

# Caché Java バインディングガイド

(Caché Version 2011.1 ベース)

V1.0

2011年8月

インターシステムズジャパン株式会社

目次

1.	はじめに .....	3
2.	Java Binding .....	4
3.	Java アプリケーションを作成する .....	19
3.1.	Caché クラスのインポート / Java プロキシクラスを出力しよう .....	20
3.2.	Java クライアントプログラムを作成しよう 1 .....	23
3.3.	Java クライアントプログラムを作成しよう 2 .....	27

図表目次

図 1	環境変数 CLASSPATH の設定 .....	5
図 2	Java プロキシクラスの生成(ターミナルからの実行) .....	6
図 3	新規プロジェクトウィザード:パラメータの設定 .....	8
図 4	Database オブジェクトの作成 .....	9
図 5	プロパティへのアクセス(set/get メソッド) .....	10
図 6	埋め込みオブジェクトの参照 .....	11
図 7	埋め込みオブジェクトの変更 .....	11
図 8	コレクション:参照方法 .....	12
図 9	コレクション:更新方法 .....	12
図 10	コレクション:Iterator を利用した場合 .....	13
図 11	リレーションシップの操作(子供から親へのアクセス) .....	14
図 12	リレーションシップの操作(親から子へのアクセス) .....	15
図 13	クラスメソッドの呼び出し .....	15
図 14	インスタンスメソッドの呼び出し .....	15
図 15	クエリの実行 .....	16
図 16	動的クエリ(ダイナミック SQL)の実行 .....	17
図 17	Caché クラスにある Java メソッド例 .....	18
図 18	Shop パッケージのプロキシクラス生成(ターミナルでの実行) .....	21
図 19	PrintOrder.java 実行例 .....	26
図 20	コンポジットパターン .....	27
図 21	Sample.Party クラス例 .....	27
図 22	Sample.Employee クラス例 .....	28
図 23	Sample.Department クラス例 .....	28
図 24	コンポジットパターン:データ作成 .....	28

## 1. はじめに

本ガイドは、CachéとJavaの連携を中心に、Java側からCachéへアクセスするためのプログラミングの基本を解説します。本ガイドでは、ユーザの皆さんがCachéとJavaでの開発を始めるのに必要な事項を説明していきます。評価版のCachéを実際にインストールして操作しながらお読みいただければ、理解が深まると思います。

また、本ガイドで使用するクラス群はCaché技術ガイド「Cachéオブジェクト操作ガイド」で使用するクラスと同じものも使用します。クラス定義の方法、サーバサイドプログラミングにご興味のある方は、「Cachéオブジェクト操作ガイド」もお読みになることをお勧めいたします。

なお、本ガイドには、Cachéのインストールの解説は含まれていません。

インストール方法解説については、弊社ダウンロードサイト内、  
解説書:Cache2011.1PCKitForWindowsInstallationJPN.pdf をご覧ください。

本ガイドをやり終えれば、CachéとJavaを連携して開発をスタートするのに必要な知識が得られます。さらに情報が必要な方は、オンラインドキュメントをご参照ください。

このガイドでは、CachéとJava、それぞれのクラスを以下のように呼ぶことにします。

Cachéクラス - 主にCachéスタジオ上で定義されるCachéのクラス

Javaプロキシクラス - CachéのJavaプロジェクションによって自動生成されるJavaクラス

Javaクラス - ユーザが作成するCachéにアクセスするためのプログラムを記述したJavaクラス

## 2. Java Binding

### 2.1 Java バインディングとは

Caché Java バインディングにより、Java アプリケーションは Caché サーバに含まれるオブジェクトと相互運用できます。Caché クラス定義に、**Java プロジェクション**を定義すると Caché は、Caché クラスにアクセスするための Java プロキシクラスを生成します。Java プロキシクラスはサーバ上の Caché メソッドに一致するメソッドと、オブジェクト・プロパティのアクセスするためのメソッドを含みます。Java プログラムでは、これらの Java プロキシクラスは通常の Java クラスと同じものとして認識します。生成された Java プロキシクラスは、インターシステムズが提供する Java ライブラリ(cachedb.jar と cachejdbc.jar)を使用し、クライアントとサーバ間の通信を行います。

詳細は以下のステップで解説します。

### 2.2 Java バインディングの準備

#### 2.2.1 Java 環境のセットアップ

Java クライアントの実行環境として、Java SDK のバージョン 1.5 以降を使用する必要があります。

Caché の接続に必要な Java ライブラリは、ファイル **cachedb.jar** と **cachejdbc.jar** に含まれます。Windows システムの場合、このファイルは、以下ディレクトリにあります。

なお、Caché のインストールディレクトリを、インストール時のデフォルトとした場合を記載しています。デフォルトのインストールディレクトリは C:\InterSystems\Cache です。

**C:\InterSystems\Cache\Dev\java\lib\{JDKnn}**

(nn は、JDK のバージョン番号)

この jar ファイルに含まれる情報の詳細は、以下の HTML ファイルから参照できます。

C:\InterSystems\Cache\Dev\java\doc\index.html

Java 環境では、システム環境変数 CLASSPATH を使用して、機能に必要なファイルを見つけます。この変数に **cachedb.jar** と **cachejdbc.jar** のロケーションを含む必要があります。

お使いの環境のシステム環境変数にあらかじめ、CLASSPATH を設定しておくことをお勧めします。Windows の場合 コントロールパネル→システム→詳細設定 タブ→環境変数 から指定できます。このとき必ず以下のように値の最後に「.」(ピリオド)も含めるようにしてください。

