあとに、ポリシー サーバからのクォータ更新イベントが続くことがあります。

図 3-14　クォータ ステート復元イベント

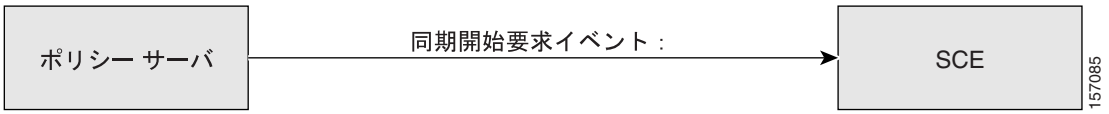


SCE 同期手順イベント

同期開始イベント

同期開始要求イベントは、同期プロセスが開始しようとしていることを SCE に通知する場合に使用します。SCE はこの要求を使用して、同期プロセスの準備に必要な内部処理を実行します。このイベントには、プッシュおよびプル コンポーネントがあります。

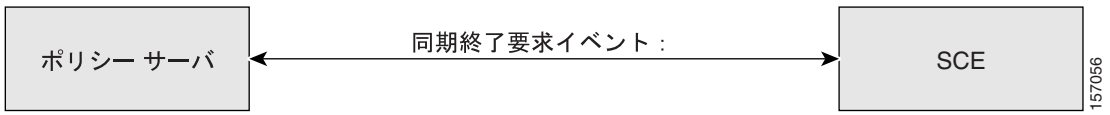
図 3-15　同期開始イベント



同期終了イベント

同期終了要求イベントは、同期プロセスが終了したことを SCE に通知する場合に使用します。このイベントには、プッシュおよびプル コンポーネントがあります。

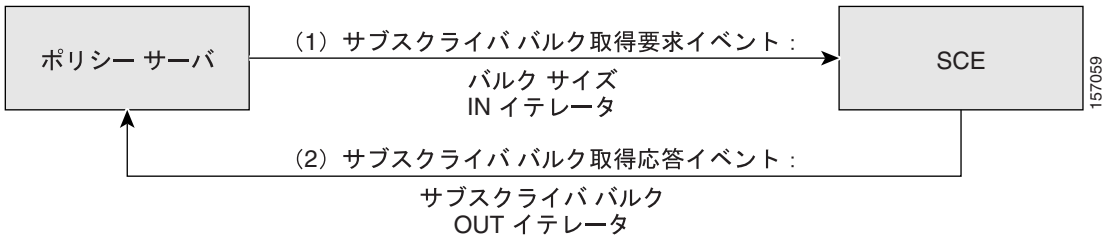
図 3-16　同期終了イベント



サブスクライバ取得イベント

ポリシー サーバは、SCE のプル モデル同期プロセス中に、SCE が現在処理しているサブスクライバをすべて取得する必要があります。サブスクライバ バルク取得要求イベントは、SCE が現在処理しているサブスクライバの次のバルクを取得するために、ポリシー サーバから SCE に送信される要求です。この要求を受信した SCE は、サブスクライバ名およびネットワーク ID を指定するサブスクライバ バルク取得応答イベントで応答します。

図 3-17　サブスクライバ取得イベント



詳細は、「プル モデル」（p.2-3）および「プル モデルの同期手順」（p.5-35）を参照してください。

CHAPTER 4

API データ型

この章では、SCMS SCE Subscriber API で使用されるさまざまな API データ型について説明します。

サブスクライバ ID

SCE Subscriber API のほとんどのメソッドでは、入力パラメータとしてサブスクライバ ID を使用する必要があります。サブスクライバ ID は、サブスクライバ名または CM MAC（メディア アクセス制御）アドレスを表すストリングです。ここでは、サブスクライバ ID のフォーマット ルールについて説明します。

サブスクライバ名では、*大文字と小文字を区別*します。使用できる文字数は最大 64 文字です。34（"）、39（'）、96（`）を除く 32 ～ 126（両端を含む）の ASCII コードで出力可能な文字はすべて使用できます。

次に例を示します。

```
String subID1="john";
String subID2="john@yahoo.com";
```

ネットワーク ID のマッピング

ネットワーク ID 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 マッピングの典型的な例です。現在のところ、IP アドレス、IP 範囲、VLAN のマッピング タイプが SCE でサポートされています。

NetworkID クラスは、さまざまなタイプのサブスクライバ ネットワーク ID を表します。

API は次のサブスクライバ マッピング タイプをサポートします。

- IP アドレスまたは IP 範囲
- VLAN タグ

（注） 同じサブスクライバに IP アドレス /IP 範囲と VLAN タグを同時に使用することはできません。

ネットワーク ID を含むサブスクライバ操作を使用している場合、発信側は NetworkID パラメータを指定する必要があります。

NetworkID クラス コンストラクタの定義は、次のとおりです。

```
public NetworkID(String mapping,short mappingType) throws Exception
public NetworkID(String[] mappings,short[] mappingTypes) throws Exception
```

NetworkID コンストラクタのパラメータは、次のとおりです。

- **java.lang.String** マッピング ID、またはマッピング ID の配列
- short マッピング タイプ、またはマッピング タイプの配列

配列を渡す場合は、MappingType 配列に、マッピング配列と同数の要素、または単一の要素が含まれている必要があります。

- 配列に複数の要素が含まれている場合は、NetworkID.TYPE_IP または NetworkID.TYPE_VLAN 定数を使用します。
- 単一配列要素の場合は、NetworkID.ALL_IP_MAPPINGS または NetworkID.ALL_VLAN_MAPPINGS 定数を使用します。

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、クラス NetworkID で指定されます。

- `com.scms.common.NetworkID.TYPE_IP`

  com.scms.common.NetworkID.ALL_IP_MAPPINGS は、マッピング ID 配列内のすべてエントリが 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** ～ **10.1.1.255** の範囲のアドレスすべてを表します。

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

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、次の範囲の 10 進数です。**[2-2046]**

VLAN マッピングの型は、**com.scms.common.NetworkID** クラスで指定されます。

- IP アドレスのマッピングの型は、クラス NetworkID で指定されます。
- `com.scms.common.NetworkID.TYPE_VLAN`
- **com.scms.common.NetworkID.ALL_VLAN_MAPPINGS** は、マッピング ID 配列内のすべてエントリが VLAN マッピングであることを指定します。

ネットワーク ID マッピングの例

単一の IP アドレスを持つ NetworkID を作成します。

```
NetworkID nid = new NetworkID("1.1.1.1",NetworkID.TYPE_IP)
```

IP アドレス範囲を持つ NetworkID を作成します。

```
NetworkID nid = new NetworkID("1.1.1.1/24",NetworkID.TYPE_IP)
```

複数の IP アドレスを持つ NetworkID を作成します。

```
NetworkID nid = new NetworkID(new String[]{"1.1.1.1","2.2.2.2","3.3.3.3"},
NetworkID.ALL_IP_MAPPINGS)
```

単一の VLAN アドレスを持つ NetworkID を作成します。

```
NetworkID nid = new NetworkID("23",NetworkID.TYPE_VLAN)
```

SCA BB サブスクライバのポリシー プロファイル

ポリシー プロファイルには、サブスクライバ ポリシー情報が記述されています。ポリシー プロファイルは一般に、ポリシー パッケージで識別される静的に定義されたポリシーと、動的な性質を持つことがある一連のサブスクライバ ポリシー プロパティの、2 つの主要パートで構成されます。パッケージ ID はポリシー パッケージを識別します。サブスクライバ トラフィックに適用されるルールの大部分は、パッケージ ID から取得されます。

SCA BB 内のサブスクライバ ポリシー 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 Service Control Application Suite for Broadband User Guide*』を参照してください。

SCA BB バージョン 3.0 には次のプロパティがあります。

- packageId ― サブスクライバのパッケージ ID を定義します。
- monitor ― このサブスクライバのトランザクションごとに Raw Data Record（RDR）を発行するかどうかを指定します。

PolicyProfile クラス

API には、API 操作に必要なサブスクライバ ポリシー プロファイルをフォーマット化するための PolicyProfile クラスがあります。

次のメソッドは、ポリシー プロパティの配列に基づいて PolicyProfile クラスを構築します。

```
public PolicyProfile(String[] policy)
```

（注） 配列内の各ストリングは、次のように符号化する**必要があります**。

```
property_name=property_value
```

次のメソッドを使用すると、上記フォーマットに従って、プロファイルにポリシー プロファイルを追加できます。

```
public void addPolicyProperty(String policyProperty)
```

（注） このメソッドはパフォーマンスを高めるために最適化されていません。パフォーマンスを最大にするには、PolicyProfile コンストラクタを使用します。

例

```
PolicyProfile pp = new PolicyProfile(new
         String[]{"packageId=22", "monitor=1"})
```

サブスクライバ クォータ

SCA BB でクォータ プロビジョニングを実行するには、サブスクライバ クォータ バケットを使用します。サブスクライバごとに 16 のバケットがあり、各バケットを容量またはセッション数で定義できます。サブスクライバが特定のサービスを利用すると、消費した容量またはセッション数が、いずれかのバケットから減らされます。各サービスで使用するバケットは、SCA BB コンソール を使用して汎用ポリシー定義内で定義されたサービス設定によって決まります。ボリューム バケットの消費量は L3 キロ バイト単位でカウントされ、セッション バケットの消費量はセッション数でカウントされます。たとえば、閲覧と電子メール サービスはバケット番号 1 から、P2P サービスはバケット番号 2 から、それぞれクォータを消費するものとし、それ以外のサービスはどれも特定のバケットに対応付けないように定義することができます。

クォータ バケットは次のコンポーネントで構成されます。

- バケット ID ― 定義済みポリシー内の定義に従うバケットの一意の ID（**String**）。有効値の範囲は [1-16] です。
- バケット値 ― クォータ バケット値（**long**）

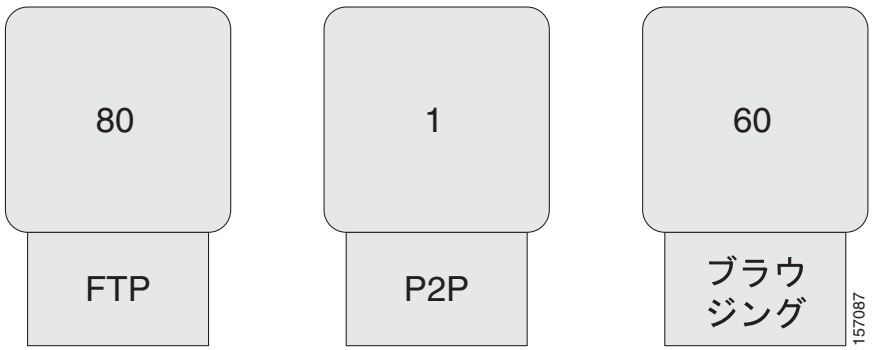
クォータ操作によって、サブスクライバのクォータ バケットは動的に変更されます。クォータ操作には 2 つのタイプがあります。

- ADD_QUOTA_OPERATION ― SCE プラットフォーム上のバケットの現在値に、新しいクォータ値を加算します。
- SET_QUOTA_OPERATION ― SCE プラットフォーム上のクォータ バケット値を新しい値で交換します。

例

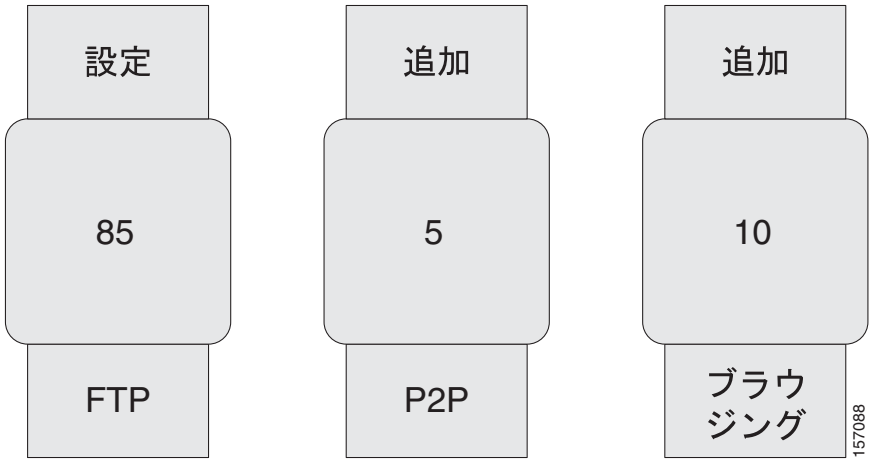
SCE のサブスクライバ A のクォータの現在値は、次のとおりです。

図 4-1　サブスクライバ クォータ ― 現在値



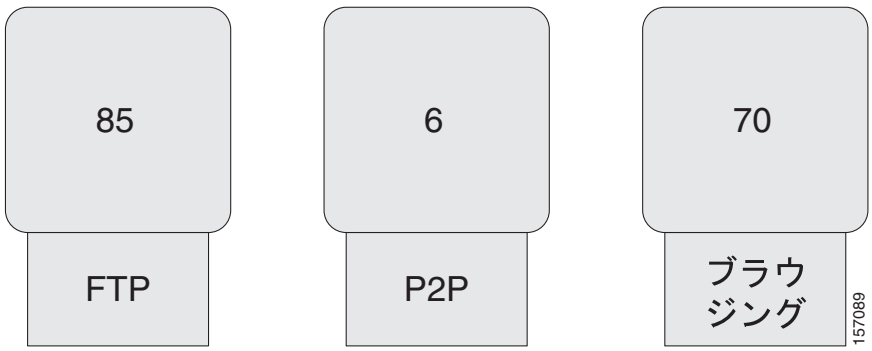
既存クォータに次のアクションを適用するとします。

図 4-2　サブスクライバ クォータ — アクションの適用



クォータ アクションを実行すると、次のようになります。

図 4-3　サブスクライバ クォータ — 結果



サブスクライバ クォータの詳細については、『*Cisco Service Control Application for Broadband User Guide*』を参照してください。

ここでは、サブスクライバ クォータ管理操作などのために API が提供するクラスについて説明します。

SCAS_BB_Quota

SCAS_BB_Quota クラスは、すべてのコールバック関数で QuotaListenerEx インターフェイスが使用するクォータ インターフェイスを実装します。「QuotaListenerEx インターフェイス クラス」(p.5-10) を参照してください。

次のメソッドは、ID および値の配列に基づいて SCAS_BB_Quota を構築します。

```
public SCAS_BB_Quota (String[] bucketIDs,
long[] bucketValues)
```

次のメソッドは、ID および値の配列、プロファイル ID、理由、およびタイムスタンプに基づいて SCAS_BB_Quota を構築します。

```
public SCAS_BB_Quota (String[] bucketIDs,
long[] bucketValues,
int quotaProfileId,
int reason,
long timestamp)
```

次のメソッドでは、クォータ バケットの ID を取得できます。

```
public String[] getBucketIDs()
```

次のメソッドでは、クォータ バケットの値を取得できます。

```
public long[] getBucketValues()
```

quotaProfileId パラメータはクォータ プロファイルの ID、つまり パッケージ ID です。次のメソッドでは、クォータ プロファイル ID を取得できます。

```
public int getQuotaProfileId()
```

reason パラメータはクォータ ステータス イベントにのみ関連し、3 つの有効値を取ります。

- 0 ― 設定時間（2 分おきなど）に達しました。
- 1 ― サブスクライバ ログアウトによってクォータ ステータス イベントがトリガーされました。
- 2 ― パッケージ変更によってクォータ ステータス イベントがトリガーされました。

次のメソッドでは、理由を取得できます。

```
public int getReason()
```

timestamp パラメータには、イベントが生成された（SCE 内の）時刻が含まれます。このパラメータは、1970 年 1 月 1 日 00:00 GMT（グリニッジ標準時）からの秒数で計算されます。

