OServ(ObjectServer) 2.1
User's Manual

2012/08/02

uewxiun@pb3.so-net.ne.jp
Takahiko Tanaka

OServ（オブジェクトサーバー）

本書は、OServ（オブジェクトサーバー）Version 2.1に関して、インストール方法、コマンド体系、実行方法、ユーザプログラムの作り込み方法などを説明するものです。OServ（オブジェクトサーバー）は、分散処理を簡単に実装するためのフレームワークとして、ユーザの独立マシン内で実行しているJava言語アプリケーションを、分散処理システムに容易に変換するための機能を提供します。このため、本書では、利用者はプログラミングの基礎的な知識を有していて、Java言語を使用してプログラムを開発ができる事を前提としています。特にJava機能のInterfaceクラスを使用した通信処理クラスの自動生成機能を有しているので、Java言語のInterfaceクラスとabstractメソッドについても知識を有している事を前提としています。

2012/06/11 田中　隆彦

ソフトウェアの著作権

本ソフトウェアは、シェアウェア（有料ソフトウェア）として販売されるもので、正規の使用（機能、仕様の評価のための試用でないもの。）の場合には、購入手続きを行ってください。パッケージ内に含まれるプログラムは、参照、変更、転用は自由ですが、問題発生時の責任は負いかねます。また、基となるプログラムの更新の妨害となる著作権の主張或は、社会的な悪事のためのプログラムの使用も禁止させて頂きます。

問い合わせ

問い合わせは下記のメールアドレスへ直接メールを送信して下さい。

uewxiun@pb3.so-net.ne.jp

動作環境

Java 1.5.0_13

OS　- Mac OS Darwin 9.6.1

Unix, Linux, Solaris Windows

1　ObjectServer(オブジェクトサーバー)の機能

OServ（オブジェクトサーバー）は、従来CORBA製品などで見られる面倒なサーバー検索処理を無くし、単体マシン上でJava言語のクラスを利用するのと同じようにサーバーに登録されたサービスクラスを利用するための仕組みを提供するソフトウェアシステムです。

クライアント・サーバー処理形式システムに於いて、クライアントプログラム内のサーバー呼び出し処理で参照されるサーバークラスをOServ（オブジェクトサーバー）で生成されるサーバースタブクラスと置き換える事によって、面倒なネットワーク間でのデータ処理を一切実装する事無く、（オブジェクトサーバー）を介した分散処理システムに移行させる事ができます。

OServ（オブジェクトサーバー）は、登録されたサービス処理の実行に際して、パーシステントなデータ保存機能も提供しています。このため、サービスは、独立した呼び出しの中で、継続的に値を保持する機能の実装が可能です。

OServ（オブジェクトサーバー）は、登録されたサービスの入出力アクセスを
特定のディレクトリツリーに対してだけに限定できます。

OServ（オブジェクトサーバー）は、登録されたサービスに対して入出力ストリームパラメータを渡すことができます。大規模データの送受信を意識することなく簡単に行って実行することができます。

OServ（オブジェクトサーバー）は、サービスの実行要求の転送機能もサポートしていて、サービス要求受付サーバーと実行サーバーを分離することができます。

OServ（オブジェクトサーバー）は、トランジェント機能としてサービスとサービスに対する要求を同時に転送して、実行することができます。

OServ（オブジェクトサーバー）の上記の機能を利用することで、既存システムに付加的機能を簡単に導入する事もできます。

2　インストール

ファイルは、zipコマンドによりアーカイブされていますので、unzipコマンドを使用するか、直接ファイルの内容を展開して御利用下さい。

```
 # unzip OServ.jar.zip
```

特別にファイルを配置する必要は無く、同胞されているjarファイルをライブラリとして指定して頂ければ動作します。

OServ（オブジェクトサーバー）は、分散処理実現のためのミドルウェアとして、サーバーサイドとクライアントサイドに分かれて動作します。OServ（オブジェクトサーバー）のコマンド、ライブラリは、１つのOServ.jarファイルにまとめられていて、両方のシステムで参照します。

サーバー側の処理

```
# mkdir SERVER
# cd SERVER
# cp ../OServ.jar .
# java -classpath ./OServ.jar oserv.ObjectServer
```