変数名	値
CLASSPATH	C:¥InterSystems¥Cache1¥dev¥java¥lib¥JDK16¥cachedb.jar; C:¥InterSystems¥Cache1¥dev¥java¥lib¥JDK16¥cachejdbc.jar;

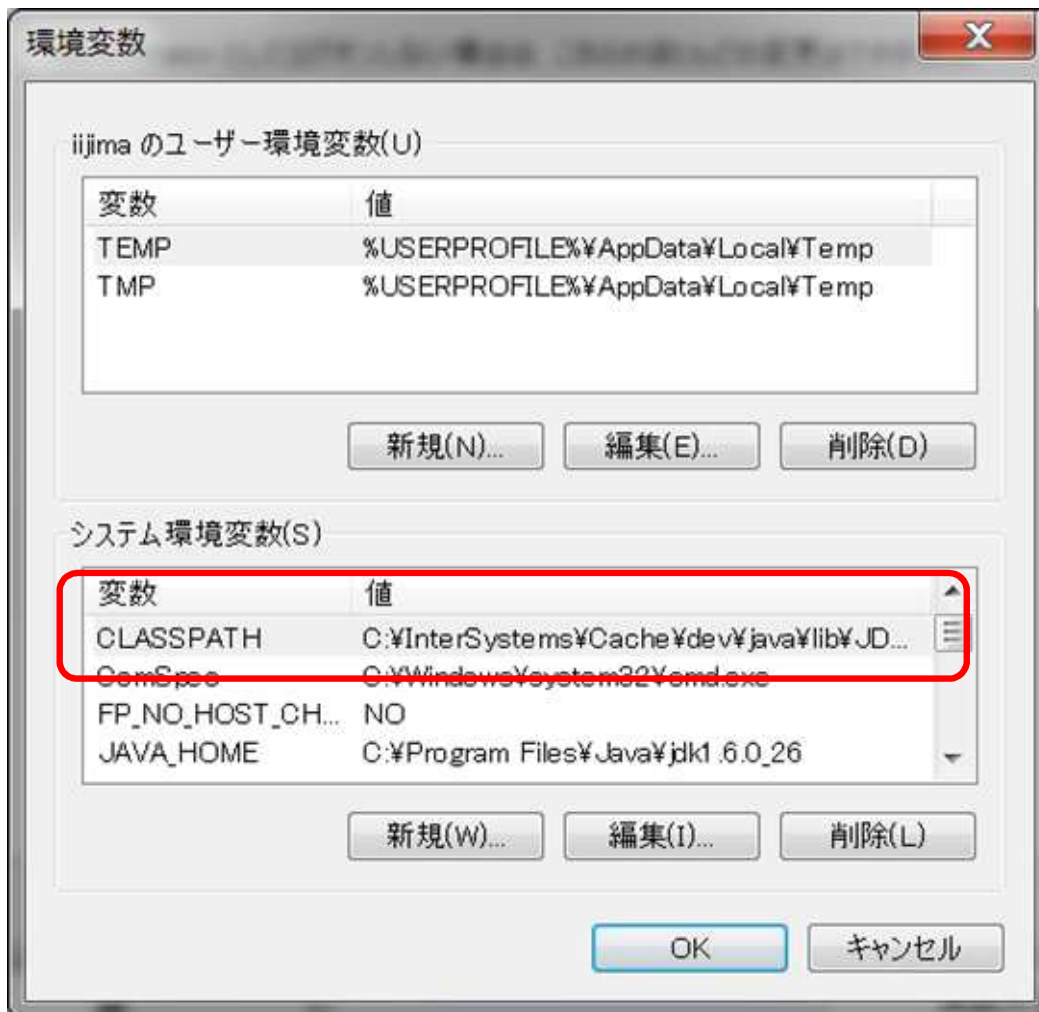


図 1 環境変数 CLASSPATH の設定

### 2.2.2 Java IDE 環境のセットアップ

開発に Java IDE 環境を利用している場合、外部 JAR として `cachedb.jar` と `cachejdbc.jar` を追加する必要があります。

### 2.2.3 サンプルプログラムを動かしてみよう。

Cache の SAMPLES ネームスペースに接続して、データのやり取りを確認します。環境の確認も兼ねて、本ガイドのサンプルを利用しながら確認します。

以下の手順で確認します。

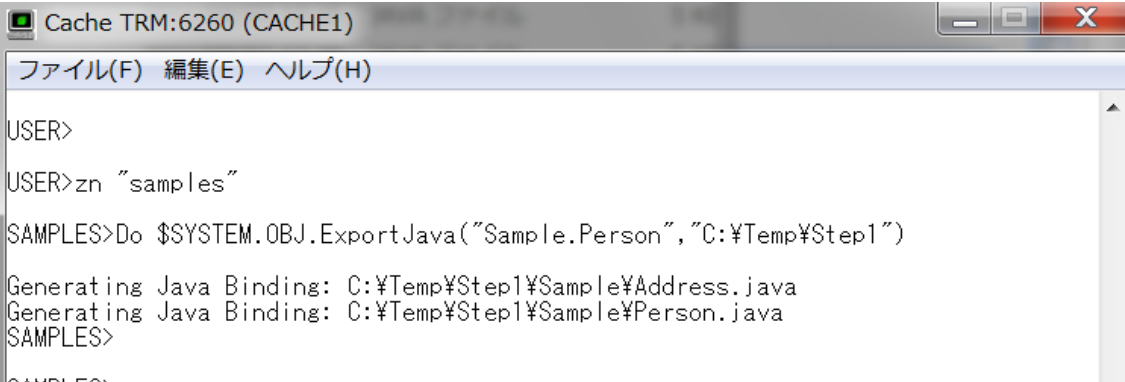
1. 本ガイドのサンプルに含まれる Step1 フォルダを、操作しやすいディレクトリにコピーします。  
例) `c:\¥temp` へコピー