次のメソッドでは、タイムスタンプを取得できます。

```
public long getTimestamp()
```

SCAS_BB_QuotaOperation

SCAS_BB_QuotaOperation クラスは QuotaOperation インターフェイスを実装します。このインターフェイスは、ログイン操作（「ログオン操作」[p.5-20] を参照）やクォータ更新操作（「quotaUpdate 操作」[p.5-29] を参照）など、サブスクライバのクォータを含むサブスクライバ プロビジョニング操作に使用されます。

次のメソッドは、ID、値、およびアクションの配列に基づいて SCAS_BB_QuotaOperation を構築します。

```
public SCAS_BB_QuotaOperation (String[] IDs,
long[] values,
short[] actions)
```

次のメソッドでは、クォータ バケットの ID を取得できます。

```
public String[] getBucketIDs()
```

次のメソッドでは、クォータ バケットの値を取得できます。

```
public long[] getBucketValues()
```

次のメソッドでは、クォータ バケットのアクションを取得できます。

```
public short[] getBucketActions()
```

バルク操作のデータ型

それぞれ独自のパラメータを持つ複数のサブスクライバに同じメソッドを実行する場合は、バルク クラスおよびバルク操作を使用します。API にはバルク クラスがあり、バルク操作の結果処理や、SCE からの一括通知に使用できます。バルク クラスは **loginBulk** や **logoutBulk** などのバルク メソッドに渡されます。

次に、バルク操作を使用する場合の注意事項を示します。

- すべてのバルク クラスは共通の BulkBase クラスから継承されます。
- SCE のメモリ制限により、バルク サイズのエントリ数は最大 100 に制限されています。

バルク イテレータ

BulkBase クラスには、バルクに含まれるデータを表示するためのイテレータがあります。

次に バルク イテレータの構文を示します。

```
Iterator getIterator()
```

このイテレータを使用すると、さまざまな通知内で SCE から受信されたバルクに対して反復操作を実行したり（logoutBulkIndication、loginPullBulkResponseIndication など）、操作に失敗した場合に、各操作で使用されたデータを検査したりできます。

イテレータにはデータを取得するための次のメソッドがあります。

```
public Object next()
public boolean hasNext()
```

next() メソッドは SubscriberData オブジェクトを返します。

SubscriberData クラスは、バルクに含まれる単一サブスクライバの情報を取得する場合に使用します。

SubscriberData

SubscriberData クラスは、特定のサブスクライバに実行できるすべての操作を表します。SubscriberData クラスには情報を取得するための次のユーティリティ メソッドが含まれています。

```
public String getSubscriberID()
public String getAnonymousID()
public String[] getMappings()
public short[] getTypes()
public boolean getAdditiveFlag()
```

ここでは、各 API 操作で使用できるさまざまなバルク データの型について説明します。

Login_BULK クラス

このクラスはサブスクライバのバルクを表し、**loginBulk** 操作に必要なデータをすべて含みます。

コンストラクタ

データで満たされた **Login_BULK** を構築するには、次のコンストラクタを使用します。

```
public Login_BULK(String[] subscriberIDs,
NetworkID[] networkIDs,
boolean[]networkIDsAdditive,
PolicyProfile[] policy,
QuotaOperation[] quota)
```

パラメータ

subscriberID ― サブスクライバの一意の ID。サブスクライバ ID のフォーマットについては、「サブスクライバ ID」（p.4-2）を参照してください。

networkID ― サブスクライバのネットワーク ID。詳細は「ネットワーク ID のマッピング」（p.4-2）を参照してください。

networkIDAdditive ― このフラグが TRUE に設定されている場合は、サブスクライバの既存の networkID に、指定した networkID が追加されます。それ以外の場合は、指定した networkID で既存の networkID が置き換えられます。

policy ― サブスクライバのポリシー プロファイル。詳細は「SCA BB サブスクライバのポリシー プロファイル」（p.4-4）を参照してください。

quota ― サブスクライバのクォータ。詳細は「サブスクライバ クォータ」（p.4-5）を参照してください。

空の Login_BULK を構築するには、次のメソッドを使用します。

```
public Login_BULK()
```

addBulkEntry メソッド

バルクにエントリを追加するには、次のメソッドを使用します。

```
public void addBulkEntry(String subscriberID,
NetworkID networkID,
boolean networkIdsAdditive,
PolicyProfile policy,
QuotaOperation quota)
```

パラメータ

subscriberID ― サブスクライバの一意の ID。サブスクライバ ID のフォーマットについては、「サブスクライバ ID」（p.4-2）を参照してください。

networkID ― サブスクライバのネットワーク ID。詳細は「ネットワーク ID のマッピング」（p.4-2）を参照してください。

networkIDAdditive ― このフラグが TRUE に設定されている場合は、サブスクライバの既存の networkID に、指定した networkID が追加されます。それ以外の場合は、指定した networkID で既存の networkID が置き換えられます。

policy ― サブスクライバのポリシー プロファイル。詳細は「SCA BB サブスクライバのポリシー プロファイル」（p.4-4）を参照してください。

quota ― サブスクライバのクォータ。詳細は「サブスクライバ クォータ」（p.4-5）を参照してください。

例

Login_BULK オブジェクトの使用方法：例

次に、Login_BULK オブジェクトの使用例を示します。

```
// バルク構築用のデータをすべて用意します。
String[] names = new String[5];
NetworkID[] mappings = new NetworkID[5];
boolean[] additive = new boolean[5];
PolicyProfile[] policy = new PolicyProfile[5];

for (int i=0; i<5; i++)
{
names[i]="sub_"+i;
mappings[i] = new NetworkID(”1.1.1.”+i,NetworkID.TYPE_IP);
additive[i] = true;
policy[i] = new PolicyProfile(new String[]{"packageId="+(i+1)});
}
// バルク オブジェクトを構築します。
Login_BULK bulk = new Login_BULK(names,mappings,additive,policy,null);
// これで loginBulk 操作でバルク オブジェクトを使用できます。
sceApi.loginBulk(bulk,null);
```

Login_BULK の操作：例

次の例は、Login_BULK オブジェクトの別の操作方法を示しています。

```
// 空のバルクを構築します。
Login_BULK bulk = new Login_BULK ();
// addBulkEntry メソッドを使用してバルクにデータを設定します。
for (int i=0; i<20; i++)
{
String name ="sub_"+i;
NetworkID mappings = new NetworkID(i+1);
boolean additive = true;
PolicyProfile policy = new PolicyProfile(
new String[]{"packageId="+(i+1)});
QuotaOperation quota = new SCAS_BB_QuotaOperation(
new String[]{“1”,”2”,”3”},
new long[]{80,80,0}
new short[]{SCAS_BB_QuotaOperation.ADD_QUOTA_OPERATION,
SCAS_BB_QuotaOperation.ADD_QUOTA_OPERATION,
SCAS_BB_QuotaOperation.SET_QUOTA_OPERATION});
bulk.addBulkEntry(name,mappings,additive,policy,quota);
}
// これで loginBulk 操作でバルク オブジェクトを使用できます。
sceApi.loginBulk(bulk,null);
```

SubscriberID_BULK クラス

入力するサブスクライバ ID のみを必要とする **logoutBulkIndication** コールバック関数では、SubscriberID_BULK クラスを使用します。「logoutBulkIndication コールバック メソッド」（p.5-10）を参照してください。

コンストラクタ

サブスクライバ ID データを含む **SubscriberID_BULK** を構築するには、次のコンストラクタを使用します。

```
public SubscriberID_BULK(String[] subscriberIDs)
```

空の **SubscriberID_BULK** を構築するには、次のメソッドを使用します。

```
public SubscriberID_BULK()
```

パラメータ

subscriberID — サブスクライバの一意の ID。サブスクライバ ID のフォーマットについては、「サブスクライバ ID」（p.4-2）を参照してください。

addBulkEntry メソッド

SubscriberID バルクにエントリを追加するには、次のメソッドを使用します。

```
addBulkEntry(String subscriberID)
```

パラメータ

subscriberID — サブスクライバの一意の ID。サブスクライバ ID のフォーマットについては、「サブスクライバ ID」（p.4-2）を参照してください。

NetworkAndSubscriberID_BULK クラス

次の操作では、サブスクライバ ID および NetworkID を必要とするバルク操作に NetworkAndSubscriberID_BULK クラスを使用します。

- **getSubscribersBulkResponse** コールバック（「LoginPullListener インターフェイス クラス」[p.5-8] を参照）
- **logoutBulk** 操作（「logoutBulk 操作」[p.5-25] を参照）
- **networkIDUpdateBulk** 操作（「networkIdUpdateBulk 操作」[p.5-27] を参照）

コンストラクタ

SubscriberID および NetworkID データを含む **NetworkAndSubscriberID_BULK** を構築するには、次のコンストラクタを使用します。

```
public NetworkAndSubscriberID_BULK(String[] subscriberIDs,
NetworkID[] networkIDs,
boolean[] netIdAdditive)
```

空の **NetworkAndSubscriberID_BULK** を構築するには、次のメソッドを使用します。

```
public NetworkAndSubscriberID_BULK()
```

パラメータ

subscriberID ― サブスクライバの一意の ID。サブスクライバ ID のフォーマットについては、「サブスクライバ ID」（p.4-2）を参照してください。

networkID ― サブスクライバのネットワーク ID。詳細は「ネットワーク ID のマッピング」（p.4-2）を参照してください。

networkIDAdditive ― このフラグが TRUE に設定されている場合は、サブスクライバの既存の networkID に、指定した networkID が追加されます。それ以外の場合は、指定した networkID で既存の networkID が置き換えられます。

addBulkEntry メソッド

バルクにエントリを追加するには、次のメソッドを使用します。

```
addBulkEntry(String subscriberID,
NetworkID networkID,
boolean netIdAdditive)
```

パラメータ

subscriberID ― サブスクライバの一意の ID。サブスクライバ ID のフォーマットについては、「サブスクライバ ID」（p.4-2）を参照してください。

networkID ― サブスクライバのネットワーク ID。詳細は「ネットワーク ID のマッピング」（p.4-2）を参照してください。

networkIDAdditive ― このフラグが TRUE に設定されている場合は、サブスクライバの既存の networkID に、指定した networkID が追加されます。それ以外の場合は、指定した networkID で既存の networkID が置き換えられます。

LoginPullResponse_BULK クラス

このクラスはサブスクライバのバルクを表し、**loginPullResponseBulk** メソッドに必要なデータをすべて含みます。

コンストラクタ

関連データを含む **LoginPullResponse_BULK** を構築するには、次のコンストラクタを使用します。

```
public LoginPullResponse_BULK(String[] anonymousSubscriberIDs,
String[] subscriberIDs,
NetworkID[] networkIDs,
boolean[] networkIdsAdditive,
PolicyProfile[] policy,
QuotaOperation[] quota)
```

空の **LoginPullResponse_BULK** を構築するには、次のメソッドを使用します。

```
public LoginPullResponse_BULK()
```

- パラメータ（p.4-13）

パラメータ

anonymousSubscriberID ― アノニマス サブスクライバの ID。この ID は loginPullRequest/loginPullBulkRequest 通知に格納されて、SCE から送信されます（「loginPullRequest コールバック メソッド」[p.5-8] および「loginPullRequestBulk コールバック メソッド」[p.5-9] を参照）。詳細は「サブスクライバ統合モデル」（p.2-3）を参照してください。

subscriberID ― サブスクライバの一意の ID。サブスクライバ ID のフォーマットについては、「サブスクライバ ID」（p.4-2）を参照してください。

networkID ― サブスクライバのネットワーク ID。詳細は「ネットワーク ID のマッピング」（p.4-2）を参照してください。

networkIDAdditive ― このフラグが TRUE に設定されている場合は、サブスクライバの既存の networkID に、指定した networkID が追加されます。それ以外の場合は、指定した networkID で既存の networkID が置き換えられます。

policy ― サブスクライバのポリシー プロファイル。詳細は「SCA BB サブスクライバのポリシー プロファイル」（p.4-4）を参照してください。

quota ― サブスクライバのクォータ。詳細は「サブスクライバ クォータ」（p.4-5）を参照してください。

addBulkEntry メソッド

バルクにエントリを追加するには、次のメソッドを使用します。

```
public addBulkEntry(String anonymousSubscriberID,
String subscriberID,
NetworkID networkID,
boolean networkIdAdditive,
PolicyProfile policy,
QuotaOperation quota)
```

パラメータ

anonymousSubscriberID ― アノニマス サブスクライバの ID。この ID は loginPullRequest/loginPullBulkRequest 通知に格納されて、SCE から送信されます（「loginPullRequest コールバック メソッド」[p.5-8] および「loginPullRequestBulk コールバック メソッド」[p.5-9] を参照）。詳細は「サブスクライバ統合モデル」（p.2-3）を参照してください。

subscriberID ― サブスクライバの一意の ID。サブスクライバ ID のフォーマットについては、「サブスクライバ ID」（p.4-2）を参照してください。

networkID ― サブスクライバのネットワーク ID。詳細は「ネットワーク ID のマッピング」（p.4-2）を参照してください。

networkIDAdditive ― このフラグが TRUE に設定されている場合は、サブスクライバの既存の networkID に、指定した networkID が追加されます。それ以外の場合は、指定した networkID で既存の networkID が置き換えられます。

policy ― サブスクライバのポリシー プロファイル。詳細は「SCA BB サブスクライバのポリシー プロファイル」（p.4-4）を参照してください。

quota ― サブスクライバのクォータ。詳細は「サブスクライバ クォータ」（p.4-5）を参照してください。

PolicyProfile_BULK クラス

updatePolicyProfileBulk 操作ではこのクラスを使用して、サブスクライバ ID およびサブスクライバ ポリシー プロファイルのバルクを表します。

- コンストラクタ（p.4-14）
- addBulkEntry メソッド（p.4-14）

コンストラクタ

関連データを含む **PolicyProfile_BULK** を構築するには、次のコンストラクタを使用します。

```
public PolicyProfile_BULK(String[] subscriberIDs, PolicyProfile[] policy)
```

空の **PolicyProfile_BULK** を構築するには、次のメソッドを使用します。

```
public PolicyProfile_BULK()
```

パラメータ

subscriberID ― サブスクライバの一意の ID。サブスクライバ ID のフォーマットについては、「サブスクライバ ID」（p.4-2）を参照してください。

policy ― サブスクライバのポリシー プロファイル。詳細は「SCA BB サブスクライバのポリシー プロファイル」（p.4-4）を参照してください。

addBulkEntry メソッド

バルクにエントリを追加するには、次のメソッドを使用します。

```
public addBulkEntry(String subscriberID,  PolicyProfile policy)
```

パラメータ

subscriberID ― サブスクライバの一意の ID。サブスクライバ ID のフォーマットについては、「サブスクライバ ID」（p.4-2）を参照してください。

policy ― サブスクライバのポリシー プロファイル。詳細は「SCA BB サブスクライバのポリシー プロファイル」（p.4-4）を参照してください。

Quota_BULK クラス

次の操作ではこのクラスを使用して、サブスクライバ ID およびクォータ バケットのバルクを表します。

- getQuotaStatusBulk 操作（バケット ID のみを指定）
- quotaStatusBulkIndication コールバック メソッド
- quotaDepletedBulkIndication コールバック メソッド
- quotaBelowThresholdIndication コールバック メソッド

コンストラクタ

関連データを含む **Quota_BULK** を構築するには、次のコンストラクタを使用します。

```
public Quota_BULK(String[] subscriberIDs, Quota[] subscribersQuota)
```

空の Quota_BULK を構築するには、次のメソッドを使用します。

```
public Quota_BULK()
```

パラメータ

subscriberID — サブスクライバの一意の ID。サブスクライバ ID のフォーマットについては、「サブスクライバ ID」（p.4-2）を参照してください。

quota — サブスクライバのクォータ。詳細は「サブスクライバ クォータ」（p.4-5）を参照してください。

addBulkEntry メソッド

バルクにエントリを追加するには、次のメソッドを使用します。

```
public addBulkEntry(String subscriberID,Quota quota)
```