(java -classpath ./OServ.jar oserv.ObjectServer >/dev/null 2>&1 &) デーモンとして起動する場合。

クライアント側の処理

```
# mkdir CLIENT
# cd CLIENT
# cp ../OServ.jar .
```

```
# mv ../SampleServer.jar .　　（一般処理のサンプル）
# mv ../PersistentServer.jar .　（パーシステント処理のサンプル）
```

（サプルサービスのサーバーへの登録。）

```
# java -classpath ./OServ.jar oserv.ObjectTool sample.SampleServer
# java -classpath ./OServ.jar oserv.ObjectTool test.PersistentServer
```

java -classpath ./OServ.jar oserv.ObjectTool -（ホスト名を指定する場合には次のように指定して下さい。）

```
# java -classpath ./OServ.jar -Doserv.service.hostname=localhost oserv.ObjectTool –l
```

（ホスト名を指定する場合には次のように指定して下さい。）

```
# java -classpath ./OServ.jar -Doserv.service.hostname=localhost oserv.ObjectTool -l
```

登録パッケージの更新は、削除せずにそのまま上書きが可能です。

（登録パッケージの削除をするときは、次のように -dオプションを使用して下さい。）
java -classpath ./OServ.jar oserv.ObjectTool -d sample.SampleServer
（遠隔では、localhostの所にネットワーク上のObjectServerを起動した
　ホスト名を指定して下さい。）

mv ../userclient .
cd userclient

　（サンプルの実行）
java -classpath ../OServ.jar:./ sample.SampleClient
(java -Doserv.service.hostname=server_hostname -classpath ../OServ.jar:./
sample.SampleClient)
java -classpath ../OServ.jar:./ test.PersistentClient

上記のクライアント要求では、既にスタブクラスが作成されています。実際にユーザが自分自身のサービスを作成して使用する場合には、実装サービスを定義するInterfaceファイルを基に、スタブクラスを生成する必要があります。生成されたスタブクラスをサーバーに登録されているサービスクラスと同様に利用できるので、クライアント側からは、サービスを意識する事無く簡単に使用する事ができます。下記は、その構築例になります。

　（ファイルを削除します。）
rm test/PersistentServer.java test/PersistentServer.class
java -classpath ../OServ.jar:./ oserv.ObjectGenerator PersistentService
test.PersistentServer
（PersistentServiceは、インタフェースクラス名です。testの下に
PersistentService.classとしてコンパイルしたInterfaceファイルを配置します。）
javac -classpath ../OServ.jar:./ test/PersistentServer.java
java -classpath ../OServ.jar:./ test.PersistentClient

（転送データの検証テストを実行します。）
```
# cd ../../CLIENT              （作成した CLIENT ディレクトリに移動）
# mv ../var .

# java -Duser.home=`pwd`/var/server -classpath ./OServ.jar
oserv.ObjectTool var.VarServer
# java -classpath ./OServ.jar:./var/client var.VarClient all
```
（パラメータの all の指定により次のテストが実行されます。
ネイティブ型の変数の受信テスト　－　1～8
VarExceptionの受信テスト　－　9
文字列オブジェクトの受信テスト　－　10
オブジェクト型の変数の受信テスト　－　11～18
Serializable Objectの受信テスト　－　19
一般オブジェクトの受信テスト　－　20
ネイティブ型の変数の送信テスト　－　21～28
一般オブジェクトの送信テスト　－　29
文字列オブジェクトの送信テスト　－　30
オブジェクト型の変数の送信テスト　－　31～38
Serializable Objectの送信テスト　－　39
Byte配列の受信テスト　－　41
Byte配列の送信テスト　－　42
all を数字に変えれば個々のテストが実行できます。）

正常な場合の実行結果は、変数テストと文字列オブジェクトでは、テスト番号が表示されます。Boolean変数のテストだけは、真偽値のtrueが表示されます。Serializable Objectでは、オブジェクト名が表示されます。Byte配列のテストでは、配列のインデックス番号に初期化されている配列の内容が表示されます。

異常な場合の実行結果は、１から8のテストでは-1、それ以外のテストでは、NULLまたは、例外メッセージが表示されます。

Windowsでの操作の注意点
javaのクラスパス（classpath）の指定で、複数のパスを指定する場合には、コロン（：）ではなく、
セミコロン（；）を使用して下さい。
```
# java -classpath ../OServ.jar;./ test.PersistentClient
```