2. `CJTest2.java` クラスを試します。

実行には、Cache の SAMPLES ネームスペースにある Cache クラス: `Sample.Person` と `Sample.Address` に対応した、Java プロキシクラスが必要となります。

Java プロキシクラスを生成するため、Cache ターミナルを起動し、SAMPLES ネームスペースで以下実行します。

メソッドの第 1 引数にはクラス名 (`Sample.Person`) を指定し、第 2 引数にプロキシクラスの生成場所をフルパスで指定します。第 2 引数は、コピーした Step1 フォルダのフルパスを指定します。(例: `c:\¥temp¥Step1`)



```
Cache TRM:6260 (CACHE1)
ファイル(F) 編集(E) ヘルプ(H)
USER>
USER>zn "samples"
SAMPLES>Do $SYSTEM.OBJ.ExportJava("Sample.Person", "C:\¥Temp¥Step1")
Generating Java Binding: C:\¥Temp¥Step1¥Sample¥Address.java
Generating Java Binding: C:\¥Temp¥Step1¥Sample¥Person.java
SAMPLES>
```

図 2 Java プロキシクラスの生成(ターミナルからの実行)

3. Java プロキシクラスと、CJTest2.java クラスをコンパイルします。  
コンパイルを行う Java プロキシクラスは以下の通りです。  
<サンプルのコピー先>%Step1%Sample%Address.java  
<サンプルのコピー先>%Step1%Sample%Person.java
4. CJTest2 クラスを実行します。  
CJTest2 クラスでは、Person クラスのデータを Caché から取得し、画面に出力します。  
実行結果は以下の通りです。  
(サンプルのコピー先を c:%temp としています。)

```
c:%temp%Step1>java CJTest2  
Name: Doe, Joe A  
SSN: 965-68-8308  
DOB: null  
Saved id: 202
```

## 2.2.4 Java Projection の設定

Java Binding のための Java プロキシファイルを出力するためには、二つの方法があります。ひとつは、スタジオで Projection ウィザードを使用して Java Projection を設定することです。

「クラス」→「追加」→「プロジェクション」 から 「新規プロジェクションウィザード」を起動します。プロジェクション名を決定し、次ページにて Java ファイルを出力するディレクトリ(ROOTDIR パラメータ)を指定します。それ以外のパラメータについては Caché Class Reference(クラスドキュメント)を参照してください。

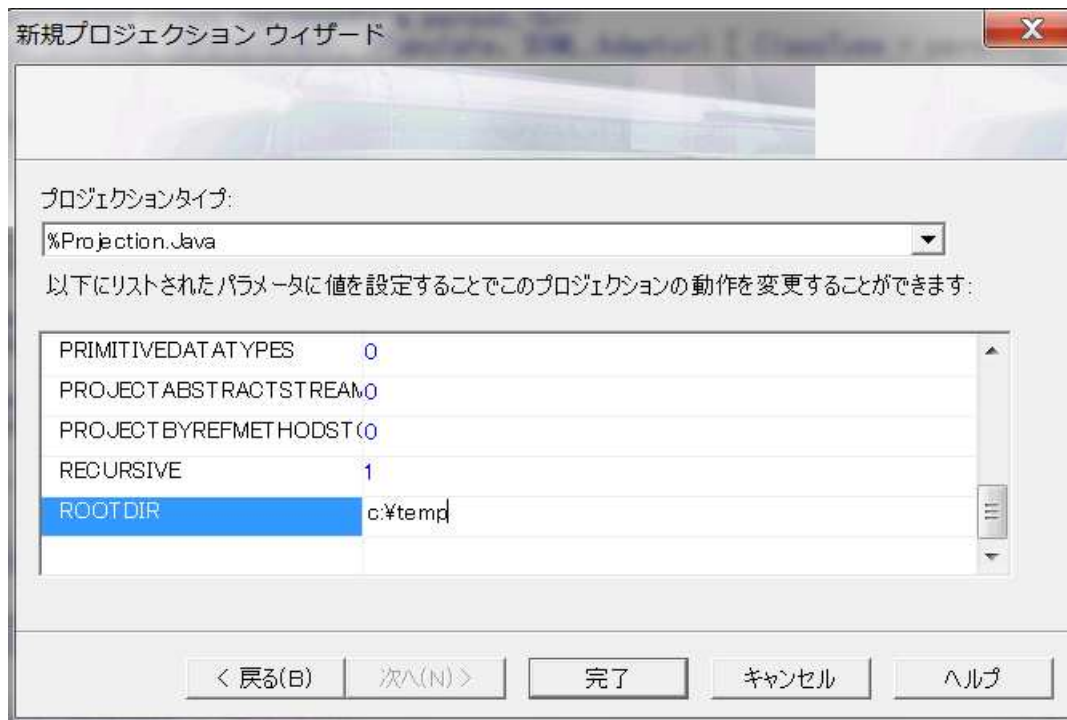


図 3 新規プロジェクションウィザード:パラメータの設定

もうひとつは、CJTest2.java クラスの確認でも実施した、以下システムオブジェクトのメソッドを使用する方法です。

```
$SYSTEM.OBJ.ExportJava()
```