パラメータ

subscriberID — サブスクライバの一意の ID。サブスクライバ ID のフォーマットについては、「サブスクライバ ID」（p.4-2）を参照してください。

quota — サブスクライバのクォータ。詳細は「サブスクライバ クォータ」（p.4-5）を参照してください。

QuotaOperation_BULK クラス

QuotaUpdateBulk 操作およびログイン操作ではこのクラスを使用して、サブスクライバ ID およびサブスクライバ クォータ操作のバルクを表します。

コンストラクタ

関連データを含む **QuotaOperation_BULK** を構築するには、次のコンストラクタを使用します。

```
public QuotaOperation_BULK(String[] subscriberIDs,
QuotaOperation[]quotaOperations)
```

空の QuotaOperation_BULK を構築するには、次のメソッドを使用します。

```
public QuotaOperation_BULK()
```

パラメータ

subscriberID — サブスクライバの一意の ID。サブスクライバ ID のフォーマットについては、「サブスクライバ ID」（p.4-2）を参照してください。

quotaOperation ― サブスクライバのクォータに対して実行するクォータ処理。詳細は「サブスクライバ クォータ」（p.4-5）を参照してください。

addBulkEntry メソッド

バルクにエントリを追加するには、次のメソッドを使用します。

```
addBulkEntry(String subscriberID, QuotaOperation quotaOperation)
```

パラメータ

subscriberID ― サブスクライバの一意の ID。サブスクライバ ID のフォーマットについては、「サブスクライバ ID」（p.4-2）を参照してください。

quotaOperation ― サブスクライバのクォータに対して実行するクォータ処理。詳細は「サブスクライバ クォータ」（p.4-5）を参照してください。

CHAPTER 5

SCE Subscriber API を使用したプログラミング

この章では、API プログラミングの構造、クラス、メソッド、およびインターフェイスについて詳細に説明します。

API クラス

次のリストに、API が提供するクラスのマッピングを示します。

- パッケージ com.scms.api.sce.prpc（p.5-2）
- パッケージ com.scms.api.sce（p.5-2）
- パッケージ com.scms.common（p.5-2）

パッケージ com.scms.api.sce.prpc

PRPC_SCESubscriberApi クラス（p.5-4）— 主要な API クラス

パッケージ com.scms.api.sce

- 通知リスナ（p.5-2）
- 接続モニタリング（p.5-2）
- SCE カスケード トポロジのサポート（p.5-2）
- 操作結果の処理（p.5-2）

通知リスナ

- LoginPullListener インターフェイス クラス（p.5-8）
- LogoutListener インターフェイス クラス（p.5-10）
- QuotaListenerEx インターフェイス クラス（p.5-10）

接続モニタリング

- ConnectionListener インターフェイス（p.5-14）

SCE カスケード トポロジのサポート

- RedundancyStateListener インターフェイス（p.5-15）

操作結果の処理

- OperationException クラス（p.5-19）
- SCESubscriberApi（インターフェイス）— OperationException 内部で受信できるエラー コード定数を含みます。
- OperationArguments クラス（p.5-18）
- OperationResultHandler インターフェイス（p.5-17）

パッケージ com.scms.common

com.scms.common パッケージには、API で使用されるすべてのデータ型が含まれています。

- Login_BULK クラス（p.4-9）
- LoginPullResponse_BULK クラス（p.4-12）
- NetworkAndSubscriberID_BULK クラス（p.4-11）
- PolicyProfile_BULK クラス（p.4-14）

- SubscriberID_BULK クラス（p.4-11）
- SubscriberData（p.4-8）
- SCAS_BB_Quota（p.4-6）
- SCAS_BB_QuotaOperation（p.4-7）
- ネットワーク ID のマッピング（p.4-2）
- PolicyProfile クラス（p.4-4）

プログラミングに関する注意事項

- コールバック メソッドを使用したプログラミング（p.5-3）

コールバック メソッドを使用したプログラミング

ここまで説明したように、API 操作の多くはコールバック メソッドに基づいています。ユーザは、特定のイベントが発生すると呼び出される「リスナ」を指定します。ここでは、コールバック メソッドを使用したプログラミングに関する主な注意事項を定義します。

コールバック メソッドのスレッド内では長い操作を実行しないでください。長い操作は**別のスレッド**から実行する必要があります。この推奨事項に従わないと、クライアント側でリソースが不足することがあります。

この注意事項は次の操作に適用されます。

- LoginPullListener コールバック メソッド
- LogoutListener コールバック メソッド
- QuotaListenerEx コールバック メソッド
- ConnectionListener コールバック メソッド

PRPC_SCESubscriberApi クラス

PRPC_SCESubscriberAPI クラス（com.scms.sce.api.prpc パッケージ上）は、次の機能を提供する主要な API クラスです。

- API の構築
- ただ 1 つの SCE と API の接続（接続属性の設定）
- 通知リスナの登録 / 登録解除
- 接続リスナの設定
- サブスクライバ プロビジョニング操作の実行
- SCE からの切断

API の構築

PRPC_SCESubscriberAPI には次のコンストラクタがあります。

構文：

```
public PRPC_SCESubscriberApi(String apiName, String sceHost)
throws UnknownHostException
public PRPC_SCESubscriberApi(String apiName,
String sceHost,
long autoReconnectInterval)
throws UnknownHostException
public PRPC_SCESubscriberApi(String apiName,
String sceHost,
int scePort,
long autoReconnectInterval)
throws UnknownHostException
```

パラメータ：

次に、API コンストラクタのコンストラクタ引数の説明を示します。

apiName ― API 名を指定します。

（注） API 名は SCE ごとに一意である必要があります。同じ名前を持つ API を複数構築して、単一の SCE に接続した場合、その SCE プラットフォームは複数の API を 1 つの API クライアントとして処理します。この機能は、ハイ アベイラビリティがサポートされている場合にのみ使用してください。ハイ アベイラビリティの詳細については、「ハイ アベイラビリティの実装」（p.5-39）を参照してください。

sceHost ― IP アドレスまたは到達可能なホスト名のいずれかになります。

scePort ― SCE に接続する PRPC プロトコル TCP ポート（デフォルト値は 14374）

autoReconnectInterval ― 再接続タスクで再接続を試行するインターバル（ミリ秒単位）を、次のように定義します。

- 値が 0 以下の場合、再接続タスクはアクティブになりません（自動再接続は試行されません）。
- 値が 0 より大きくて、接続障害が発生している場合、再接続タスクは <autoReconnectInterval> ミリ秒ごとにアクティブになります。
- 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イネーブルにするには、API の **connect** メソッドを 1 回以上呼び出す**必要があります**。

例

次のコードは、10 秒間隔で自動再接続を行って、API を構築します。

```
PRPC_SCESuscriberAPI sceApi = new PRPC_SCESuscriberAPI("MyApi",
"10.1.1.1",
10000);
sceApi.connect();
```

次のコードは、自動再接続をサポートしないで、API を構築します。

```
PRPC_SCESuscriberAPI sceApi = new PRPC_SCESuscriberAPI("MyApi",
"10.1.1.1");
sceApi.connect();
```

リスナ セットアップ操作

API を初期化したら、API を使用するアプリケーションのタイプ、および使用するトポロジに基づいて、API に利用対象リスナを設定する必要があります。トポロジの詳細については、「サポート対象トポロジ」（p.2-6）を参照してください。

リスナ セットアップ操作の内容は、次のとおりです。

- 接続リスナの設定（詳細は「接続モニタリング」[p.5-14] を参照）
- `public void setConnectionListener(ConnectionListener listener)`
- ログインプル リスナの設定(詳細は「LoginPullListener インターフェイス クラス」[p.5-8] を参照）
- `public void registerLoginPullListener(LoginPullListener listener)`
- ログアウト リスナの設定（詳細は「LogoutListener インターフェイス クラス」[p.5-10] を参照）
- `public void registerLogoutListener(LogoutListener listener)`
- クォータ リスナの設定（詳細は「QuotaListenerEx インターフェイス クラス」[p.5-10] を参照）
- `public void registerQuotaListener(QuotaListener listener)`
- 冗長ステート リスナの設定（詳細は「RedundancyStateListener インターフェイス」[p.5-15] を参照）
- `public void setRedundancyStateListener(RedundancyStateListener listener)`

(注)　API にリスナを登録すると、SCE 内でリソースが割り当てられて、リスナへの確実なメッセージ配信がサポートされます。API を使用するアプリケーションがクラッシュし、少しあとに再起動した場合でも、メッセージは保持され、API 再接続時に SCE に送信されます。

高度なセットアップ操作

API を使用すると、特定の内部プロパティを初期化して、API をカスタマイズできます。初期化には API **init** メソッドを使用します。

(注)　設定を有効にするには、**connect** メソッドより先に **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 ― SCE に送信されるまで API で蓄積できる要求の最大数を定義する内部バッファ サイズ（デフォルト：1024)
- 処理のタイムアウト ― 応答しない PRPC プロトコル接続に関する希望のタイムアウト間隔（ミリ秒）（デフォルト：45 秒）

構文

init メソッドの構文は、次のとおりです。

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

パラメータ

properties（java.util. プロパティ）― 「高度なセットアップ操作」（p.5-5）に記載されているプロパティの設定をイネーブルにします。

- 出力キュー サイズを設定するには、プロパティ キーとして **prpc.client.output.machinemode.recordnum** を使用します。
- 操作のタイムアウトを設定するには、プロパティ キーとして **com.scms.api.sce.prpc.regularInvocationTimeout** または **com.scms.api.sce.prpc.listenerInvocationTimeout** を使用します。

（注） **com.scms.api.sce.prpc.listenerInvocationTimeout** は、リスナ コールバックから呼び出すことができる操作に使用します。デッドロックを回避するために、このタイムアウトは **com.scms.api.sce.prpc.regularInvocationTimeout** よりも短く設定する必要があります。

プロパティのカスタマイズ例

初期化中にプロパティをカスタマイズする例を示します。

```
// API の構築
PRPC_SCESuscriberAPI sceApi = new PRPC_SCESuscriberAPI("MyApi",
"10.1.1.1",10000);
// API の初期化
java.util.Properties p = new java.util.Properties();
p.setProperty("prpc.client.output.machinemode.recordnum", 2048+"");
api.init(p);
// API との接続
sceApi.connect();
```

（注） **init** メソッドは、**connect** メソッドよりも**先に**呼び出されます。

SCE との接続

API を設定したら、SCE との接続を試行する必要があります。自動再接続機能がアクティブな場合、これ以降に発生する切断はすべて API で処理されます。

SCE に接続するには、次のメソッドを使用します。

```
public void connect() throws Exception
```

API 操作中のどの時点でも、メソッド **isConnected()** を使用して、API が SCE に接続されているかどうかを確認できます。

```
public boolean isConnected()
```

（注） どの API インスタンスも、ただ 1 つの SCE プラットフォームとの接続をサポートしています。

getApiVersion

構文

```
public String getApiVersion()
```

説明

このメソッドは API バージョンを問い合わせます。バージョンは <Major Version.Minor Version> としてフォーマットされたストリングです。

API の終了

サーバおよびクライアントのリソースを解放するには、**disconnect** メソッドを呼び出します。

```
public void disconnect()
```

disconnect メソッドを呼び出すと、SCE と API との接続の信頼性を管理する SCE 内のリソースが解放されます。アプリケーションが再起動していて、メッセージを失いたくない場合は、disconnect メソッドを使用しないでください。

main クラス内で **finally** 文を使用することを推奨します。次に例を示します。

```
public static void main(String [] args) throws Exception
{
PRPC_SCESubscriberApi sceapi = new PRPC_SCE_SubscriberApi ("myApi",
"sceHost");
try
{
                …
                   // コードをここに記述します。
        }
        finally
        {
                sceapi.disconnect();
        }
}
```

通知リスナ

特定のイベントが発生すると、SCE プラットフォームは複数のタイプの通知を発行します。次の 3 つの通知タイプがあります。

- ログインプル通知
- ログアウト通知
- クォータ通知

通知が送信されるのは、これらの通知を待ち受けるように登録されたリスナが存在する場合のみです。通知タイプごとに異なるリスナを登録できます。これらの通知をトリガーするイベントの詳細については、「API イベント」（p.3-1）の章を参照してください。

LoginPullListener インターフェイス クラス

LoginPullListener インターフェイスは、**プル** モデルでのみ使用される一連のコールバック関数を定義します。

プル モデルでの動作を想定しているポリシー サーバで、サブスクライバ プロビジョニング プロセスのネットワーク ID 管理を行う場合は、このポリシー サーバで LoginPullListener を登録して、SCE からのログインプル要求への応答や、SCE プラットフォームの同期化を実現する必要があります。

これらの通知の待ち受けをイネーブルにするために、API では次の通知タイプに対応するようにリスナを設定できます。

```
public void registerLoginPullListener(LoginPullListener listener)
public void unregisterLoginPullListener(LoginPullListener listener)
```

（注） API でサポートされる LoginPullListener は一度に 1 つずつです。また、複数の API で同じ LoginPullListener を登録することも避けてください。両方の SCE が同じログインプル要求に応答した場合に、SCE プラットフォームが同期されなくなることがあります。

LoginPullListener は、ログインプル通知リスナを登録できるようにする場合に実装するインターフェイスです。このインターフェイスは次のように定義されます。

```
public interface LoginPullListener
{
public void loginPullRequest (String anonymousSubscriberID,
NetworkID networkID)
public void loginPullRequestBulk(NetworkAndSubscriberID_BULK  subs)

public void getSubscribersBulkResponse(
NetworkAndSubscriberID_BULK  subs,
SubscriberBulkResponseIterator iterator)
}
```

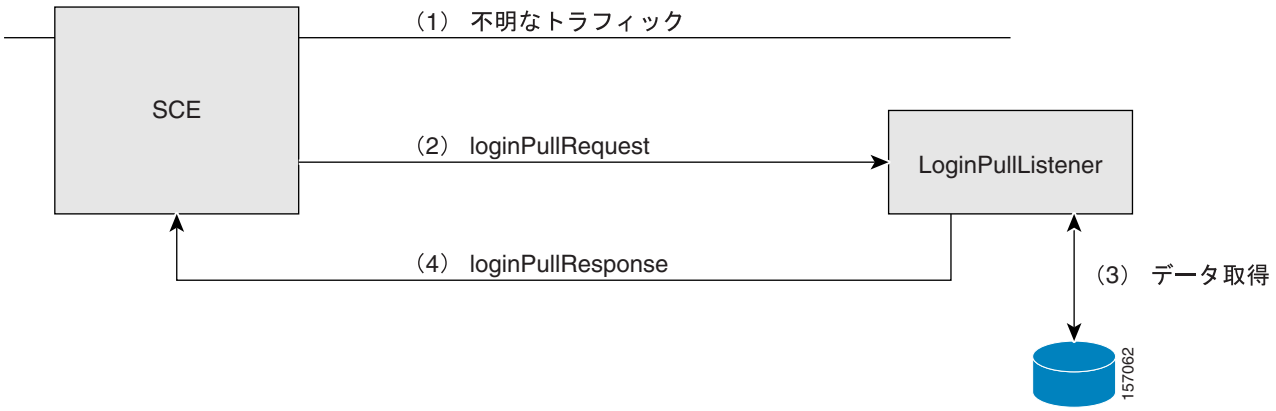
loginPullRequest コールバック メソッド

SCE で IP アドレスが不明なサブスクライバ側トラフィックが検出されると、SCE は IP アドレスに基づいてサブスクライバ ログイン情報要求を送信します（「プル モデル」[p.3-3] を参照）。通常、ポリシー サーバはこの IP が割り当てられたサブスクライバ データの設定データを使用して応答します。

