CHAPTER 4

ノンブロッキング API

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

信頼性のサポート

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

- 信頼モード（p.4-2）
- 非信頼モード（p.4-2）

信頼モード

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

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

非信頼モード

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

自動再接続のサポート

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

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

マルチスレッドのサポート

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

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

ResultHandler インターフェイス

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

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

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

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

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

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

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

ResultHandler インターフェイス例

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

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

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

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

ノンブロッキング API の構築

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

ノンブロッキング API の構文

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

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

ノンブロッキング API の引数

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

- **autoReconnectInterval**

  次のように、再接続タスクの再接続試行間隔（ミリ秒単位）を決めます。

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

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

- **reliable**

  API が信頼モードで動作するかどうかを決定するフラグです。

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

  LEG の名前。「API の構築」（p.2-3）を参照してください。

ノンブロッキング API 例

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

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

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

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

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

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

ノンブロッキング API の初期化

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

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

次のプロパティを設定できます。

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

ノンブロッキング API の初期化構文

ノンブロッキング API の init メソッドの構文は、次のとおりです。

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

ノンブロッキング API の初期化パラメータ

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

- **properties (java.util.Properties)**

  前述のプロパティを設定できます。

  - 出力キュー サイズを設定するには、**prpc.client.output.machinemode.recordnum** を使用します。
  - 操作タイムアウトを設定するには、**prpc.client.operation.timeout** を使用します。

ノンブロッキング API の初期化例

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

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

ノンブロッキング API メソッド

ここでは、ノンブロッキング API のメソッドについて説明します。

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

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

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

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

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

ここでは、次のメソッドについて説明します。

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

login

構文

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

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

logoutByName

構文

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

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

logoutByNameFromDomain

構文

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

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

logoutByMapping

構文

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

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

loginCable

構文

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

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

logoutCable

構文

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

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

ノンブロッキング API のコード例

ここでは、サブスクライバのログインおよびログアウトのコード例を紹介します。

ログインおよびログアウト

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

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

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

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

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