ターミナルから

```
USER>Do $SYSTEM.OBJ.ExportJava("Shop.Customer","C:¥Temp")
```

を実行します。対象クラス、出力ディレクトリは引数を利用して任意に指定できます。

どちらの場合も、そのオブジェクトと関連があるクラス(参照、継承元など)はすべて出力されません。

## 2.3 Java バインディングアクセス方法

この章では Java からどのようにして Cache のオブジェクトにアクセスするのか、順に説明していきます。

### 2.3.1 パッケージのインポート

Java Binding を行うには、cachedb.jar で提供されている com.intersys.objects パッケージをインポートする必要があります。

```
import com.intersys.objects.*;
```

### 2.3.2 Java から Cache への接続

Java から Cache'へアクセスするためには、Cache と接続する必要があります。  
接続のためには、Database オブジェクトを用意し、CacheDatabase クラスの getDatabase メソッドを実行します。

```
Database      dbconnection = null;  
String        url="jdbc:Cache://localhost:1972/SAMPLES";  
  
dbconnection = CacheDatabase.getDatabase (url, username, password);
```

**図 4 Database オブジェクトの作成**

より軽量で高速なアクセス (Java Light Binding) を提供する getLightDatabase メソッドも存在します。

```
dbconnection = CacheDatabase.getLightDatabase(url, username, password);
```

ただしこの方法では、Cache サーバプロセス側にインスタンスが生成されないため、Cache クラスに定義したインスタンスメソッドが呼び出せないなど、いくつかの制限があります。

### 2.3.3 Java から Cache のデータを取得するには

ここからはガイドのサンプルファイルにある Step1 フォルダに含まれる CJTest\*.java クラスから各プロパティへのアクセス方法を、ピックアップして解説します。皆さんの環境でも実際に動かしながら確認してみてください。

#### 2.3.3.1 通常のプロパティ

通常のプロパティはそのままドットシンタックスを使用してデータ取得が可能です。ただし、set メソッド、get メソッドを使用します。

例) サンプルファイル: Step1\CJTest2.java

```
//データの設定 (CJTest2.java)
/* Set some properties */
person.setName("Doe, Joe A");
person.setSSN(generateSSN ());

//データの取得 (CJTest2.java)
/* Fetch some properties */
System.out.println( "Name: " + person.getName() );
System.out.println( "SSN:  " + person.getSSN() );
```

図 5 プロパティへのアクセス(set/get メソッド)

### 2.3.3.2 埋め込みオブジェクト

埋め込みオブジェクトのデータを取得する場合には、まず埋め込みオブジェクトのインスタンスを生成し、通常のプロパティと同様に get メソッドを使用します。また、ドットシンタックスを使用して直接参照することも可能です。

例) サンプルファイル: Step1¥CJTest4.java

```
//インスタンス生成後、参照する場合 (CJTest4.java)
/* Display existing values */
addr = person.getHome();
System.out.println( "Street: " + addr.getStreet() );
System.out.println( "City:   " + addr.getCity() );
System.out.println( "State:  " + addr.getState() );
System.out.println( "Zip:    " + addr.getZip() );

//次のように直接参照も可能です。
System.out.println( "State:  " + person.getHome().getState() );
System.out.println( "Zip:    " + person.getHome().getZip() );
```

図 6 埋め込みオブジェクトの参照

変更も同様です。

```
//インスタンス生成後、変更する場合 (CJTest4.java)
/* Modify some values */
addr.setStreet( "One Memorial Drive" );
addr.setCity( "Cambridge" );
addr.setState( "MA" );
addr.setZip( "02142" );

//次のように直接変更も可能です。
person.getHome().setStreet( "One Memorial Drive" );
person.getHome().setCity( "Cambridge" );
```

図 7 埋め込みオブジェクトの変更

### 2.3.3.3 Collection プロパティ 1 (List/Array)

List/Array のプロパティにアクセスするには以下の 2 種類の方法があります。

1. Java で一般的な `java.util.List` / `java.util.Map` を利用する。
2. Cache が提供する `ListOfObjects` / `ArrayOfObjects` を利用する。

Java Projection の `NEWCOLLECTION` パラメータが 1 の場合は 1. の形式で出力され、0 の場合は 2. の形式で出力されます。現在の `NEWCOLLECTION` パラメータの既定値は 1 になっており、ここではそちらを利用したアクセス方法を紹介します。

参照は以下のように行います。

例) サンプルファイル: `Step1¥CJTest5.java`

```
//リスト数を取得してループ (CJTest5.java)
/* Iterate over the FavoriteColors collection */
colors = person.getFavoriteColors();
for (r = 0; r < colors.size(); r++) {
    System.out.println("  Element #" + r + " -> " + colors.get(r));
}
```

図 8 コレクション:参照方法

次は変更のサンプルです。

```
//最初のデータを削除して新しいデータ(Red)を追加 //(CJTest5.java)
/* Remove the first element */
if (colors.size() > 0)
    colors.remove(0);

/* Insert a new element */
colors.add("Red");
```

図 9 コレクション:更新方法

Iterator を使用して以下のようにアクセスすることもできます。

```
//Iteratorオブジェクトを使用してアクセス
/* Iterate over the FavoriteColors collection */
colors = person.getFavoriteColors();
Iterator ci = colors.iterator();
r=0;
while(ci.hasNext()){
    r++;
    System.out.println("  Element #" + r + " -> " + ci.next().toString());
}
```

図 10 コレクション:Iterator を利用した場合

#### 2.3.3.4 Collection プロパティ 2 (Relationship)

Relationship を利用した Cache クラスを使って、実装方法を確認します。

Cache クラスには、Cache オブジェクト操作ガイドで使用したクラスを利用しています。

(以下、ガイドでは親:子の関係を使って説明しますが、1:多の関連の操作も同様です。)