この要求は登録済みリスナにディスパッチされ、**loginPullRequest** コールバック関数をトリガーします。このコールバック時に、リスナはこの IP アドレスに一致するサブスクライバのサブスクライバ情報を取得し、**loginPullResponse** をアクティブにして、その情報を SCE に配信する必要があります（「loginPullResponse 操作」[p.5-23] を参照）。この IP アドレスに対応する情報が存在しない場合、応答は発行されません。

次の図に、**loginPullRequest** コールバック メソッドを示します。

図 5-1　loginPullRequest コールバック メソッド



パラメータ

- **anonymousSubscriberID** ― **loginPullResponse** 操作には、このアノニマス サブスクライバ ID を指定する**必要があります**（「loginPullResponse 操作」[p.5-23] を参照）。「アノニマス サブスクライバ ID」（p.2-2）も参照してください。
- **networkID** ― 不明なサブスクライバのネットワーク ID。詳細は「ネットワーク ID」（p.2-2）を参照してください。

loginPullRequestBulk コールバック メソッド

このコールバック関数は、上記の **loginPullRequest** コールバック関数のバルク バージョンです。

パラメータ

- **subs** ― 複数のサブスクライバの NetworkID とアノニマス ID のペアを含みます。詳細は、loginPullRequest コールバック メソッドの「パラメータ」（p.5-9）を参照してください。

ポリシー サーバがこの要求に応答するには、loginPullBulkResponse メソッドをアクティブにするか、バルク内のネットワーク ID ごとに loginPullResponse メソッドをアクティブにします。「loginPullResponseBulk 操作」（p.5-24）および「loginPullResponse 操作」（p.5-23）を参照してください。subs パラメータ内のデータに関して反復操作を実行するには、バルク クラスから提供される next() 反復メソッドを使用します（「バルク イテレータ」[p.4-8] を参照）。

GetSubscribersBulkResponse コールバック メソッド

このコールバック メソッドは、**プル** モデル内の SCE 同期プロセス中に使用します。詳細は、「SCE-API の同期」（p.5-33）を参照してください。

LogoutListener インターフェイス クラス

サブスクライバ プロビジョニング プロセスのネットワーク ID 管理操作を行うポリシー サーバでは、特定のサブスクライバが SCE プラットフォームから実際に削除された場合に通知される LogoutListener を登録しなければならないことがあります。

API を使用すると、ログアウト通知を受信できるように **LogoutListener** を設定できます。

```
public void registerLogoutListener(LogoutListener listener)
public void unregisterLogoutListener(LogoutListener listener)
```

(注)　API でサポートされる LogoutListener は一度に 1 つずつです。

ここでは、**LogoutListener** インターフェイスのコールバック関数について説明します。

- logoutIndication コールバック メソッド（p.5-10）
- logoutBulkIndication コールバック メソッド（p.5-10）

logoutIndication コールバック メソッド

subscriberID で識別されるサブスクライバの最終 Network ID のログアウトを識別すると、SCE プラットフォームは、ログアウト通知を発行します。これにより、登録されたすべてのログアウト通知リスナに関する **logoutIndication** コールバック関数が呼び出されます。

```
public void logoutIndication(String subscriberID)
```

パラメータ

- **subscriberID** ― サブスクライバの一意の ID。詳細は、「サブスクライバ ID」（p.4-2）を参照してください。SCE はこのサブスクライバ ID を処理しなくなります。

logoutBulkIndication コールバック メソッド

サブスクライバ グループの最終 Network ID のログアウトを識別すると、SCE プラットフォームはログアウト バルク通知を発行します。これにより、登録されたすべてのログアウト通知リスナに関する **logoutBulkIndication** コールバック関数が呼び出されます。

```
public void logoutBulkIndication(SubscriberID_BULK subs)
```

パラメータ

- **subs** ― ログアウトしたサブスクライバのサブスクライバ ID を含みます。詳細は「SubscriberID_BULK クラス」（p.4-11）を参照してください。

QuotaListenerEx インターフェイス クラス

(注)　バージョン 3.0.5 以降、QuotaListener インターフェイスは廃止されています。代わりに、QuotaListenerEx を使用してください。下位互換性を保つために QuotaListener インターフェイスは存続していますが、バージョン 3.0.5 の API と統合する場合は、QuotaListenerEx インターフェイスを使用する必要があります。

サブスクライバ プロビジョニング プロセスでクォータ管理操作を行うポリシー サーバでは、SCE プラットフォームから発行されるクォータ関連通知を受信できなければなりません。

API を使用すると、クォータ通知を受信できるように **QuotaListener** を設定できます。

```
public void registerQuotaListener(QuotaListener listener)
public void unregisterQuotaListener(QuotaListener listener)
```

(注)　API でサポートされる QuotaListener は一度に 1 つずつです。

(注)　下位互換性を保つために QuotaListener インターフェイスが使用されていますが、QuotaListenerEx を実装するオブジェクトを渡すことを推奨します。

ここでは、**QuotaListenerEx** インターフェイスのコールバック関数について説明します。

(注)　クォータ コールバック メソッドのバルク バージョンは、このリリースの API では使用されません。

quotaStatusIndication コールバック メソッド

クォータ ステータス通知は、特定のサブスクライバに特定のクォータ バケット セットの残りの値を配信します。この通知は SCE によって定期的に発行されるか、または **getQuotaStatus** 操作が呼び出された場合に発行され（「getQuotaStatus 操作」[p.5-31] を参照）、**quotaStatusIndication** コールバック関数を起動すると登録済みリスナに配信されます。

```
public void quotaStatusIndication(String subscriberID,
Quota quota)
```

パラメータ

- **subscriberID** ― サブスクライバの一意の ID。詳細は「サブスクライバ ID」（p.4-2）を参照してください。
- **quota** ― サブスクライバのクォータ。詳細は「サブスクライバ クォータ」（p.4-5）を参照してください。

quotaStatusBulkIndication コールバック メソッド

クォータ ステータス バルク通知は、サブスクライバ グループに特定のクォータ バケット セットの残りの値を配信します。この通知は SCE によって定期的に発行されるか、または **getQuotaStatus** 操作が呼び出された場合に発行され（「クォータ ステータス取得イベント」[p.3-6] を参照）、**quotaStatusIndication** コールバック関数を起動すると登録済みリスナに配信されます。

```
public void quotaStatusBulkIndication(Quota_BULK subs)
```

定期的に発行される通知の期間を設定できます。詳細については、『*Cisco Service Control Application for Broadband User Guide*』を参照してください。

パラメータ

- **subs** ― サブスクライバのバルクのクォータ データを含みます。詳細は「Quota_BULK クラス」（p.4-14）を参照してください。

quotaBelowThresholdIndication コールバック メソッド

サブスクライバのクォータが設定済みしきい値よりも低下すると、SCE プラットフォームは **quotaBelowThresholdIndication** コールバック関数を起動して、登録済みリスナに配信される通知を発行します。

```
public void quotaBelowThresholdIndication(String subscriberID,
Quota quota)
```

パラメータ

- **subscriberID** ― サブスクライバの一意の ID。詳細は「サブスクライバ ID」（p.2-2）を参照してください。
- **quota** ― サブスクライバのクォータ。詳細は「サブスクライバ クォータ」（p.4-5）を参照してください。

quotaBelowThresholdIBulkndication コールバック メソッド

サブスクライバ グループのクォータが設定済みしきい値よりも低下すると、SCE プラットフォームは **quotaBelowThresholdBulkIndication** コールバック関数を起動して、登録済みリスナに配信される通知を発行します。

```
public void quotaBelowThresholdBulkIndication(Quota_BULK subs)
```

パラメータ

- **subs** ― サブスクライバのバルクのクォータ データを含みます。詳細は「Quota_BULK クラス」（p.4-14）を参照してください。

quotaDepletedIndication コールバック メソッド

サブスクライバのクォータが枯渇すると、SCE プラットフォームは **quotaDepletedIndication** コールバック関数を起動して、登録済みリスナに配信される通知を発行します。

```
public void quotaDepletedIndication(String subscriberID,
Quota quota)
```

パラメータ

- **subscriberID** ― サブスクライバの一意の ID。詳細は「サブスクライバ ID」（p.2-2）を参照してください。
- **quota** ― サブスクライバのクォータ。詳細は「サブスクライバ クォータ」（p.4-5）を参照してください。

quotaDepletedBulkIndication コールバック メソッド

サブスクライバ グループのクォータが枯渇すると、SCE プラットフォームは **quotaDepletedBulkIndication** コールバック関数を起動して、登録済みリスナに配信される通知を発行します。

```
public void quotaDepletedBulkIndication (SubscriberID_BULK subs)
```

パラメータ

- **subs** ― クォータが枯渇したサブスクライバの名前を含みます。詳細は「SubscriberID_BULK クラス」（p.4-11）を参照してください。

quotaStateRestore コールバック メソッド

サブスクライバがポリシー サーバにログインすると、ポリシー サーバは SCE へのログイン操作を実行します。SCE は **quotaStateRestore** コールバック関数を起動して、SCE 内のサブスクライバ クォータを復元するための要求をポリシー サーバに発行します。ポリシー サーバは「クォータ更新イベント」（p.3-5）を使用して、この関数に応答しなければなりません。

```
public void quotaStateRestore(String subscriberID,
Quota quota)
```

パラメータ

- **subscriberID** ― サブスクライバの一意の ID。詳細は「サブスクライバ ID」（p.2-2）を参照してください。
- **quota** ― サブスクライバのクォータ。詳細は「サブスクライバ クォータ」（p.4-5）を参照してください。この通知が作成された時点ですべてのクォータ バケットが空であるため、バケット ID 配列のサイズは 0 です。

quotaStateBulkRestore コールバック メソッド

サブスクライバ グループがポリシー サーバにログインすると、ポリシー サーバは SCE へのログイン操作を実行します。SCE は **quotaStateBulkRestore** コールバック関数を起動して、SCE 内のサブスクライバ クォータを復元するための要求をポリシー サーバに発行します。ポリシー サーバは「クォータ更新イベント」（p.3-5）を使用して、この関数に応答しなければなりません。

```
public void quotaStateBulkRestore(SubscriberID_BULK subs)
```

パラメータ

- **subs** ― クォータが枯渇したサブスクライバの名前を含みます。詳細は「SubscriberID_BULK クラス」（p.4-11）を参照してください。

接続モニタリング

SCMS SCE Subscriber API は SCE プラットフォームとの接続をモニタします。SCE との接続確立時または切断時に特定の処理を実行するように要求しているポリシー サーバでは、ConnectionListener インターフェイスを実装できます。

ConnectionListener インターフェイス

API を使用すると、接続リスナを設定できます。

```
setConnectionListener(ConnectionListener listener)
```

接続リスナは、次のように定義されたインターフェイスです。

```
public interface ConnectionListener {
/**
* SCE との接続がダウンしている場合に呼び出されます。
*/
public void connectionIsDown();
/**
* SCE との接続が確立されている場合に呼び出されます。
*/
public void connectionEstablished();
}
```

SCE との同期を開始するには、接続確立コールバックを使用します。詳細は「SCE-API の同期」（p.5-33）を参照してください。

切断リスナ：例

次に、**stdout** にメッセージを出力して処理を戻す切断リスナの単純な実装例を示します。

```
import com.scms.api.sce.ConnectionListener;
public class MyConnectionListener implements ConnectionListener {
public void connectionIsDown(){
System.out.println("Message: connection is down.");
return;
}
public void connectionEstablished(){
System.out.println("Message: connection is established.");
// SCE との同期を開始するスレッドを起動します。
}
}
```

SCE カスケード トポロジのサポート

SCMS SCE Subscriber API は SCE カスケード トポロジをサポートしています。カスケード SCE プラットフォームに接続されているポリシー サーバはカスケード構成内のどの SCE がアクティブでどれがスタンバイなのかを認識する必要があります。ポリシー サーバはアクティブな SCE **だけに** ログオン操作を送信します。同様に、ポリシー サーバがサブスクライバ同期を実行しなければならないのはアクティブな SCE **だけ**です。

スタンバイ SCE は、アクティブ SCE からサブスクライバについて学習します。これによってステートフル フェールオーバーが実現されます。ポリシー サーバは、アップデートされた最新のサブスクライバ情報を取得できるように、フェールオーバー イベントを識別し、アクティブになった SCE と同期をとる必要があります。

アクティブな SCE を知るために、ポリシー サーバには RedundancyStateListener インターフェイスを実装できます。

isRedundancyStatusActive メソッド

API には、SCE の冗長ステートを監視する目的で、RedundancyStateListener インターフェイスと **isRedundnacyStatusActive** メソッドがあります。

```
public boolean isRedundancyStatusActive()
```

このメソッドからの戻り値の意味は次のとおりです。

- TRUE ― SCE の現在の状態がアクティブである場合
- FALSE ― そうでない場合

SCE がアクティブかどうかを検証するために、カスケード SCE の初回の接続時と SCE へのログオン操作の送信前にこのメソッドを使用することを推奨します。

RedundancyStateListener インターフェイス

カスケード SCE の状態変化を監視できるようにするため、この API では冗長ステート リスナの設定が可能です。

```
setRedundancyStateListener(RedundancyStateListener listener)
```

冗長ステート リスナには、カスケード SCE の冗長ステートがアクティブからスタンバイへ、あるいはその逆に変化すると呼び出されるコールバック メソッドを定義します。

冗長ステート リスナは、次のように定義されたインターフェイスです。

```
public interface RedundancyStateListener {
public void redundancyStateChanged(SCESubscriberApi sceApi,
boolean isActive);
}
```

（注） ポリシー サーバは、アクティブになった SCE に対して同期手順を実行する必要があります。これは、SCE への接続確立時にポリシー サーバによって実行される手順と同様です。

(注)　API インスタンスごとに特定の SCE プラットフォームとの接続が確立されます。したがって、カスケードの設定では、2 つの SCE Subscriber API インスタンスが必要です。

パラメータ

- **sceApi** ― 状態が変化した SCE を表す API インスタンス。このパラメータを使用すると、複数の SCE に 1 つのリスナを実装できます。
- **isActive** ― SCE がアクティブになった場合 TRUE になります。SCE が非アクティブになると、FALSE になります。

カスケード違反エラーを無視する SCE の設定

SCE 3.1.0 は、スタンバイ SCE にログオン操作が実行されるとエラーを返すようにデフォルトで設定されています。この動作を変更するには、SCE に `ignore-cascade-violation` CLI を使用します。

カスケード違反を無視するように SCE を設定するには、SCE プラットフォームで次の CLI を使用します。

```
(config)#>management-agent sce-api ignore-cascade-violation
```

カスケード違反が無視されるかどうかを表示するには、SCE プラットフォームで次の CLI を使用します。

```
#>show management-agent sce-api
```

カスケード違反時にエラーを送信するように SCE を設定するには、SCE プラットフォームで次の CLI を使用します。

```
(config)#>no management-agent sce-api ignore-cascade-violation
```

このフラグをデフォルト値（カスケード違反時にエラーを送信する）に設定するには、SCE プラットフォームで次の CLI を使用します。

```
(config)#>default management-agent sce-api ignore-cascade-violation
```

(注)　カスケード違反の無視を SCE に設定するのは、既存の SCE API コードとの下位互換性が必要な場合だけにしてください。カスケード機能を十分に活用するためには、SCE の冗長ステートを監視して使用する必要があります。

結果処理

API を使用すると、**操作ごとに**結果ハンドラを設定して、操作結果を異なる方法で処理できます。

SCE で実行された操作結果が API に戻されると、OperationResultHandler インターフェイスの handleOperationResult コールバックが呼び出されます。

特定の操作で結果処理が不要な場合は、**handler** 引数に **null** を挿入します。

（注） すべての操作に、同じ操作結果ハンドラを渡すことができます。

OperationResultHandler インターフェイス

このインターフェイスは、API を通して実行される操作結果を受信する場合に実装します。

操作結果ハンドラは次の単純なメソッドを使用して呼び出されます。

```
public interface OperationResultHandler {
/**
* 結果を処理します。
*/
public void handleOperationResult(Object[] result,
OperationArguments handback);

}
```