削除コマンドは、rmではなく、delを、移動コマンドは、mvではなく、move、複製コマンドは、cpではなく、copyを使用して下さい。

3　コマンドリファレンス

この章では、OServパッケージに含まれるコマンドについて説明します。OServパッケージは、全てJava言語で記述されているためコマンドの実行に際しては、JavaVMを次のように実行する必要があります。

```
java –classpath archive.jar:directory class名
```

class名の所がシンタックスで記述される部分になります。

ObjectServer

シンタックス

oserv.ObjectServer

処理概要

ObjectServerは、Java言語で作成されたクラスのメソッドの実行管理を行うサーバープログラムで、ユーザが作成したサービスの登録受付、サービスの実行要求を処理します。内部はスレッド化されており、要求毎に独立スレッドとして処理されています。ObjectServerは、パーシステント機能を実現するためにクラスメソッドの実行の前後で、staticで宣言されたデータの値を復元、退避しています。
登録サービスによるファイルアクセスは、Javaのセキュリティマネージャを登録されたクラスの実行の前後で実行サービスに固有のものに入れ替える事により、user.dirのプロパティに指定されたディレクトリ以下以外には読み込み、書き込み、削除ができないようになっています。登録するサービスクラスの実装者は、Fileクラスの getAbsolutePath() メソッドで、取得したパス名で、ファイルを作成、書き込む事で、サーバー側にデータを保持する事ができます。相対的なパスで、Fileクラスを生成しても、操作対象はカレントディレクトリ以下になりますので、絶対パスへの変換後に、Fileクラスを再生成する、あるいはファイルアクセスクラスを変換されたパスで生成する必要があります。

ObjectTool

シンタックス
　oserv.ObjectTool [-d class | -l | -L | class]

処理概要
　ObjectTool コマンドは、ObjectServerにユーザが作成したサービスクラスを登録するためのコマンドです。ObjectToolは、引数として登録するクラス名とそれに、パッケージ名を追加したクラスのパス名を必要としています。ユーザは、サービスクラス名と同じファイル名の.jarファイルの中に関連クラスと接続のためのインタフェースファイルをコンパイルして入れておく必要があります。

　class
　　サービスクラスのパッケージ名を含むクラス名を指定します。Jarフォーマットのアーカイブファイルのファイル名が、サービスクラス名、または、サービスクラスのパッケージ名であると仮定していて、カレントディレクトリ下で、その名前のファイルを検索して、ObjectServerに登録要求を発行します。

　-d class
　　登録時の指定と同じものを指定します。ObjectServerに削除要求を発行します。

　-l
　　ObjectServerに参照要求を発行します。

　-L
　　ObjectServerにリストの参照要求を発行して、forwardingリストを表示します。

ObjectSystem

シンタックス

oserv.ObjectSystem [-p | -s] class

処理概要

このコマンドは、サービス要求の実行タイプを並列実行（平行実行）と逐次実行で切り替えるためのコマンドです。逐次実行タイプのサービスでは、サービス実行中の次のサービスの実行は実行中エラーになります。

-p

並列実行タイプの指定です。

-s

逐次実行タイプの指定です。

class

登録されたクラス名

ObjectKill

シンタックス

oserv.ObjectKill [-l | ids | -s]

処理概要

このコマンドは、システムサービス要求を発行して、プログラムのサービス要求を処理しているスレッド情報を返すあるいは、スレッドの強制終了を要求するためのコマンドです。オプションの指定の無い場合に、内部のスレッドのリストを表示します。

-l

サービススレッドを生成して、無限ループの実行を要求するオプションです。

-s

システムスレッドを参照します。

ids

強制終了を指定するスレッドのIDを指定します。

ObjectService

シンタックス
　oserv.ObjectService [-f target file] service

処理概要
　ObjectServerから、serviceで指定されたユーザサービスのアーカイブファイルを取得するためのコマンドです。

-f target
　このオプションが指定されると、サービスのアーカイブファイルから、指定されたファイルだけを取り出します。

ObjectGenerator

シンタックス
　oserv.ObjectGenerator interface　class

処理概要
　ユーザプログラムからサービスを利用する場合に、ネットワーク経由で呼び出せるようにユーザプログラム側で、あたかも呼び出されるサービスプログラム本体のように動作するスタブクラスを生成するためのコマンドです。