<Cache オブジェクト操作ガイドのクラス定義をインポートする場合は、弊社 Web サイトの「技術ガイド」のページより、ダウンロードし **USER** ネームスペースへインポートしてください>

Cache 技術ガイド : <http://www.intersystems.co.jp/support/guide.html>

ここでは、**USER** ネームスペースにインポートした (Shop パッケージ以下) Cache クラスを利用します。

これから説明する Java クラスは、後述の P23[Java クライアントプログラムを作成しよう 1] で 1 ステップずつ解説を加えながら作成していただきます。

以下の説明では、Cache クラスが Relationship を利用している時、どのように Java クラスで実装を行うかの概略を説明します。

それでは、Relationship がある場合の実装を確認します。

子から親のデータを参照、更新する場合、子から見て親は一意に決定できるので、通常のプロパティと同じようにアクセスできます。

```
//直接親のデータを参照
customer = po.getCustomer();
customer.getName();
//子から親を設定
po.setCustomer(cust);
```

図 11 リレーションシップの操作(子供から親へのアクセス)

親から子のデータを取得する場合、子の集合の Java データタイプ RelationshipObject を使用します。このクラスは java.util.Map インターフェースを実装しているため、通常の Map オブジェクトとしてもアクセスできます。以下は Iterator を使って子供の要素を次々に取得する方法です。

```
//Iteratorを使用してLineItemを取得
Iterator oi = po.getItems().values().iterator();
while (oi.hasNext()) {
    Shop.LineItem lineitem = (Shop.LineItem) oi.next();
    lineitem.getUnitPrice();
    ...
}
```

図 12 リレーションシップの操作(親から子へのアクセス)

#### 2.3.3.5 Caché のメソッドを実行する

前述(P14 [2.3.3.4 Collection プロパティ 2 (Relationship)])の解説と同様に、USER ネームスペースにインポートした[Caché オブジェクト操作ガイド]の Caché クラスを使用して、以下実装の概略を説明します。

クラスメソッドを実行する場合、引数に Database オブジェクトが必要です。

```
//クラスメソッドの例: POrderクラスのcreateメソッドを呼び出す
Shop.POrder po
= (Shop.POrder) Shop.POrder.create(dbconnection,cust,shipto);
```

図 13 クラスメソッドの呼び出し

インスタンスメソッドの場合は、接続先は明らかですので、Database オブジェクトは必要ありません。

```
//インスタンスメソッドの例: POrder クラスの place メソッドを呼び出す。
//この場合は dbconnetion は必要ありません。
po.place();
```

図 14 インスタンスメソッドの呼び出し

### 2.3.3.6 Caché のクエリを実行する。

ここでは、Caché の **SAMPLE ネームスペース** にある Sample.Person クラスを利用して、クエリの呼び出しを確認します。

例) サンプルファイル: Step1¥CJTest6.java

CacheQuery オブジェクト、java.sql.ResultSet オブジェクトを用意し、実行します。  
 CJTest6.java では Sample.Person クラスの ByName クエリを実行しています。

CacheQuery	cq = <b>null</b> ;
java.sql.ResultSet	rs = <b>null</b> ;

```
//Cache'クエリを作成し、実行結果を ResultSet に格納する
// (CJTest6.java)
/* Create a ResultSet */
System.out.println( "Creating a ResultSet" );
/* Create a CacheQuery */
cq = new CacheQuery( server, "Sample.Person", "ByName" );
/* Execute the query and loop across the returned rows */
rs = cq.execute(query);
while (rs.next()) {
    /* Dump the columns in each row */
    String s = "";
    for (int i = 1; i <= rs.getMetaData().getColumnCount(); i++) {
        if (s.length() > 0) {
            s += ": ";
        }
        s += rs.getString( i );
    }
    System.out.println( s );
}
```

図 15 クエリの実行

動的にクエリを生成して実行することもできます。

例) サンプルファイル: Step1¥CJTest7.java

SQL 文を作成する。

```
/* Create the SQL statement */
sql = "SELECT ID, Name, DOB, SSN FROM Sample.Person
      WHERE Name %STARTSWITH ?";
```

```
//同様に Cache'クエリを作成し、実行結果を ResultSet に格納する
// (CJTest7.java)
/* Create a CacheQuery */
cq = new CacheQuery( server, sql );
/* Execute the query and loop across the returned rows */
rs = cq.execute(query);
while (rs.next()) {
    /* Dump the columns in each row */
    String s = "";
    for (int i = 1; i <= rs.getMetaData().getColumnCount(); i++) {
        if (s.length() > 0) {
            s += ": ";
        }
        s += rs.getString( i );
    }
    System.out.println( s );
}
```

図 16 動的クエリ(ダイナミック SQL)の実行

## 2.4 Java Method

Cache クラス中に Java 言語を用いたメソッドを書くことができます。これは Java メソッドと呼ばれ、Projection された Java クラス内に出力されます。Cache 上ではこのメソッドを単独で実行することはできません。

Java メソッド内では Java 言語を用いて、ロジックを記述しなければいけません。

次章にて Java メソッドを使用したサンプルプログラムを作成します。

```
Method JavaPrintName(tab As %Integer) [ Language = java ]
{
    try {
        System.out.println( "名前" + this.getName() );
    } catch(Exception ex) {
        System.out.println( "Caught exception: "
            + ex.getClass().getName() + ": " + ex.getMessage() );
        ex.printStackTrace();
    }
}
```