API を通じて実行した操作結果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） OperationResultHandler インターフェイスは結果を取得する唯一の方法です。API メソッドが発信側に返された直後に、結果を返すことはできません。操作結果を受信できるようにするには、処理を呼び出すときに各操作の結果ハンドラを設定します（例を参照）。

次に、OperationResultHandler インターフェイスから返されるデータを示します。

- **result** ― 実際の操作結果。配列内の各エントリは次のいずれかの値をとります。
  - **NULL** ― 操作に成功したことを示します。
  - **OperationException** ― 操作に失敗したことを示します（以下を参照）。非バルク操作の場合、結果配列にはエントリが 1 つのみ含まれます。
- バルク操作の場合、結果配列の各エントリは、バルク操作の関連エントリに対応します。
- **handback** ― 操作を呼び出すたびに、API はこのオブジェクトを自動的に提供します。このオブジェクトには、呼び出し時に渡されたすべての引数など、呼び出された操作に関する情報が含まれています。この操作の入力引数は、API マニュアルに記載された引数名で取得されます。たとえば、このデータを使用して、操作失敗後にパラメータを検査 / 出力したり、操作呼び出しを反復したりできます。

（注） バルク オブジェクトを含む操作では、バルク内の特定の要素に対して操作が失敗した場合でも、バルクが終了するまでバルク処理が継続します。

OperationArguments クラス

処理名を取得するには、次のメソッドを使用します。

```
public String getOperationName()
```

引数名を取得するには、次のメソッドを使用します。

```
public String[] getArgumentNames()
```

特定の操作引数を取得するには、次のメソッドを使用します。引数として、操作シグニチャ内の処理の引数名を使用します。

```
public Object getArgument(String name)
```

例

OperationResultHandler インターフェイスの実装例：

```
public class MyOperationHandler implements OperationResultHandler
{
long sucessCounter = 0;
long errorCounter = 0;

public void handleOperationResult(Object[] result,
OperationArguments handback)
{
for (int index=0; index <result.length; index++)
{
if (result[index]==null)
{
// 成功
successCounter++;
}
else
{
// 失敗
errorCounter++;
// エラーの詳細を抽出
OperationException ex = (OperationException)result[index];
// 操作名を抽出
String operationName = handback.getOperationName();

// 操作名およびエラー メッセージを出力
System.out.println("Error for operation "+
operationName +":" +
ex.getErrorMessage());
// 操作引数を出力
String[] argNames = handback.getArgumentNames();
if (argNames!=null)
{
for (int j=0; j<argNames.length; j++)
{
System.out.println(argNames[j]+  "="+
handback.getArgument(argNames[j]));
}
}
}
}
}
}
```

（注） 上記の実装例は、正規の操作とバルク操作の両方に使用できます。

次に、ログイン操作の結果ハンドラの例を示します。

```
public class LoginOperationHandler implements OperationResultHandler
{

public void handleOperationResult(Object[] result,
OperationArguments handback)
{
for (int index=0; index <result.length; index++)
{
if (result[index]!=null)
{
// 失敗
// エラーの詳細を抽出
OperationException ex =
(OperationException)result[index];
// 操作名およびエラー メッセージを出力
System.out.println("Error for login operation "+
":" + ex.getErrorMessage());
// サブスクライバ ID パラメータ値を出力
System.out.println("subscriberID"+
handback.getArgument("subscriberID"));
}
}
}
}
```

OperationException クラス

com.scms.api.sce.OperationException Java クラスは、通常の Java の使用法と対照的に、SCMS SCE Subscriber API の関数エラーをすべて提供します。このような「対照的な」アプローチが採用されたのは、SCMS SCE Subscriber API に「言語およびプロトコルに依存しない」特性があり、以降に実装するすべての SCE API で同じ外観が必要となるためです（Java、C、C++）。各 OperationException 例外は、次の情報を提供します。

- 一意のエラー コード（**long**）
- 情報メッセージ（**java.lang.String**）
- サーバ側のスタック トレース（**java.lang.String**）

エラー コードおよびその意味についての詳細は、「エラー コードのリスト」（p.A-1）を参照してください。

サブスクライバ プロビジョニング操作

ここでは、サブスクライバ プロビジョニングのために使用できる API のメソッドを示します。各メソッドのシグニチャのあとに、その入力パラメータと戻り値を記載します。

SCE との接続が確立される前に呼び出されたすべてのメソッドは、**java.lang.IllegalStateException** を戻します。

ログオン操作

構文

```
void login(String subscriberID,
NetworkID networkID,
boolean networkIdAdditive,
PolicyProfile policy,
QuotaOperation quotaOperation,
OperationResultHandler handler) throws Exception
```

説明

SCE にサブスクライバを追加したり、サブスクライバを更新したりします。この操作は次のアルゴリズムに従って実行されます。

- サブスクライバ ID が SCE 内に存在しない場合は、データがすべて指定された新しいサブスクライバが追加されます。

- サブスクライバ ID が存在する場合は、次のようになります。
  - **networkIdAdditive** フラグが TRUE に設定されている場合は、サブスクライバの既存の networkID に、指定した networkID が追加されます。それ以外の場合は、指定した networkID で既存の networkID が置き換えられます。
  - **policy** — ポリシーは新しいポリシー値で*アップデートされます*。PolicyProfile 内で指定されていないサブスクライバ ポリシー エントリは、変更されないか、またはデフォルト値を使用して作成されたままになります。
  - **quota** — クォータはバケット値および与えられた操作に基づいて*アップデートされます*。「サブスクライバ クォータ」（p.4-5）を参照してください。
- 別のサブスクライバと輻輳している networkID がある場合は、別のサブスクライバの networkID が暗黙的にログアウトして、新しいサブスクライバがログインします。

関連イベントについては、「プッシュ モデル」（p.3-2）を参照してください。

パラメータ

subscriberID — サブスクライバの一意の ID。サブスクライバ ID のフォーマットについては、「サブスクライバ ID」（p.2-2）を参照してください。

networkID — サブスクライバのネットワーク ID。詳細は「ネットワーク ID のマッピング」（p.4-2）を参照してください。

networkIDAdditive — このフラグが TRUE に設定されている場合は、サブスクライバの既存の networkID に、指定した networkID が追加されます。それ以外の場合は、指定した networkID で既存の networkID が置き換えられます。

policy — サブスクライバのポリシー プロファイル。詳細は「SCA BB サブスクライバのポリシー プロファイル」（p.4-4）を参照してください。

quota — サブスクライバのクォータ。詳細は「サブスクライバ クォータ」（p.4-5）を参照してください。

handler — この操作の結果ハンドラ。*OperationResultHandler* インターフェイスについては、「結果処理」（p.5-17）を参照してください。

エラー コード

次に、このメソッドで返されることがあるエラー コードのリストを示します。

- ERROR_CODE_FATAL_EXCEPTION
- ERROR_CODE_RESOURCE_SHORTAGE
- ERROR_CODE_OPERATION_ABORTED
- ERROR_CODE_INVALID_PARAMETER
- ERROR_CODE_NO_APPLICATION_INSTALLED

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

例

次の例では、既存のマッピングは変更せずに、*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 を追加します。

```
login(
"john",                              // サブスクライバ名
new NetworkID(new String[]{"192.168.12.5"},
SCESubscriberApi.ALL_IP_MAPPINGS),
true,                                // isMappingAdditive が true
null,                                // ポリシーなし
null);                               // クォータなし
```

次の例では、IP アドレス 192.168.12.5 を追加して、直前のマッピングを上書きします。

```
login(
"john",                              // サブスクライバ名
new NetworkID(new String[]{"192.168.12.5"},
SCESubscriberApi.ALL_IP_MAPPINGS),
false,                               // isMappingAdditive が false
null,                                // ポリシーなし
null);                               // クォータなし
```

その他の例については、「ログインおよびログアウト」（p.5-40）を参照してください。

loginBulk 操作

構文

```
void loginBulk(Login_BULK subsBulk,
OperationResultHandler handler) throws Exception
```

説明

バルク内のサブスクライバごとに、ログイン操作で指定されたロジックを適用します。

パラメータ

subsBulk ― 「Login_BULK クラス」（p.4-9）を参照してください。

handler ― この操作の結果ハンドラ。*OperationResultHandler* インターフェイスについては、「結果処理」（p.5-17）を参照してください。

エラー コード

次に、このメソッドで返されることがあるエラー コードのリストを示します。

- ERROR_CODE_FATAL_EXCEPTION
- ERROR_CODE_RESOURCE_SHORTAGE
- ERROR_CODE_OPERATION_ABORTED

- ERROR_CODE_INVALID_PARAMETER
- ERROR_CODE_NO_APPLICATION_INSTALLED

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

loginPullResponse 操作

構文

```
void loginPullResponse(String subscriberID,
String anonymousSubscriberID,
NetworkID networkID,
PolicyProfile policy,
QuotaOperation quota,
OperationResultHandler handler) throws Exception
```

説明

SCE からの **loginPullRequest** 呼び出しへの応答、または **loginPullBulkRequest** への応答として、SCE にサブスクライバ ログイン情報を送信します。

関連イベントについては、「プル モデル」（p.3-3）を参照してください。

パラメータ

subscriberID ― サブスクライバの一意の ID。サブスクライバ ID のフォーマットについては、「サブスクライバ ID」（p.2-2）を参照してください。

anonymousSubscriberID ― アノニマス サブスクライバの ID。これは、loginPullRequest/loginPullBulkRequest 通知内で SCE により送信されます（「LoginPullListener インターフェイス クラス」[p.5-8] を参照）。詳細は「アノニマス サブスクライバ ID」（p.2-2）を参照してください。

networkID ― サブスクライバのネットワーク ID。詳細は「ネットワーク ID のマッピング」（p.4-2）を参照してください。この ID には、loginPullRequest から受信したネットワーク ID が含まれている必要があります。SCE のこのサブスクライバに別のネットワーク ID が設定されている場合は、既存のネットワーク ID にこのネットワーク ID が追加されます。

policy ― サブスクライバのポリシー プロファイル。詳細は「SCA BB サブスクライバのポリシー プロファイル」（p.4-4）を参照してください。

quota ― サブスクライバのクォータ。詳細は「サブスクライバ クォータ」（p.4-5）を参照してください。

handler ― この操作の結果ハンドラ。*OperationResultHandler* インターフェイスについては、「結果処理」（p.5-17）を参照してください。

エラー コード

次に、このメソッドで返されることがあるエラー コードのリストを示します。

- ERROR_CODE_FATAL_EXCEPTION
- ERROR_CODE_RESOURCE_SHORTAGE
- ERROR_CODE_OPERATION_ABORTED
- ERROR_CODE_INVALID_PARAMETER
- ERROR_CODE_NO_APPLICATION_INSTALLED

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

loginPullResponseBulk 操作

構文

```
void loginPullResponseBulk(LoginPullResponse_BULK subsBulk,
OperationResultHandler handler) throws Exception
```

説明

バルク内のサブスクライバごとに、loginPullResponse 操作で指定されたロジックを適用します。

関連イベントについては、「プル モデル」（p.3-3）を参照してください。

パラメータ

subsBulk ― 「LoginPullResponse_BULK クラス」（p.4-12）を参照してください。

handler ― この操作の結果ハンドラ。**OperationResultHandler** インターフェイスについては、「結果処理」（p.5-17）を参照してください。

エラー コード

次に、このメソッドで返されることがあるエラー コードのリストを示します。

- ERROR_CODE_FATAL_EXCEPTION
- ERROR_CODE_RESOURCE_SHORTAGE
- ERROR_CODE_OPERATION_ABORTED
- ERROR_CODE_INVALID_PARAMETER
- ERROR_CODE_NO_APPLICATION_INSTALLED

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

ログアウト操作

構文

```
void logout(String subscriberID,
NetworkID networkID,
OperationResultHandler handler) throws Exception
```

説明

SCE からサブスクライバの特定の networkID を削除します。この ID が指定されたサブスクライバの最終 networkID の場合は、SCE からサブスクライバが削除されます。サブスクライバ ID を指定しない場合は、このネットワーク ID が属するサブスクライバに関係なく、指定されたネットワーク ID が SCE から削除されます。ネットワーク ID を指定しない場合は、このサブスクライバのすべてのネットワーク ID が削除されます。

サブスクライバ レコードが SCE 内にない場合は、ログアウト操作に成功します。

関連イベントについては、「ログアウト イベント」（p.3-3）を参照してください。

パラメータ

subscriberID ― サブスクライバの一意の ID。サブスクライバ ID のフォーマットについては、「サブスクライバ ID」（p.2-2）を参照してください。

networkID ― サブスクライバのネットワーク ID。詳細は「ネットワーク ID のマッピング」（p.4-2）を参照してください。

handler ― この操作の結果ハンドラ。**OperationResultHandler** インターフェイスについては、「結果処理」（p.5-17）を参照してください。

エラー コード

次に、このメソッドで返されることがあるエラー コードのリストを示します。

- ERROR_CODE_FATAL_EXCEPTION
- ERROR_CODE_OPERATION_ABORTED

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

logoutBulk 操作

構文

```
void logoutBulk(NetworkAndSubscriberID_BULK  subsBulk,
OperationResultHandler handler) throws Exception
```

説明

バルク内のサブスクライバごとに、ログアウト操作で指定されたロジックを適用します。

関連イベントについては、「ログアウト イベント」（p.3-3）を参照してください。

パラメータ

subsBulk ― 「NetworkAndSubscriberID_BULK クラス」（p.4-11）を参照してください。

handler ― この操作の結果ハンドラ。**OperationResultHandler** インターフェイスについては、「結果処理」（p.5-17）を参照してください。

エラー コード

次に、このメソッドで返されることがあるエラー コードのリストを示します。

- ERROR_CODE_FATAL_EXCEPTION
- ERROR_CODE_OPERATION_ABORTED

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

networkIdUpdate 操作

構文

```
void networkIDUpdate(String subscriberID,
NetworkID networkID,
boolean networkIdAdditive,
OperationResultHandler handler) throws Exception
```

説明

既存のサブスクライバのネットワーク ID を追加または置換します。

(注) この操作が有効なのは、SCE にサブスクライバ レコードがある場合のみです。それ以外の場合、操作は失敗します。

関連イベントについては、「ネットワーク ID 更新イベント」（p.3-4）を参照してください。

パラメータ

subscriberID ― サブスクライバの一意の ID。サブスクライバ ID のフォーマットについては、「サブスクライバ ID」（p.2-2）を参照してください。

networkID ― サブスクライバのネットワーク ID。詳細は「ネットワーク ID のマッピング」（p.4-2）を参照してください。

networkIDAdditive ― このフラグが TRUE に設定されている場合は、サブスクライバの既存の networkID に、指定した networkID が追加されます。それ以外の場合は、指定した networkID で既存の networkID が置き換えられます。

エラー コード

次に、このメソッドで返されることがあるエラー コードのリストを示します。

- ERROR_CODE_SUBSCRIBER_NOT_EXIST
- ERROR_CODE_FATAL_EXCEPTION
- ERROR_CODE_RESOURCE_SHORTAGE
- ERROR_CODE_OPERATION_ABORTED
- ERROR_CODE_INVALID_PARAMETER
- ERROR_CODE_NO_APPLICATION_INSTALLED

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

networkIdUpdateBulk 操作

構文

```
void networkIDUpdateBulk(NetworkAndSubscriberID_BULK subsBulk,
OperationResultHandler handler) throws Exception
```

説明

バルク内のサブスクライバごとに、networkIDUpdate 操作で指定されたロジックを適用します。

関連イベントについては、「ネットワーク ID 更新イベント」（p.3-4）を参照してください。

パラメータ

subsBulk ― 「NetworkAndSubscriberID_BULK クラス」（p.4-11）を参照してください。

handler ― この操作の結果ハンドラ。**OperationResultHandler** インターフェイスについては、「結果処理」（p.5-17）を参照してください。