引数としてインタフェースクラス名とスタブのクラス名を取ります。サーバー側のサービスがパッケージを持っている場合には、生成されるスタブファイルではなくパッケージ階層を持ったクラス名で指定する必要があります。インタフェースファイルがパッケージと同じ名前のディレクトリの下にある場合で、実行者が同じディレクトリ下でコマンドを実行する場合には、スタブファイルあるいはインタフェースファイルにパッケージ名指定せずに実行できます。生成されるスタブファイルは実行したディレクトリ下に配置されます。

インタフェースファイルがパッケージと同じ名前のディレクトリの下にあり、実行者がその親ディレクトリでコマンドを実行する場合にはインタフェースクラス名にパッケージのクラス階層を指定して実行する事ができます。

インタフェースファイルが実行者が親ディレクトリでコマンドを実行してインタフェースファイルがパッケージと同じ名前のディレクトリではなく、実行者が操作している親ディレクトリにいるあるいはパッケージディレクトリが無い場合には、パッケージディレクトリを作成してインタフェースファイルを複製し、スタブクラスを生成します。

ObjectGeneratorには、インタフェースの読み込み機能があり、OServに既にサービスが登録されている場合に、OServからインタフェースファイルを自動取得してパッケージディレクトリを作成して格納し、サービスクラスを生成します。

　interface
　　呼び出しメソッド情報のために読み込まれるインタフェースファイル。（パッケージ階層があり、classで指定しない場合には、/で区切り、指定する。）

　class
　　サービスクラスのパッケージを含むクラス名interfaceで指定する場合、あるいは、パッケージ名のディレクトリ下で実行する場合にはパッケージ名は省略可能。

4　アーキテクチャー

OServ(ObjectServer)は、図１．クラスの関係図に示されるように、内部の呼び出しをネットワーク経由の呼び出しに変換するための機構を備えたソフトウェアシステムです。

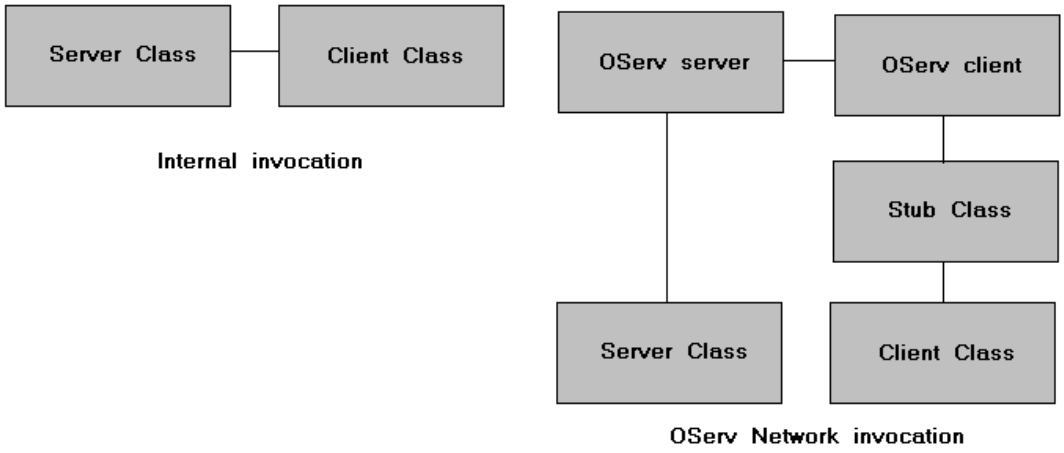

図１．クラス関係図

図１．クラス関係図の中で、OSerは、ObjectServerクラスを示しています。OServ clientは、ObjectClientとして実装されています。ObjectTool は、Serverクラスと ObjectServerサーバーとの間の接続のためのモジュールの登録要求を実行します。Stub Classは、ObjectGeneratorコマンドで、インタフェースクラスを基に生成されるクラスです。ObjectClientは、このクラスの基底クラスとして取り込まれ、Stub Classの各メソッドの中で使用される通信メソッドを提供しています。