図 17 Cache クラスにある Java メソッド例

### 3. Java アプリケーションを作成する

それでは、実際に Java アプリケーションを作成してみましょう。

ここでは

Caché オブジェクト操作ガイド のオブジェクトを利用したアプリケーション

デザインパターンとして有名な Composite パターンを実装したアプリケーション

を作成します。

### 3.1. Caché クラスのインポート / Java プロキシクラスを出力しよう

ここで使用するクラス群は Caché オブジェクト操作ガイドで使用するクラスと同じものです。すでにこのガイドを終了されている方は、インポートの必要はありません。

まだの方は、弊社 Web サイトから Caché オブジェクト操作ガイド用サンプルをダウンロードしてください。( <http://www.intersystems.co.jp/support/guide.html> )

**注意)** 以下作成する Java ファイルは **ネームスペース USER に接続する** として作成しています。別のネームスペースへサンプルをインポートした方は、Java ファイル内 Caché への接続部分を対象のネームスペースに修正し、お試しください。

P8[2.2.4 Java Projection の設定] で説明したように、Java Binding のために Java プロキシクラスを生成する必要があり、生成方法は 2 種類あります。

方法 1 と方法 2 のいずれかを利用して、Java プロキシクラスを生成します。

【方法 1】 Java プロキシクラスを Java Projection を使用して生成する

**Shop パッケージ内、全クラスに Java Projection を追加し、コンパイルを実行してください。**

Projection 定義は以下の通りです。

```
Projection JavaProjection As %Projection.Java;
```

Projection 定義の ROOTDIR パラメータを明示的に指定しない場合、Java プロキシクラス用ファイルは

```
C:¥InterSystems¥Cache¥Devuser¥java¥<ネームスペース名>
```

上記ディレクトリに出力されます。

USER ネームスペースでコンパイルを行った場合は、以下ディレクトリに出力されます。

```
C:¥InterSystems¥Cache¥devuser¥java¥USER
```

Shop パッケージ内クラスは、サブディレクトリ Shop 以下に出力されます。

例) C:¥InterSystems¥Cache¥devuser¥java¥USER¥Shop¥Product.java

最後に、**出力された java プロキシクラスを、Java の環境で全てコンパイルします。**

【方法 2】 ターミナルから \$SYSTEM.OBJ.ExportJava() メソッドを使用して生成する手順は以下の通りです。

### 1. プロキシクラスの生成

Caché クラスに対応する Java プロキシクラスを生成するため、以下メソッドを実行します。

```
Do $SYSTEM.OBJ.ExportJava("Shop.IndvCustomer","C:¥Temp¥Step2")
(第 2 引数は、PrintOrderSample.java クラスがあるディレクトリを指定します。)
```

```
Cache TRM:6260 (CACHE1)
ファイル(F) 編集(E) ヘルプ(H)
USER>
USER>Do $SYSTEM.OBJ.ExportJava("Shop.IndvCustomer","C:¥Temp¥Step2")
Generating Java Binding: C:¥Temp¥Step2¥Shop¥Address.java
Generating Java Binding: C:¥Temp¥Step2¥Shop¥Customer.java
Generating Java Binding: C:¥Temp¥Step2¥Shop¥IndvCustomer.java
Generating Java Binding: C:¥Temp¥Step2¥Shop¥LineItem.java
Generating Java Binding: C:¥Temp¥Step2¥Shop¥POrder.java
Generating Java Binding: C:¥Temp¥Step2¥Shop¥Product.java
USER>
USER>Do $SYSTEM.OBJ.ExportJava("Shop.CorporateCustomer","C:¥Temp¥Step2")
Generating Java Binding: C:¥Temp¥Step2¥Shop¥Address.java
Generating Java Binding: C:¥Temp¥Step2¥Shop¥CorporateCustomer.java
Generating Java Binding: C:¥Temp¥Step2¥Shop¥Customer.java
Generating Java Binding: C:¥Temp¥Step2¥Shop¥LineItem.java
Generating Java Binding: C:¥Temp¥Step2¥Shop¥POrder.java
Generating Java Binding: C:¥Temp¥Step2¥Shop¥Product.java
USER>
```

図 18 Shop パッケージのプロキシクラス生成(ターミナルでの実行)

同様に、Shop.CorporateCustomer クラスについても実行します。

### 2. Java クラスのコンパイル

**出力された java プロキシクラスを、Java の環境で全てコンパイルします。**

【データ作成方法】

インポートしたクラス定義のデータ作成を行うため、Step2¥Shop.Utils.xml を USER ネームスペースにインポートします。インポート後、ターミナルで以下実行します。

```
do ##class(Shop.Utils).CreateData()
```

CreateData()メソッドにあるような、Caché サーバー側のオブジェクト操作について詳細は、オブジェクト操作ガイドをご参照ください。

### 3.2. Java クライアントプログラムを作成しよう 1

このクライアントプログラムは Cache クラス POrder のインスタンス一覧を表示する、PrintOrder.java を作成します。

(Cache 内 Shop.POrder クラスにある printOrderByDate() メソッドと同等の内容を、Java 側から試すため、PrintOrder.java を作成します。)

<PrintOrder.java クラスの完成版は Step2 フォルダにあります。>

それでは

```
public class PrintOrder {  
  
    public static void main(String[] args) {  
    }  
}
```

1 ステップずつプログラムを作成していきます。

- ・ com.intersys.object のインポート  
インターシステムズの提供しているパッケージをインポートする必要があります。その他にも後で必要となるパッケージをここでインポートしておきます。

```
import com.intersys.objects.*;  
  
import java.util.*;  
import java.sql.SQLException;  
  
public class PrintOrder {
```