エラー コード

次に、このメソッドで返されることがあるエラー コードのリストを示します。

- ERROR_CODE_SUBSCRIBER_NOT_EXIST
- ERROR_CODE_FATAL_EXCEPTION
- ERROR_CODE_RESOURCE_SHORTAGE
- ERROR_CODE_OPERATION_ABORTED
- ERROR_CODE_INVALID_PARAMETER
- ERROR_CODE_NO_APPLICATION_INSTALLED

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

profileUpdate 操作

構文

```
void profileUpdate(String subscriberID,
PolicyProfile policy,
OperationResultHandler handler) throws Exception
```

説明

既存のサブスクライバのポリシー プロファイルを変更します。サブスクライバ レコードが SCE 内にない場合は、この操作に失敗します。

関連イベントについては、「プロファイル更新イベント」（p.3-5）を参照してください。

パラメータ

subscriberID ― サブスクライバの一意の ID。サブスクライバ ID のフォーマットについては、「サブスクライバ ID」（p.2-2）を参照してください。

policy ― サブスクライバのポリシー プロファイル。詳細は「SCA BB サブスクライバのポリシー プロファイル」（p.4-4）を参照してください。

handler ― この操作の結果ハンドラ。**OperationResultHandler** インターフェイスについては、「結果処理」（p.5-17）を参照してください。

エラー コード

次に、このメソッドで返されることがあるエラー コードのリストを示します。

- ERROR_CODE_SUBSCRIBER_NOT_EXIST
- ERROR_CODE_FATAL_EXCEPTION
- ERROR_CODE_OPERATION_ABORTED
- ERROR_CODE_INVALID_PARAMETER

- ERROR_CODE_NO_APPLICATION_INSTALLED

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

profileUpdateBulk 操作

構文

```
void profileUpdateBulk(PolicyProfile_BULK subsBulk,
OperationResultHandler handler) throws Exception
```

説明

バルク内のサブスクライバごとに、profileUpdate 操作で指定されたロジックを適用します。

関連イベントについては、「プロファイル更新イベント」（p.3-5）を参照してください。

パラメータ

subsBulk ― 「PolicyProfile_BULK クラス」（p.4-14）を参照してください。

handler ― この操作の結果ハンドラ。**OperationResultHandler** インターフェイスについては、「結果処理」（p.5-17）を参照してください。

エラー コード

次に、このメソッドで返されることがあるエラー コードのリストを示します。

- ERROR_CODE_SUBSCRIBER_NOT_EXIST
- ERROR_CODE_FATAL_EXCEPTION
- ERROR_CODE_OPERATION_ABORTED
- ERROR_CODE_INVALID_PARAMETER
- ERROR_CODE_NO_APPLICATION_INSTALLED

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

quotaUpdate 操作

構文

```
void quotaUpdate(String subscriberID,
QuotaOperation quotaOperation,
OperationResultHandler handler) throws Exception
```

説明

サブスクライバのクォータの更新処理を実行します。

関連イベントについては、「クォータ更新イベント」（p.3-5）を参照してください。

パラメータ

subscriberID ― サブスクライバの一意の ID。サブスクライバ ID のフォーマットについては、「サブスクライバ ID」（p.2-2）を参照してください。

quotaOperations ― サブスクライバのクォータに対して実行するクォータ処理。詳細は「サブスクライバ クォータ」（p.4-5）を参照してください。

handler ― この操作の結果ハンドラ。**OperationResultHandler** インターフェイスについては、「結果処理」（p.5-17）を参照してください。

エラー コード

次に、このメソッドで返されることがあるエラー コードのリストを示します。

- ERROR_CODE_SUBSCRIBER_NOT_EXIST
- ERROR_CODE_FATAL_EXCEPTION
- ERROR_CODE_OPERATION_ABORTED
- ERROR_CODE_INVALID_PARAMETER
- ERROR_CODE_NO_APPLICATION_INSTALLED

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

quotaUpdateBulk 操作

構文

```
void quotaUpdateBulk(QuotaOperation_BULK subsBulk,
OperationResultHandler handler) throws Exception
```

説明

バルク内のサブスクライバごとに、quotaUpdate 操作のロジックを適用します。

関連イベントについては、「クォータ更新イベント」（p.3-5）を参照してください。

パラメータ

subsBulk ― 「QuotaOperation_BULK クラス」（p.4-15）を参照してください。

handler ― この操作の結果ハンドラ。**OperationResultHandler** インターフェイスについては、「結果処理」（p.5-17）を参照してください。

エラー コード

次に、このメソッドで返されることがあるエラー コードのリストを示します。

- ERROR_CODE_SUBSCRIBER_NOT_EXIST
- ERROR_CODE_FATAL_EXCEPTION
- ERROR_CODE_OPERATION_ABORTED
- ERROR_CODE_INVALID_PARAMETER
- ERROR_CODE_NO_APPLICATION_INSTALLED

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

getQuotaStatus 操作

構文

```
void getQuotaStatus(String subscriberID,
Quota quota,
OperationResultHandler handler) throws Exception
```

説明

指定されたクォータ バケット セットの現在のクォータ残量を問い合わせる要求を送信します。この要求のあとに、問い合わせ対象データを含む getQuotaStatusIndication が続きます（**非同期**）。「quotaStatusIndication コールバック メソッド」（p.5-11）を参照してください。

関連イベントについては、「クォータ ステータス取得イベント」（p.3-6）を参照してください。

パラメータ

subscriberID ― サブスクライバの一意の ID。サブスクライバ ID のフォーマットについては、「サブスクライバ ID」（p.2-2）を参照してください。

quota ― 取得するクォータ バケットの名前リスト（値は除外）を含みます。バケット名のみでの構築方法についての詳細は、「サブスクライバ クォータ」（p.4-5）を参照してください。

handler ― この操作の結果ハンドラ。**OperationResultHandler** インターフェイスについては、「結果処理」（p.5-17）を参照してください。

エラー コード

次に、このメソッドで返されることがあるエラー コードのリストを示します。

- ERROR_CODE_SUBSCRIBER_NOT_EXIST
- ERROR_CODE_FATAL_EXCEPTION
- ERROR_CODE_OPERATION_ABORTED
- ERROR_CODE_INVALID_PARAMETER
- ERROR_CODE_NO_APPLICATION_INSTALLED

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

getQuotaStatusBulk 操作

構文

```
void getQuotaStatuBulk(Quota_BULK subsBulk,
OperationResultHandler handler) throws Exception
```

説明

このメソッドは、上記の getQuotaStatus メソッドのバルク バージョンです。

関連イベントについては、「クォータ ステータス取得イベント」（p.3-6）を参照してください。

パラメータ

subsBulk — 「Quota_BULK クラス」（p.4-14）を参照してください。

handler — この操作の結果ハンドラ。**OperationResultHandler** インターフェイスについては、「結果処理」（p.5-17）を参照してください。

エラー コード

次に、このメソッドで返されることがあるエラー コードのリストを示します。

- ERROR_CODE_SUBSCRIBER_NOT_EXIST
- ERROR_CODE_FATAL_EXCEPTION
- ERROR_CODE_OPERATION_ABORTED
- ERROR_CODE_INVALID_PARAMETER
- ERROR_CODE_NO_APPLICATION_INSTALLED

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

SCE-API の同期

SCE およびポリシー サーバで切断、ログオン メッセージの消失、リブートなどが発生して、サブスクライバのデータに矛盾が発生した場合は、問題が発生することがあります。これらの問題が発生すると、サブスクライバのトラフィックが別のサブスクライバであるかのように分類されたり、サブスクライバのトラフィックに不正なサービスが実行されたり、リソースが失われたりすることがあります。

このような矛盾を防止するには、API を使用して SCE とポリシー サーバ間のサブスクライバ データを同期させて、通信チャネルの信頼性をできるかぎり高い状態に維持します。同期を開始するのは、**常に**ポリシー サーバです（API を使用）。

（注） 同時に複数のポリシー サーバで同期プロセスを実行すると、SCE 内のサブスクライバ情報がすべてのサーバと整合性がとれなくなります。

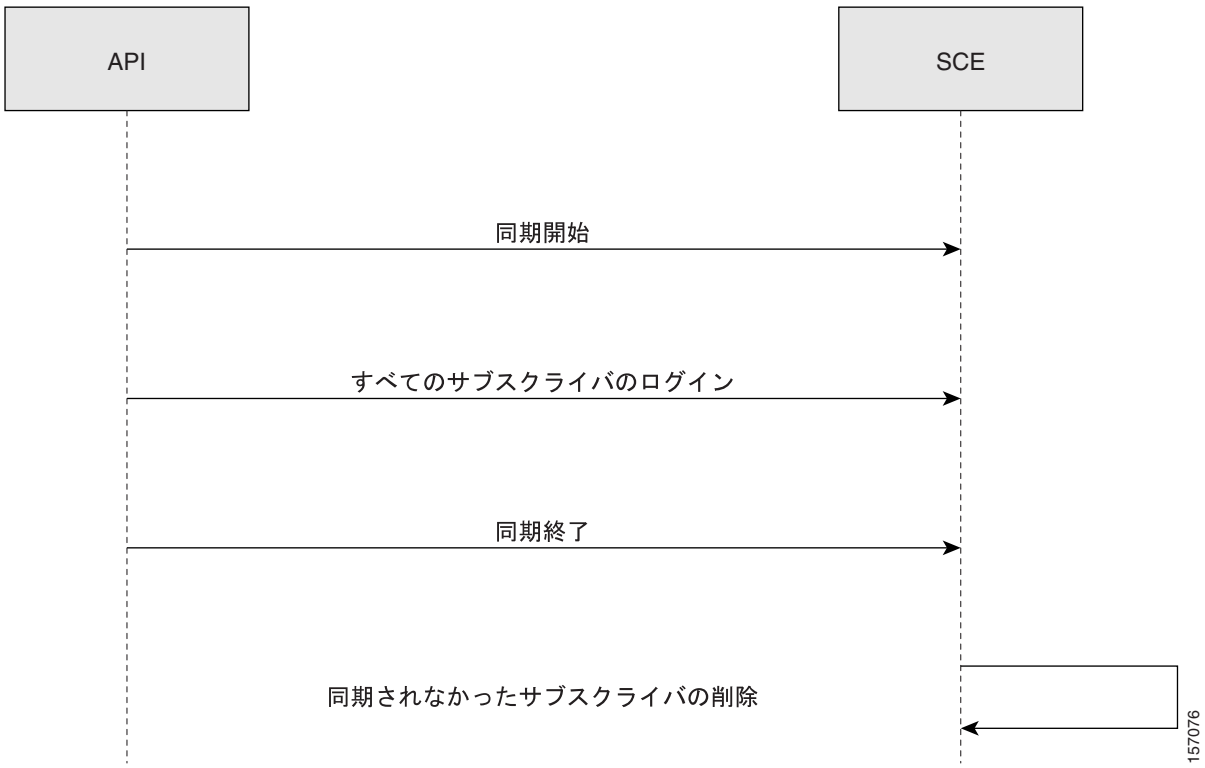
次に、同期実行中にポリシー サーバが従う必要がある、同期に関する注意事項を示します。

- プッシュ モデルの同期手順（p.5-33）
- プル モデルの同期手順（p.5-35）

プッシュ モデルの同期手順

1. ポリシー サーバから SCE に SCE の同期開始を通知します。
2. SCE が処理する必要があるすべてのサブスクライバに、ポリシー サーバがログインします。可能であれば、ログイン操作をバルクで実行します。
3. ポリシー サーバから SCE に同期終了を通知します。
4. SCE が同期プロセスに含まれないサブスクライバ データをすべて削除します。

図 5-2　プッシュ モデルの同期手順



(注)　同期プロセス中に、正規ログイン操作を実行できます。

次に、*プッシュ* モデルでの同期手順に使用するメソッドについて説明します。

- synchronizePushStart（p.5-34）
- synchronizePushEnd（p.5-35）

synchronizePushStart

- 構文（p.5-34）
- 説明（p.5-34）
- パラメータ（p.5-35）

構文

```
void synchronizePushStart(OperationResultHandler handler)
```

説明

この操作は、サーバとの同期を開始することを SCE に通知する場合に限って、*プッシュ* モデルで使用します。SCE はすべてのサブスクライバ データに「ダーティ ビット」をマークします。このデータが同期プロセス中に再適用された場合、ダーティ ビットはリセットされます。このメソッドを呼び出すたびに、同期プロセスは再起動されます。

パラメータ

handler ― この操作の結果ハンドラ。**OperationResultHandler** インターフェイスについては、「結果処理」（p.5-17）を参照してください。

synchronizePushEnd

構文

```
void synchronizePushEnd(boolean success, OperationResultHandler handler)
```

説明

この操作は、サーバとの同期が終了したことを SCE に通知する場合に限って、*プッシュ* モデルで使用します。SCE はサブスクライバ データベース全体をスキャンして、**synchronizePushStart** で「ダーティ ビット」が割り当てられたデータを検索し、削除します。

パラメータ

success ― 同期に成功したことを SCE に通知するフラグ

handler ― この操作の結果ハンドラ。*OperationResultHandler* インターフェイスについては、「結果処理」（p.5-17）を参照してください。

プル モデルの同期手順

1. ポリシー サーバから SCE に SCE の同期開始を通知します。
2. ポリシー サーバが、現在処理中のサブスクライバ ID およびネットワーク ID をすべて SCE から取得します。
3. ポリシー サーバが同期外れをすべて修正します。

アルゴリズム

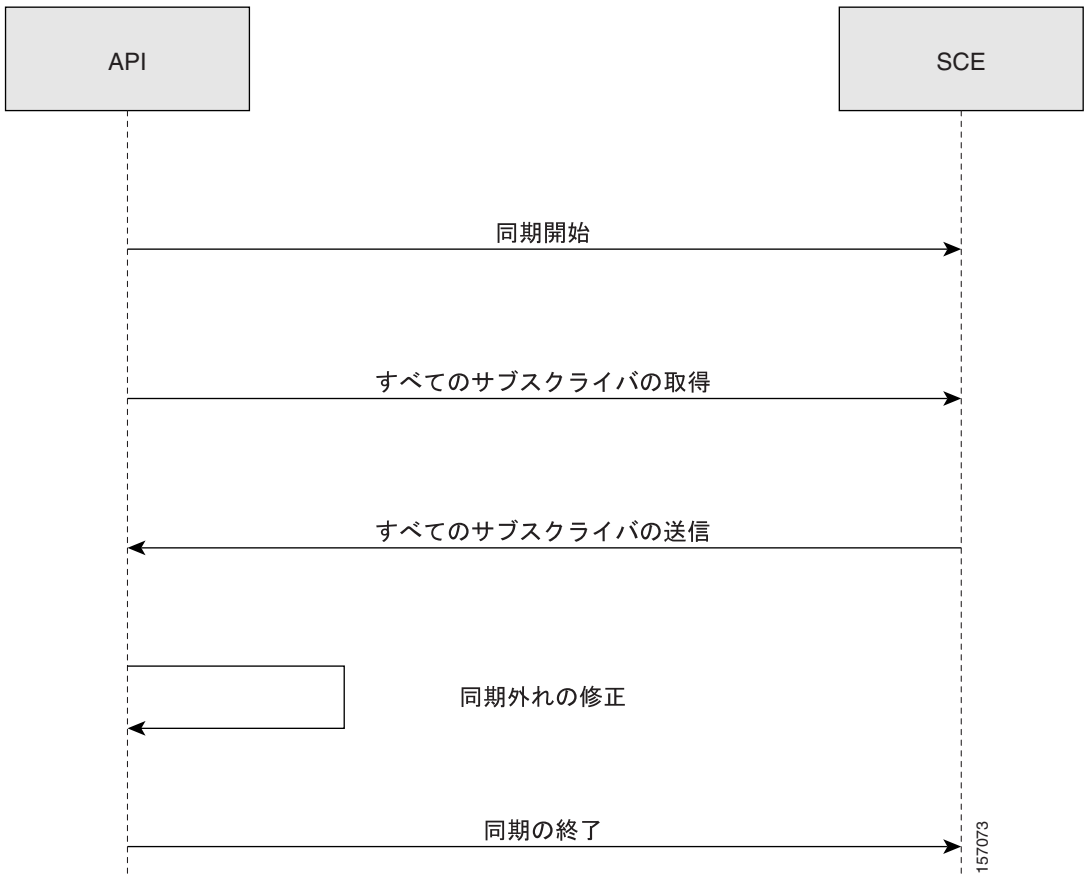
同期手順を計画する場合は、次のアルゴリズム テンプレートを使用します。

取得したサブスクライバ（<SubscriberID, IP address>）ごとに次の手順を実行します。

- ポリシー サーバ データベースに <SubscriberID, IP address> が存在する場合、
  SCE にポリシー プロファイルおよび networkID アップデートを送信します。
- それ以外の場合、
  SCE にログアウトおよびサブスクライバ IP を送信します。