上記のようにサーバー処理とクライアント処理は、Java言語のInterfaceクラスを介して呼び出す事を前提としています。Java言語のInterfaceクラスは、メソッド呼び出しのためのabstract宣言をまとめて行うための機能です。abstract宣言は、遅延バインディングのための、実装制約として実態の実装を強制するために使用されます。Implementsキーワードといっしょにクラス宣言に付加することで、実装の不整合を検出することができます。

counterメソッドを持ったBase Interfaceクラス宣言は、次のようなものです。

```
    public interface Base {
        public String counter();
}
```

上記のように宣言することでcounterは、abstract宣言したメソッドと同じになります。

Base Interfaceクラス宣言を使用したSampleクラス実装する場合には、次のように実装します。

```
    public class Sample extends Object implements Base {
        public static int count;
        public String counter() {
            counter = counter + 1;
            return String.valueOf(counter);
        }
    }
```

上記のcounterのメソッドが実装されない場合には、コンパイル時に異常が発生します。

Interfaceクラス宣言は、オブジェクト（クラス）使用時に、仮想的なものとしてアクセスする方法も提供していて、次のようにInterfaceクラスのメソッドとして呼び出し、実際のクラスを参照しないように実装することができます。

```
public class Test {
    public void Exec(Base operator) {
        System.out.println(“Counter “ + operator.count());
    }
}
```

登録されたサービスは、OServ(ObjectServer)内部で実行されますが、危険な処理ができないように、実行の前後でSecurityManagerを入れ替えて、ファイルアクセスを制限しています。JavaVM内のファイルアクセスシステムでは、ファイルアクセス時にSecurityManagerを参照してアクセス制限を実施しているため、SecurityManagerのクラスを実装する事により、ファイルアクセスを制限することができます。また、登録されたサービス内ではSecurityManagerの入れ替えを禁止しているためサーバーシステムの安全も保障されています。

5　サービスの実装

5.1　実装方法

処理の分散
利用者が分散処理システムを構築しようとする場合、最初にシステムの機能をサーバー処理とクライアント処理に分割することが必要です。クライアント側のコマンド処理は主にユーザインタフェース処理と機能の前処理、後処理が中心の処理として、サーバー側のサービス処理は、主要な機能処理として一般的に実装します。

この際、サービスを提供する側とサービスを利用する側で呼び出しインタフェースの整合性を保つためにjava言語のinterfaceクラスの機能を利用して定義する事を本システムでは前提としています。サービス側で提供されるメソッドで、コマンド側から呼び出されるメソッドは、全てこのinterfaceクラスで、実装が約束されている前提で処理されています。interfaceクラスに登録されていないメソッドをクライアントから呼び出している場合には、分散処理構築時に異常が発生するので、注意が必要です。

Interfaceクラスの定義
上記のように処理を分割して、サーバー側のサービス処理を決めた場合に、
interfaceクラスの実装は次のようになります。

```
public interface PersistentService {
	public String counter();
}
```

interfaceクラスで定義されたメソッドは、abstract宣言をしなくてもabstract宣言しているメソッドとして、未実装の場合に、コンパイル時にエラーとなります。

サービスの実装
サーバー側のサービス処理では次のように共通メソッドのためにinterfaceクラスを指定して下さい。

```
public class PersistentServer extends Object implements PersistentService {

	public String counter() {
	}
}
```

OServメソッドの実装
OServオブジェクトサーバーは、初期化処理のユーザイグジットを持っています。ユーザは、主たる ObjectServerクラス、OServ()メソッドを実装するだけで、実行権を得ることができます。

OServ()メソッドは、publicとして、パラメータと戻り値を取らず、下記の様に実装されます。

```
static boolean initial_flag;
public void OServ() {
        Initial_flag = true;
}
```

OServ()メソッドは、OServオブジェクトサーバー登録時と、起動時に一度だけ実行されるもので、異常発生時には無視されます。ユーザモジュールのサーバー側で、外部からの操作が必要な場合に有効です。