- ・ 変数の宣言

接続に必要な Database オブジェクト、および url、user、password を設定します。

また、結果を格納する Shop.POrder、クエリに必要な CacheQuery、ResultSet もここで作成しておきましょう。

(接続ユーザ、パスワードはインストール時用意される事前定義のユーザである \_system を利用しています。ユーザ名などはお使いの環境に合わせて適宜変更してお試しください。なお、ユーザ管理については以下ドキュメントより詳細をご参照いただけます。

[http://localhost:57772/csp/docbook/DocBook.UI.Page.cls?KEY=GCAS\\_users](http://localhost:57772/csp/docbook/DocBook.UI.Page.cls?KEY=GCAS_users))

```
public static void main(String[] args) {

    Database          dbconnection = null;
    String            url="jdbc:Cache://localhost:1972/USER";
    String            username="_SYSTEM";
    String            password="SYS";

    Shop.POrder po = null;
    CacheQuery        cq=null;
    java.sql.ResultSet rs=null;
```

- ・ Database コネクションの作成、クエリの実行

CacheDatabase.getDatabase メソッドを実行してコネクションを確立します。

また、Shop.POrder の Extent クエリを作成し、実行した結果を ResultSet に格納します。

(Extent クエリについて: Extent クエリは Cache クラスで自動的に生成されるすべてのインスタンスを返すクエリです。)

```
try {
    dbconnection = CacheDatabase.getDatabase (url, username, password);

    cq = new CacheQuery(dbconnection, "Shop.POrder", "Extent");
    rs = cq.execute();
```

- ・ 結果の取得

ResultSet を while ループでまわし、ID フィールドの値を取得します。Shop.POrder オブジェクトをオープンし、データを取得します。

(オブジェクトをオープンしなくても、クエリの結果に各フィールドは含まれていますが、ここではあえてオブジェクトを使用しています。)

```
while (rs.next()){
    Id id = new Id( rs.getInt("ID"));
    po = (Shop.POrder)Shop.POrder._open( dbconnection, id );

    System.out.println( "【注文番号】: " + po.getOrderNumber() );
    System.out.println( "【顧客名】: " + po.getCustomer().getName() );
    System.out.println( "【合計金額】: " + po.getTotalPrice() );
    System.out.println();
}
```

- ・ RelationShip の取得(親から子)

Iterator を使用して、Shop.POrder オブジェクトの子、LineItem オブジェクトを取得します。

```
//RelationShip(LineItem:One)のデータ取得(Map)
Collection ci = po.getItems().values();
Iterator oi = ci.iterator();

while (oi.hasNext()) {
    Shop.LineItem lineitem = (Shop.LineItem) oi.next();
    System.out.println( "【商品名】: " + lineitem.getProduct().getName() );
    System.out.println( "【数量】: " + lineitem.getAmount() );
}
System.out.println();
```

- ・ オブジェクトの Close  
オブジェクトをクローズします。ループが繰り返され、変数 po には次の Shop.POrder オブジェクトが格納されます。

```
dbconnection.closeObject(po.getOref());  
po = null;  
  
System.out.println("-----");
```

実行結果は以下のようになります。

```
【注文番号】: 0000000001  
【顧客名】: 日本 太郎  
【合計金額】: 118000  
  
【商品名】: デスクトップPC Pentium IV 2.6G  
【数量】: 1  
  
-----  
【注文番号】: 0000000004  
【顧客名】: インターシステムズジャパン  
【合計金額】: 944000  
  
【商品名】: デスクトップPC Pentium IV 2.6G  
【数量】: 10  
  
-----
```

図 19 PrintOrder.java 実行例

<ここまでの Java サンプルプログラムは、Step2 フォルダ 以下にあります(PrintOrder.java)>

### 3.3. Java クライアントプログラムを作成しよう 2

次のプログラムでは、代表的なデザインパターンであるコンポジットパターンを実装します。オブジェクト指向言語 Java と、オブジェクトデータベース Caché の相性の良さがご理解いただけます。

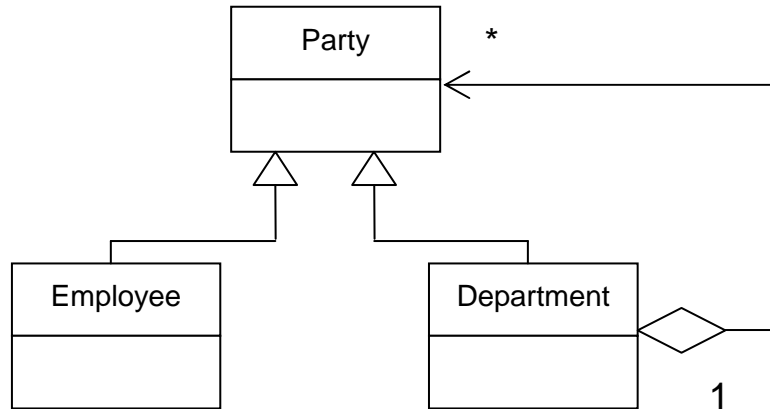


図 20 コンポジットパターン

まず Caché クラスを定義します。

上位の Party クラスを以下のように定義します。

- Java プロジェクション
- Name プロパティ
- JavaPrintName メソッド (Java メソッド)

のみ含みます。JavaPrintName メソッドの実装は継承先で行うので、ここでは何も記述しません。

```

Class Sample.Party Extends %Persistent
{
    Projection JavaProjection1 As %Projection.Java;
    Property Name As %String;
    Method JavaPrintName(tab As %Integer) [ Language = java ]
    {
    }
}
    
```