ステップ 2 および 3 は同時に一括して実行されます。

図 5-3　プル モデルの同期手順



(注)　同期プロセス中に、正規ログイン操作を実行できます。

次に、*プル* モデルでの同期手順に使用するメソッドについて説明します。

- synchronizePullStart（p.5-36）
- synchronizePullEnd（p.5-37）
- getSubscribersBulk（p.5-37）

synchronizePullStart

- 構文（p.5-36）
- 説明（p.5-36）
- パラメータ（p.5-37）

構文

```
void synchronizePullStart(OperationResultHandler handler)
```

説明

この操作は、サーバとの同期を開始する必要があることを SCE に通知する場合に限って、*プル* モデルで使用します。

パラメータ

handler ― この操作の結果ハンドラ。**OperationResultHandler** インターフェイスについては、「結果処理」（p.5-17）を参照してください。

synchronizePullEnd

構文

```
void synchronizePullEnd(boolean success, OperationResultHandler handler)
```

説明

この操作は、サーバとの同期が終了したことを SCE に通知する場合に限って、*プル* モデルで使用します。

パラメータ

handler ― この操作の結果ハンドラ。**OperationResultHandler** インターフェイスについては、「結果処理」（p.5-17）を参照してください。

success ― 同期に成功したことを SCE に通知するフラグ

getSubscribersBulk

構文

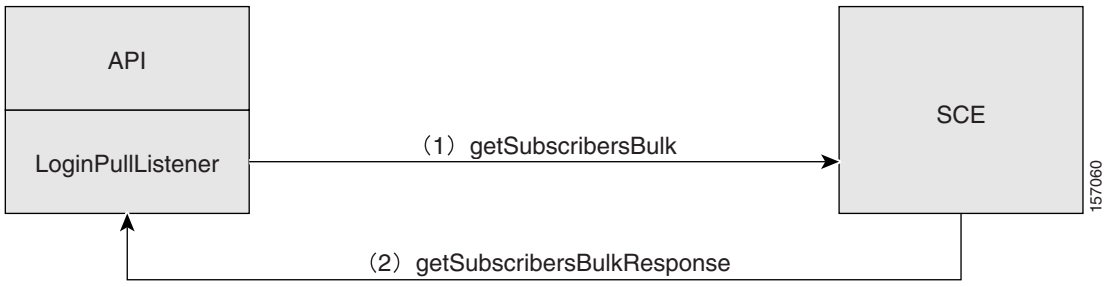
```
void getSubscribersBulk(int bulkSize,
SubscribersBulkResponseIterator iterator,
OperationResultHandler handler)
```

説明

この操作は、SCE が現在処理中のサブスクライバのバルクを取得するために、*プル* モデルの同期プロセスで使用します（「プル モデルの同期手順」[p.5-35]）。

この要求（**getSubscribersBulk**）を受信すると、SCE は subscriberID および対応する NetworkID を含む **getSubscribersBulkResponse** 通知を非同期に発行します（「LoginPullListener インターフェイス クラス」を参照）。このメソッドは、次に呼び出される **getSubscribersBulk** に渡されるイテレータを指定します。反復の終了を示すために、最後のバルクのイテレータは null になります。

図 5-4　Get Subscribers Bulk



パラメータ

bulkSize ― 取得するバルクのサイズ。最大バルク サイズは 100 エントリに制限されています。

iterator ― SCE 側のサブスクライバのイテレータ。このイテレータは getSubscribersBulkResponseIndication で受信され、次に呼び出される getSubscribersBulk メソッドに渡す必要があります。最初に getSubscribersBulk メソッドを呼び出す場合は、イテレータとして **null** を使用します（**null** を使用すると、最初から処理が開始されます）。

handler ― この操作の結果ハンドラ。**OperationResultHandler** インターフェイスについては、「結果処理」（p.5-17）を参照してください。

高度な API プログラミング

ハイ アベイラビリティの実装

API のハイ アベイラビリティ サポート機能では、ポリシー サーバのハイ アベイラビリティ方式が 2 ノード クラスタ タイプであり、同時にアクティブにできるサーバは 1 台のみであると想定しています。別のサーバ（スタンバイ）は、SCE に接続されません。

アクティブ サーバに障害が発生すると、ユーザの 2 ノード クラスタ方式によって、スタンバイ サーバにフェールオーバーします。

（注） **SCE プラットフォームをプロビジョニングするポリシー サーバごとに、ハイ アベイラビリティをそれぞれ同時に実装できます。**

SCMS SCE Subscriber API でハイ アベイラビリティを実装するには、次の作業を行う必要があります。

- 2 台のポリシー サーバに対応するように 2 ノード クラスタを設定します。
- *同じ API 名*を持つ API インスタンスをクラスタ内のサーバ（ノード）ごとに 1 つずつ、合計 2 つ構築します（コンストラクタの説明については、「API の構築」[p.5-4] を参照）。クラスタ実行中に SCE プラットフォームに接続する必要がある API インスタンスは 1 つのみです。フェールオーバーが発生すると、障害のあるサーバが SCE から切断され、スタンバイ サーバがアクティブになり、定義済みタイムアウト内に SCE に再接続します（「API 切断タイムアウトの設定」[p.1-7] を参照）。API 名が同じであるため、SCE は同じ API が再接続し、情報が失われないかのように動作します。

（注） 障害のあるポリシー サーバで使用する API 内で、**unregisterXXXListener** メソッドを暗黙的に呼び出さないでください。データが消失することがあります。**disconnect()** メソッドを呼び出しても、リスナは登録解除されません。

API コードの例

ここでは、API を使用するためのコード例を示します。

ログインおよびログアウト

次の例では、事前定義された数のサブスクライバが SCE にログインしたのち、ログアウトします。この例では自動再接続のサポートを使用するため、接続リスナは定義しません。

次のコード例には、成功した結果および失敗した結果をカウントする**結果ハンドラ**の実装例も含まれています。

```
// 操作結果を処理するクラス
import com.scms.api.sce.OperationArguments;
import com.scms.api.sce.OperationException;
import com.scms.api.sce.OperationResultHandler;
public class MyOperationResultHandler implements OperationResultHandler
{
long count = 0;

public void handleOperationResult(Object[] result,
OperationArguments handback)
{
for (int index=0; index <result.length; index++)
{
count++;
if (result[index]==null)
{
//100 回の操作ごとに success を出力します。
//if (++count%100 == 0)
{
System.out.println("\tsuccess "+count);
}
}
else // エラー - すべてのエラーを出力
{
// 失敗
count++;
// エラーの詳細を抽出
OperationException ex =
(OperationException)result[index];
// 操作名を抽出
String operationName = handback.getOperationName();
// 操作名およびエラー メッセージを出力
System.out.println("Error for operation "+
operationName+": "+
ex.getMessage());
}
}
}
public synchronized void waitForLastResult(int lastResult)
{
while (count<lastResult)
{
try
{
wait(100);
}
catch (InterruptedException ie)
{
ie.printStackTrace();
}
```

```
}
}
}
```

受信したログアウト通知数をカウントする単純な LogoutListener 実装を含むクラス：

```
import com.scms.api.sce.LogoutListener;
import com.scms.common.NetworkAndSubscriberID_BULK;
import com.scms.common.SubscriberID_BULK;
class MyLogoutListener implements LogoutListener
{
long count = 0;

public void logoutIndication(String subscriberID)
{
increaseCounter(1);
}

synchronized void increaseCounter(long value)
{
count = count + value;
}

synchronized long getCounter()
{
return count;
}
// 結果数「last result」を受け取るまで待機します。
public synchronized void waitForLastResult(int lastResult)
{
while (count<lastResult)
{
try
{
wait(100);
}
catch (InterruptedException ie)
{
ie.printStackTrace();
}
}
}

public void logoutBulkIndication(SubscriberID_BULK subs)
{
increaseCounter(subs.getSize());
}
}
```

main メソッドを含むクラス：

```
import com.scms.api.sce.prpc.PRPC_SCESubscriberApi;
import com.scms.common.*;
public class LogonPolicyServer {
public static void main (String args[]) throws Exception
{
int numSubscribersToLogin = 500;
//5 秒の再接続インターバルで API をインスタンス化します。
PRPC_SCESubscriberApi api = new PRPC_SCESubscriberApi(
"myAPI",
args[0], // SCE の IP
5000);
try {
// 操作結果ハンドラをインスタンス化します。
// すべての操作に対してハンドラを 1 つ使用します。
MyOperationResultHandler resultHandler =
new MyOperationResultHandler();
// ログアウト リスナをインスタンス化します。
MyLogoutListener listener = new MyLogoutListener();
// ログアウト通知に登録します。
api.registerLogoutListener(listener);
// SCE に接続します。
api.connect();
// ログイン
System.out.println("login of "+numSubscribersToLogin+
" subscribers");
PolicyProfile pp = new PolicyProfile(
new String[]{"packageId=1",
"monitor=1"});
for (int i=0; i<numSubscribersToLogin; i++)
{
api.login("sub"+i,
new NetworkID(getMappings(i), // ip を生成します。
NetworkID.ALL_IP_MAPPINGS),
true,                          // 追加フラグ
pp,                            // ポリシー
null,                          // クォータなし
resultHandler);
}
// サブスクライバがログインするまで待機します。
resultHandler.waitForLastResult(numSubscribersToLogin);
// すべてのサブスクライバをログアウトします。
System.out.println("logout of "+numSubscribersToLogin+
" subscribers");
for (int i=0; i<numSubscribersToLogin; i++)
{
NetworkID nid = new NetworkID(getMappings(i),
NetworkID.ALL_IP_MAPPINGS);
api.logout("sub"+i,nid,resultHandler);
}
// すべてのサブスクライバがログアウトするまで待機します。
// ただし、今回は、
// ログアウト リスナを使用して結果をカウントします。
listener.waitForLastResult(numSubscribersToLogin);
}
finally
{
api.unregisterLogoutListener
api.disconnect();
}
}
// サンプル プログラムの「自動」マッピング ジェネレータ
private static String[] getMappings(int i) {
return new String[]{ "10." +((int)i/65536)%256 + "." +
((int)(i/256))%256 + "." + (i%256)};
}
}
```

ログインプル要求およびログインプル応答

次のコード例は、ログインプル要求およびログインプル応答の操作方法を示します。

このクラスは、ログアウトおよびログイン プル通知のためのリスナの実装例です。

```
import java.util.Iterator;
// 直前の例の結果ハンドラ
import MyOperationResultHandler;
import com.scms.api.sce.*;
import com.scms.common.*;
class MyListener implements LoginPullListener, LogoutListener
{
// 通知カウンタ
long logoutCount = 0;
long pullCount=0;
// API インスタンス - SCE へのログインプル応答を送信する場合に使用します。
PRPC_SCESubscriberApi api = null;

// 直前の（ログインおよびログアウト）例
// から操作ハンドラを構築します。
MyOperationResultHandler h = new MyOperationResultHandler();
public MyListener(PRPC_SCESubscriberApi api)
{
this.api = api;
}
// ログアウト カウンタを増加させます。
public void logoutIndication(String subscriberID)
{
increaseLogoutCounter(1);
System.out.println("Got logout notification " +
getLogoutCounter());
}
// ログアウト カウンタを増加させます。
public void logoutBulkIndication(SubscriberID BULK subs)
{
System.out.println("Got logout notification");
increaseLogoutCounter(subs.getSize());
}
public void loginPullRequest (String anonymousSubscriberID,
NetworkID networkID)
{
try
{
increasePullCounter(1);
System.out.println("Got pull request" + getPullCounter());

// ポリシーを作成します。
PolicyProfile pp = new PolicyProfile(
new String[]{"packageId=1",
"monitor=1"});
// プル応答で応答します。
// サブスクライバ名を取得します。取得元は、
// ポリシー サーバ データベースなどです。
// この例では、サブスクライバ カウンタに基づく
// 固定名を使用します。
api.loginPullResponse(anonymousSubscriberID,
"sub"+getPullCounter(),
networkID,
pp,     // ポリシー
null,   // クォータなし
h); // 直前の例のハンドラ
}
catch (Exception ex)
{
System.out.println(ex.getMessage());
}
}
public void loginPullRequestBulk(NetworkAndSubscriberID BULK subs)
```

```
{
try
{
increasePullCounter(subs.getSize());
System.out.println("Got pull request" + getPullCounter());
// バルク形式のプル応答で応答します。
PolicyProfile pp = new PolicyProfile(
new String[]{"packageId=1",
"monitor=1"});
LoginPullResponse_BULK responseBulk =

new LoginPullResponse_BULK();
Iterator subsIterator = subs.getIterator();
// 受信したバルクを反復し（IP およびアノニマス ID）
// 応答バルクを作成します。
int count=0;
while(subsIterator.hasNext())
{
// サブスクライバ名を取得します。取得元は、
// ポリシー サーバ データベースなどです。
// この例では、サブスクライバ カウンタに基づく
// 固定名を使用します。
String subName = "sub_"+count;
SubscriberData sub = (SubscriberData)subsIterator.next();
// バルクからサブスクライバ マッピングを抽出し、
// これらのマッピングに基づいて新しい NetworkID を構築します。
NetworkID subNetId = new NetworkID(sub.getMappings(),
NetworkID.ALL_IP_MAPPINGS);
responseBulk.addEntry(sub.getAnonymousSubscriberID(),
subName,
subNetId,
true,
pp,
null);
count++;
}
// 上記のように構築されたバルクをバルク応答で使用します。
// 直前の例のハンドラを使用します。
api.loginPullBulkResponse(responseBulk,h);
}
catch (Exception ex)
{
System.out.println(ex.getMessage());
}
}
public void getSubscribersBulkResponse(
NetworkAndSubscriberID BULK subs,
SubscruberBulkResponseIterator iterator)
{
// この例では実装しません。
}

synchronized void increaseLogoutCounter(long value)
{
logoutCount = logoutCount + value;
}
synchronized void increasePullCounter(long value)
{
pullCount = pullCount + value;
}
synchronized long getPullCounter()
{
return pullCount;
}
synchronized long getLogoutCounter()
{
return logoutCount;
}
// 結果数「last result」を受け取るまで待機します。
```

```
public synchronized void waitForPullResult(int lastResult) {
while (pullCount<lastResult) {
try {
wait(100);
} catch (InterruptedException ie) {
ie.printStackTrace();

}
}
}
public synchronized void waitForLogoutResult(int lastResult) {
while (logoutCount<lastResult) {
try {
wait(100);
} catch (catch (InterruptedException ie) {
ie.printStackTrace();
}
}
}
}
```

main メソッドを含むクラス：

```
import java.util.Iterator;
import com.scms.api.sce.*;
import com.scms.common.*;
public class LogonPolicyServer {
static PRPC_SCESubscriberApi api = null;

// このサンプル プログラムは SCE からのプル要求を待機して、
// プル応答で応答します。
// 500 のサブスクライバがすべてログインすると、プログラムが終了します。
public static void main (String args[]) throws Exception
{
int numSubscribersToLogin = 500;
//5 秒の再接続インターバルで API をインスタンス化します。
api = new PRPC_SCESubscriberApi("myAPI","1.1.1.1",5000);
// 上記の例から、操作結果ハンドラを
// 構築します。
MyOperationResultHandler handler =
new MyOperationResultHandler();

// ログアウトおよびログインプル リスナをインスタンス化します。
MyListener listener = new MyListener(api);
try
{
// ログアウト通知に登録します。
api.registerLogoutListener(listener);
api.registerLoginPullListener(listener);
// SCE に接続します。
api.connect();

// SCE からのログインプル要求を待機します。
// これらの要求は、SCE に不明サブスクライバのトラフィックが
// ある場合に送信されます。
System.out.println("Waiting for pull requests for "+
numSubscribersToLogin+
" subscribers");
// すべてのサブスクライバがログインするまで待機します。
listener.waitForPullResult(numSubscribersToLogin);
// すべてのサブスクライバをログアウトします。
System.out.println("logout of "+numSubscribersToLogin+
" subscribers");
for (int i=0; i<numSubscribersToLogin; i++)
{
api.logout("sub"+i,null,handler);
}
// すべてのサブスクライバがログアウトするまで待機します。
```

```
listener.waitForLogoutResult(numSubscribersToLogin);
}
finally
{
api.unregisterLoginPullListener();
api.unregisterLogoutListener();
api.disconnect();
}
}
}
```