5.2　分散処理への移行

スタブファイルの生成
クライアント側のコマンド処理では、上記で定義したinterfaceクラスを使用して、サービスとの接続用のスタブクラスを実際のサービスクラスと同じ名前で生成して使用します。この接続用のスタブクラスを生成するために、ObjectGeneratorコマンドを下記のように実行して下さい。最初のパラメータのPersistentServiceは、読み込まれるinterfaceクラスです。第二のパラメータのPersistentServerは、testパッケージのPersistentServer.javaとして作成されるスタブクラスのクラス名の指定になります。

```
# cd test
# java -classpath ./OServ.jar oserv.ObjectGenerator PersistentService PersistentServer
```

上記の記述で生成された仮想のサービスクラスは、サーバー側のコマンド処理プログラムで次のようにして使用できます。

>>PersistentClient.javaファイル

```
package test;

import java.lang.*;
import java.io.*;

public class PersistentClient extends Object {
	PersistentServer pserv;

	public void PrintCounter() {
		String vl;

		pserv = new PersistentServer();
		vl = pserv.counter();
		System.out.println("Persistent Counter " + vl);
	}

	public static void main(String[] arg) {
		PersistentClient pcl = new PersistentClient();

		pcl.PrintCounter();
	}
}
```

サービス側でパーシステントにデータを保持する場合には、static変数として宣言する必要があります。

>>PersistentServer.javaファイル

```
package test;

import java.lang.*;
import java.io.*;

import test.*;

public class PersistentServer extends Object implements PersistentService {
	public static Integer cvalue = new Integer(0);

	public String counter() {
		int ct = cvalue.intValue();
		ct = ct + 1;
		cvalue = new Integer(ct);
		return String.valueOf(ct);
	}
}
```

作成されたサービス側のクラスは、インタフェースクラスは、コンパイルしてclassファイルを生成します。

```
# cd test
# javac -classpath ../ PersistentServer.java
```

そしてclassファイルをjarフォーマットでアーカイブして0Servへ登録できる形式にして下さい。アーカイブの仕方は次のようになります。

```
# jar cvf PersistentServer.jar test/PersistentServer.class test/PersistentService.class
```

サービスクラス名とアーカイブで作成されるjarファイルの名前は一致させて下さい。

OServへの登録には次のObjectToolコマンドを使用して下さい。

```
# java -classpath ./OServ.jar oserv.ObjectTool localhost/test.PersistentServer
```

ObjectToolの前の部分はホスト名で、後ろの部分は、対象となるパッケージとクラス名である、test.PersistentServerを指定して下さい。

5.3　ストリームパラメータ

OServでは、ストリームオブジェクトを呼び出すメソッドのパラメータとして指定することができます。OServによりメソッドパラメータはネットワーク転送されるために、Serializable のクラスオブジェクトに限られています。OServでは、入力ストリームの場合にはファイルの内容を転送してサービス起動処理内で入力ストリームを内部生成して呼び出しメソッドにパラメータとして受け渡してします。出力ストリームの場合には、内部生成した出力ストリームに書き出されたデータを、ネットワーク転送して、指定された本来の出力ストリームに書き出します。パラメータとして指定可能な入出ストリームは下記のものです。

java.io.InputStream
java.io.FileInputStream
java.io.DataInputStream
java.io.BufferInputStream
java.io.ByteArrayInputStream
java.io.PushbackInputStream
java.io.SequenceInputStream
java.io.LineNumberInputStream
java.io.FilterInputStream

java.io.OutputStream
java.io.FileOutputStream
java.io.DataOutputStream
java.io.BufferOutputStream
java.io.ByteArrayOutputStream
java.io.FilterOutputStream
java.io.PrintStream
java.io.PipedOutputStream

5.4　トランジェント機能

トランジェント機能は、実行環境と共有ディレクトリを提供するための機能です。通常のサービスを使用するクライアントの起動時に、oserv.transient.homeプロパティを使用して、サーバーのアーカイブが置いてあるディレクトリを指定することで、登録操作を自動化され、共有ディレクトリ下でプログラムの実行することができるようになります。実際の実行形式は次のようなものです。

```
# java –Doserv.transient.home=`pwd` -classpath ../OServ.jar:./
sample.SampleClient

# ls
SampleServer.jar sample
```