図 21 Sample.Party クラス例

次に、この Party クラスを継承した Employee クラスを定義します。JavaPrintName メソッドを実装します。

```
Class Sample.Employee Extends Sample.Party
{
Method JavaPrintName(tab As %Integer) [ Language = java ]
{
    try {
        //表示位置の調整
        for (int i=1;i<tab.intValue();i++) {
            System.out.print(" ");
        }
        System.out.println( "Emp: " + this.getName() );
    } catch(Exception ex) {
        System.out.println( "Caught exception: " + ex.getClass().getName() +
": " + ex.getMessage() );
        ex.printStackTrace();
    }
}
}
```

図 22 Sample.Employee クラス例

最後に、Party クラスを継承した Department クラスを作成します。

このクラスには

- Parties プロパティ

を追加します。このプロパティは Collection タイプで、「Department」が「Employee」のインスタンスを複数保持します。

```
Class Sample.Department Extends Sample.Party
{
Property Parties As list Of Party;

Method JavaPrintName(tab As %Integer) [ Language = java ]
{
    try {
        //表示位置の調整
        for (int i=1;i<tab.intValue();i++) {
            System.out.print(" ");
        }
        System.out.println( "Dept: " + this.getName() );
        java.util.Iterator pi = this.getParties().iterator();
        while (pi.hasNext()) {
            Sample.Party party = (Sample.Party) pi.next();
            //下位のコンポーネントの表示位置 +2
            int a = tab.intValue()+2;
            Integer newtab = new Integer(a);
            party.JavaPrintName(newtab);
        }
    } catch(Exception ex) {
        System.out.println( "Caught exception: " + ex.getClass().getName()
+ ": " + ex.getMessage() );
        ex.printStackTrace();
    }
}
}
```

図 23 Sample.Department クラス例

これで Cache クラスの定義は完了です。コンパイルを実行し、Java クラスを取得します。

＜ここまでのサンプルクラス定義は、Step2 フォルダ以下にあります。

(JavaBinding\_practice2.xml) >

**注意** これから作成する Java ファイルは、**ネームスペース USER を接続対象として**作成しています。別のネームスペースにサンプルクラス定義をインポートする場合には、Cache への接続部分を対象のネームスペースに修正してお試しください。

ターミナルを使用して以下のようにデータを作成します。

Department クラスの Parties プロパティには Insert メソッドを使用してデータを追加します。

サンプルクラス定義を (Step2¥JavaBinding\_practice2.xml) インポートされた方は、

Sample.Party クラス内クラスメソッド DataPopulate()を実行し作成することもできます。

```
USER> Set top=##class(Sample.Department).%New()

USER> Set top.Name="ABC 株式会社"

USER> write top.%Save()
1
USER>

USER> Set dep1=##class(Sample.Department).%New()

USER> Set dep1.Name="営業部"

USER> Do top.Parties.Insert(dep1)

USER> write top.%Save()
1
USER> Set sp=##class(Sample.Employee).%New()

USER> Set sp.Name="中村主任"

USER> D dep1.Parties.Insert(sp)

USER> write dep1.%Save()
1
USER>
```

図 24 コンポジットパターン: データ作成

クライアントのコード Partymain.java を作成します。コードは以下のようになります。

```
import com.intersys.objects.*;

public class Partymain {

    public static void main( String[] args )
        throws Exception
    {
        Database          dbconnection = null;
        String             url="jdbc:Cache://localhost:1972/USER";
        String             username="_SYSTEM";
        String             password="SYS";
        Sample.Party      party = null;
        Id                 id = null;

        try {
            dbconnection = CacheDatabase.getDatabase (url, username, password);

            for (int i = 0; i < args.length; i++)
                if (args[i].equals("-id"))
                    id = new Id (args[++i]);

            if (id == null)
                id = new Id( 1 );

            if (!(Sample.Party.exists (dbconnection, id)))
            {
                System.out.println ("There is no Party with id " +
                    id.toString() + " in the database.");
                dbconnection.close();
                return;
            }
        }
    }
}
```

```
//指定されたIDのインスタンスをオープン
party = (Sample.Party) Sample.Party._open( dbconnection, id );

Integer tab = new Integer(2);

//JavaPrintNameメソッドを呼び出し。
//子要素がある場合は、メソッド内で再帰的に呼び出される
party.JavaPrintName(tab);

dbconnection.closeObject(party.getOref());
party = null;

dbconnection.close();
System.out.println( "Disconnected." );

} catch (CacheException ex) {
    System.out.println( "Caught exception0: " + ex.getClass().getName() +
": " + ex.getMessage() );
    ex.printFullTrace(System.out);
} catch (Exception ex) {
    System.out.println( "Caught exception0: " + ex.getClass().getName() +
": " + ex.getMessage() );
    ex.printStackTrace();
}
}
```

<ここまでの Java プログラムは Step2 フォルダ 以下にあります。(Partymain.java)>

P28 図 24 コンポジットパターン:データ作成 の例文に沿ってデータを登録した場合、実行結果は以下ようになります。

(実行時、-id に続けて引数を指定できます。P28 の登録例の中では、Department として「ABC 株式会社」を登録しています。ABC 株式会社の ID を引数に指定すると、以下実行結果が得られます。引数を指定しない場合は、-id 1 を指定した結果が表示されます。)

```
Dept: ABC株式会社
  Dept: 営業部
    Emp: 中村主任
Disconnected.
```

補足 1:

Cache 内に登録したデータの ID を調査したい場合は、管理ポータルを利用します。

管理ポータル→システムエクスプローラ→SQL→SQL スキーマを参照

→<左ペインでネームスペース名を選択>→Sample パッケージ選択

→Party テーブルの「テーブルを開く」