CHAPTER 6

トラブルシューティング

この章では、API ロギング機能を使用した、API との統合に関するトラブルシューティング方法について説明します。API ロギングを使用すると、API クライアントと SCE 側の両方で受信されたパラメータを含めて、呼び出し中の処理をモニタできます。

SCE ロギング

SCE プラットフォームには、ポリシー サーバで呼び出されたすべての操作を SCE ユーザログ ファイルに記録する機能があります。

デフォルト ログ メッセージ

追加設定しなくても、SCE はデフォルトで次のメッセージを発行します。

- 接続操作の場合：

```
<client-name>- connect operation was called, registered listeners: <type of the
listeners that were registered>
```

- 切断操作の場合：

```
<client-name>- disconnected
```

- registerLoginPullListener 操作の場合：

```
<client-name>- registered a Login Pull Listener
```

- unregisterPullListener 操作の場合：

```
<client-name>- unregistered a Pull Listener
```

- registerLogoutListener 操作の場合：

```
<client-name>- registered a Logout Listener
```

- unregisterLogoutListener 操作の場合：

```
<client-name>- unregistered a Logout Listener
```

- registerQuotaListener 操作の場合：

```
<client-name>- registered Quota Listener
```

- unregisterQuotaListener 操作の場合：

```
<client-name>- unregister Quota Listener
```

- synchronizePushStart 操作の場合：

```
<client-name>- synchronize Push Start
```

- synchronizePushEnd 操作の場合：

```
<client-name>- synchronize Push End
```

- synchronizePullStart 操作の場合：

```
<client-name>- synchronize Pull Start
```

- synchronizePullEnd 操作の場合：

```
<client-name>- synchronize Pull End
```

サブスクライバ操作のログ メッセージ

サブスクライバ操作のログ メッセージは、専用フラグでアクティブ化されます。これらのメッセージを受信するには、フラグをイネーブルにします。

ロギングをイネーブルにするには、SCE プラットフォームで次の CLI を使用します。

```
(config)# management-agent sce-api logging
```

USERLOG ファイルを表示するには、SCE プラットフォームで次の CLI を使用します。

```
#>logger get user-log   FILE NAME
```

(注)　ロギングをイネーブルにすると、パフォーマンスが低下します。したがって、ロギングはトラブルシューティングを行う場合のみ使用することを推奨します。

ロギングをディセーブルにするには、SCE プラットフォームで次の CLI を使用します。

```
(config)#>no management-agent sce-api logging
```

ロギングがイネーブルかどうかを表示するには、SCE プラットフォームで次の CLI を使用します。

```
#>show management-agent sce-api
```

ロギング フラグがイネーブルの場合に、次の操作を実行すると、以下のメッセージが発行されます。

- ログイン操作
- networkIDUpdate 操作
- ログアウト操作
- quotaUpdate 操作
- loginPullResponse 操作
- profileUpdate 操作
- getQuotaStatus 操作

```
<operation name>operation was called
with parameters:
subscriberID - <subscriber ID>
anonymousSubscriberID - <anonymousSubscriberID >
mappings - <mappings list>
mappings types - <mapping types list>
policy - <policy properties list>
quota - <quota operation/quota buckets list>
```

次のバルク操作の場合：

- loginBulk 操作
- networkIDUpdateBulk 操作
- logoutBulk 操作
- quotaUpdateBulk 操作
- loginPullBulkResponse 操作
- profileUpdateBulk 操作
- getQuotaStatuBulkRequest 操作
- getSubscribersBulk

次のメッセージが発行されます。

```
<operation name>operation was called with parameters:
bulk size - <bulk size>
```

LoginPullListener の場合は、次のメッセージが発行されます。

- loginPullRequest の場合：

```
loginPullRequest operation was called with parameters:
anonymousSubscriberID - <anonymous subscriber ID>
mappings - <mappings list>
mapping types - <mapping types>
```

- loginPullRequestBulk の場合：

```
loginPullRequestBulk operation was called with parameters:
bulk size - <bulk size>
```

- getSubscribersBulkResponse の場合：

```
getSubscribersBulkResponse operation was called with parameters:
bulk size - <bulk size>
```

LogoutListener の場合は、次のメッセージが発行されます。

- logoutIndication の場合

```
logoutIndication operation was called with parameters:
subscriberID - <anonymous subscriber ID>
```

- logoutBulkIndication の場合：

```
logoutBulkIndication operation was called with parameters:
bulk size - <bulk size>
```

QuotaListenerEx の場合は、次のメッセージが発行されます。

- quotaStatusIndication の場合：

```
quotaStatusIndication operation was called with parameters:
subscriberID - <Subscriber ID>
quota - <subscriber quota>
```

- quotaBelowThresholdIndication の場合：

```
quotaBelowThresholdIndication operation was called with parameters:
subscriberID - <Subscriber ID>
quota - <subscriber quota>
```

- quotaDepletedIndication の場合：

```
quotaDepletedIndication operation was called with parameters:
subscriberID - <Subscriber ID>
quota - <subscriber quota>
```

- quotaStateRestore の場合：

```
quotaStateRestore operation was called with parameters:
subscriberID - <Subscriber ID>
quota - <subscriber quota>
```

- quotaStatusBulkIndication の場合：

```
quotaStatusBulkIndication operation was called with parameters:
subs - <bulk size>
```

- quotaBelowThresholdBulkIndication の場合：

```
quotaBelowThresholdBulkIndication operation was called with parameters:
subs - <bulk size>
```

- quotaDepletedBulkIndication の場合：

```
quotaDepletedBulkIndication operation was called with parameters:
subs - <bulk size>
```

- quotaStateBulkRestore の場合：

```
quotaStateBulkRestore operation was called with parameters:
subs - <bulk size>
```

API クライアント ロギング

API には、アクティブ化されたすべての処理を **${user.home}** ディレクトリ内の apilog ファイルに記録する機能があります。ロギング パラメータは Log4J プロパティ ファイルを使用して設定します。ロギングをイネーブルにするには、アプリケーションの CLASSPATH にこのファイルが格納されていることを確認します。このファイルはアプリケーションを起動するときに読み込まれるため、ファイルを変更した場合は、アプリケーションを再起動する必要があります。

次に、**log4.properies** ファイルの内容を示します。

```
# default Log4j configuration for SCE Subscriber API
log4j.rootCategory=INFO, apiStdout
# In order to enable the logging to the file Replace the above
# line with the following:
# log4j.rootCategory=INFO, files
# stdout is set to be a ConsoleAppender.
log4j.appender.apiStdout=org.apache.log4j.ConsoleAppender
log4j.appender.apiStdout.layout=org.apache.log4j.PatternLayout
log4j.appender.apiStdout.layout.ConversionPattern=+ %d{dd-MMM HH:mm:ss.SSS} [%t] %-5p
%c%n%m%n
# files is set to be a RollingFileAppender.
#log4j.appender.files=org.apache.log4j.RollingFileAppender
#log4j.appender.files.layout=org.apache.log4j.PatternLayout
#log4j.appender.files.layout.ConversionPattern=+ %d{dd-MMM yyyy HH:mm:ss.SSS} [%t]
%-5p %c %x\n%m\n
#log4j.appender.files.File=${user.home}/apilog
#log4j.appender.files.Threshold=INFO
#log4j.appender.files.ImmediateFlush=true
#log4j.appender.files.MaxFileSize=1MB
#log4j.appender.files.MaxBackupIndex=4
# In order to enable debug logging uncomment the following line
#log4j.category.com.scms.api.sce.prpc=DEBUG
```

デバッグ ロギングをイネーブルにするには、ファイルの最終行をコメント解除します。デフォルトでは、ロギングは標準出力に対して実行されます。ロギングをファイルに転送するには、ファイルの説明に従って、# log4j.rootCategory=INFO, files 行をコメント解除します。

API クライアント ログ メッセージ

log4j.properties ファイルを適切に設定すると、API クライアントは次のメッセージを発行します。

- API コンストラクタの場合：
- `PRPC_SCESubscriberApi` コンストラクタは、次のパラメータと一緒に呼び出されます。

  ```
  apiName - <apiName>
  host - <sceHost>
  port - <scePort>
  auto-reconnect - <autoReconnectInterval>
  ```

- init 操作の場合：

  ```
  init operation was called with parameters <properties>
  ```

- setConnectionListener の場合：

  ```
  setConnectionListener operation was called
  ```

- setRedundancyStateListener の場合：

  ```
  setRedundancyStateListener operation was called
  ```

- isConnected の場合：

```
isConnected operation was called
```

- getAPIVersion の場合：

```
getAPIVersion operation was called
```

次の操作の場合：

- ログイン操作
- networkIDUpdate 操作
- ログアウト操作
- quotaUpdate 操作
- loginPullResponse 操作
- profileUpdate 操作
- getQuotaStatus 操作

次のメッセージが発行されます。

```
<operation name>operation was called with parameters:
subscriberID - <subscriber ID>
anonymousSubscriberID - <anonymousSubscriberID>
mappings - <mappings list>
mappings types - <mapping types list>
policy - <policy properties list>
quota - <quota operation/quota buckets list>
```

次のバルク操作の場合：

- loginBulk 操作
- networkIDUpdateBulk 操作
- logoutBulk 操作
- quotaUpdateBulk 操作
- loginPullBulkResponse 操作
- profileUpdateBulk 操作
- getQuotaStatuBulkRequest 操作
- getSubscribersBulk 操作

次のメッセージが発行されます。

```
operation name>operation was called with parameters:
bulk size - <bulk size>
```

- 接続操作の場合：

```
connect operation was called, registered listeners:
<type of the listeners that were registered>
```

- 切断操作の場合：

```
disconnect operation was called
```

- registerLoginPullListener 操作の場合：

```
registerLoginPullListener operation was called
```

- unregisterPullListener 操作の場合：

```
unregisterPullListener operation was called
```

- registerLogoutListener 操作の場合：

```
registerLogoutListener operation was called
```

- unregisterLogoutListener 操作の場合：

```
unregisterLogoutListener operation was called
```

- registerQuotaListener 操作の場合：

```
registerQuotaListener operation was called
```

- unregisterQuotaListener 操作の場合：

```
unregisterQuotaListener operation was called
```

- synchronizePushStart 操作の場合：

```
synchronizePushStart operation was called
```

- synchronizePushEnd 操作の場合：

```
synchronizePushEnd operation was called
```

- synchronizePullStart 操作の場合：

```
synchronizePullStart operation was called
```

- synchronizePullEnd 操作の場合：

```
synchronizePullEnd operation was called
```

LoginPullListener リスナ コールバック メソッドの場合は、次のメッセージが発行されます。

- loginPullRequest の場合：

```
loginPullRequest operation was called with parameters:
anonymousSubscriberID - <anonymous subscriber ID>
mappings - <mappings list>
mapping types - <mapping types>
```

- loginPullRequestBulk の場合：

```
loginPullRequestBulk operation was called with parameters:
bulk size - <bulk size>
```

- getSubscribersBulkResponse の場合：

```
getSubscribersBulkResponse operation was called with parameters:
bulk size - <bulk size>
```

LogoutListener リスナ コールバック メソッドの場合は、次のメッセージが発行されます。

- logoutIndication の場合

```
logoutIndication operation was called with parameters:
subscriberID - <anonymous subscriber ID>
```

- logoutBulkIndication の場合：

```
logoutBulkIndication operation was called with parameters:
bulk size - <bulk size>
```

QuotaListenerEx リスナ コールバック メソッドの場合は、次のメッセージが発行されます。

- quotaStatusIndication の場合：

```
quotaStatusIndication operation was called with parameters:
subscriberID - <Subscriber ID>
quota - <subscriber quota>
```

- quotaBelowThresholdIndication の場合：

```
quotaBelowThresholdIndication operation was called with parameters:
subscriberID - <Subscriber ID>
quota - <subscriber quota>
```

- quotaDepletedIndication の場合：

```
quotaDepletedIndication operation was called with parameters:
subscriberID - <Subscriber ID>
quota - <subscriber quota>
```

- quotaStateRestore の場合：

```
quotaStateRestore operation was called with parameters:
subscriberID - <Subscriber ID>
quota - <subscriber quota>
```

- quotaStatusBulkIndication の場合：

```
quotaStatusBulkIndication operation was called with parameters:
subs - <bulk size>
```

- quotaBelowThresholdBulkIndication の場合：

```
quotaBelowThresholdBulkIndication operation was called with parameters:
subs - <bulk size>
```

- quotaDepletedBulkIndication の場合：

```
quotaDepletedBulkIndication operation was called with parameters:
subs - <bulk size>
```

- quotaStateBulkRestore の場合：

```
quotaStateBulkRestore operation was called with parameters:
subs - <bulk size>
```

APPENDIX **A**

エラー コードのリスト

この章では、API から戻されるエラー コードを示します。

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 **OperationException** が戻されたのかを知るために役立ちます。エラー コードを抽出するには、**getErrorCode** メソッドを使用します。

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_NO_APPLICATION_INSTALLED	操作を実行するために必要なアプリケーションがインストールされていません。
ERROR_CODE_INVALID_PARAMETER	メソッドに指定された引数の 1 つが無効です。
ERROR_CODE_SUBSCRIBER_ALREADY_EXISTS	操作の実行対象となったサブスクライバは、すでに SCE に存在しています。
ERROR_CODE_SUBSCRIBER_DOES_NOT_EXIST	操作の実行対象であるサブスクライバは、SCE に存在していません。
ERROR_CODE_FATAL_EXCEPTION	操作を実行するときに、SCE に発生したエラーが多すぎます。
ERROR_CODE_RESOURCE_SHORTAGE	内部エラー
ERROR_CODE_OPERATION_ABORTED	内部エラー
ERROR_CODE_ARRAY_ACCESS	内部エラー
ERROR_CODE_ATTRIBUTE_NOT_FOUND	内部エラー
ERROR_CODE_CLASS_CAST	内部エラー
ERROR_CODE_CLASS_NOT_FOUND	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	内部エラー
ERROR_CODE_OBJECT_NOT_FOUND	内部エラー
ERROR_CODE_OPERATION_NOT_FOUND	内部エラー
ERROR_CODE_OUT_OF_MEMORY	内部エラー
ERROR_CODE_RUNTIME	内部エラー

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_NULL_POINTER	内部エラー
ERROR_CODE_UNKNOWN	内部エラー