5.5　セキュリティ機能

ファイルアクセス手順
利用者が作成するOServ（オブジェクトサーバー）サービスの内部で、ファイルを操作する場合には、実際に動作するサーバー側にファイルが作成されます。OServでは、利用者プログラムには、user.dirプロパティに指定されたディレクトリ以下のファイル以外にはアクセスできないように制限されているために、サービスプログラムの実装者は、必ず、FileクラスのgetAbsolutePath()メソッドを使用してアクセス可能ディレクトリ位置を取得して操作する必要があります。ヌル文字列をパラメータとして指定して呼び出す事により、user.dirプロパティで指定されている位置を取得するか、自分の使用するパスをgetAbsolutePath()メソッドで変換する必要があります。

```
File fd = new File("");
String uer_dir = fd.getAbsolutePath();
fd = new File(user_dir + File.separator + file_path);
```

あるいは

```
File fd = new File(file_path);
String absolute_path = fd.getAbsolutePath();
fd = new File(absolute_path);
```

実際のセキュリティのアクセス異常は、読み込み、書き込み、削除等のファイル操作時に発生します。

6　コンフィグレーション

OServ（オブジェクトサーバー）は、TCP/IPのソケットを使用した動的なオブジェクト実行サービスプログラムです。

```
oserv.table.size=100
oserv.debug.level=3
oserv.request.port=2000
oserv.connection.timeout=10000
oserv.retry.count=10000
oserv.max.thread=40
oserv.buffer.maximum=1000
oserv.service.hostname=localhost
oserv.log.file=tty
oserv.user.work=userwork
oserv.forward.servers=oserv://localhost:2001/
oserv.forward.clients=oserv://localhost:2002/
oserv.forward.tunnel=1
oserv.forward.refuse=oserv://other/
oserv.request.refuse=oserv://other/
oserv.object.refuse=oserv://other/
oserv.system.forbidden=0
oserv.message.language=jpn
```

oserv.table.size は、登録用テーブルサイズの初期値と拡張単位を指定するためのプロパティです。テーブルがいっぱいになると同数のエントリ分だけ拡張されます。

oserv.debug.levelは、oservのシステムメッセージの出力を指定するためのプロパティです。3を指定するとログ情報が出力され、5が指定されるとデバッグ情報が出力されます。

oserv.object.portは、oservのオブジェクト（サービス）の登録を受け付けるためのソケットポートを指定するためのプロパティです。

oserv.request.portは、oservの実行要求を受け付けるためのソケットポートを指定するためのプロパティです。

oserv.connection.timeoutは、Javaの java.net.Socketのタイプアウト時間を指定するためのプロパティです。

oserv.retry.countは、ネットワーク受信時のリトライ回数の上限値を設定するためのプロパティです。

oserv.max.threadは、スレッド数の上限値を指定するためのプロパティです。10以上、1000以下の整数で指定可能です。

oserv.buffer.maximumは、データの受信バッファのサイズを指定するためのプロパティです。単位はバイトサイズで指定します。

oserv.service.hostname=localhostは、クライアント側でObjectServerが動作しているホストを指定するためのプロパティです。

oserv.log.fileは、oserv.debug.levelの設定に基づいて出力されるメッセージの出力先を指定するためのプロパティです。ttyの指定は、標準出力への出力の指定として解釈されます。また、/で始まらないパスの指定は、oserv.work.pathで指定された作業ディレクトリの下の階層として扱われます。通常は、ObjectServerを起動したディレクトリの下になります。

oserv.work.pathは、ObjectServerが動作するディレクトリを指定するためのプロパティです。登録されたオブジェクトを管理するためのpersistentディレクトリがその下に作成されます。

oserv.user.workは、ObjectServerが登録されたオブジェクトのサービスがファイルを作成するディレクトリを指定するためのプロパティです。デフォルトでは、userworkとなり、oserv.work.pathで指定されたディレクトリの下に作成されます。絶対パスでは、そのままのディレクトリで扱われますが、相対パスでは、常に、作業ディレクトリの下に作成されます。

oserv.forward.tunnel は、サービスを転送する段数を制限するためのプロパティです。値を１に設定することで、転送サービスを行うホストをサーバーとして設定することを禁止できます。

oserv.forward.servers は、サービスを転送する先のサーバーをリストで指定するためのプロパティです。クライアントホストは、起動時に指定されたサーバーから情報を取得します。サーバーのリストは、カンマでくぎったURI表現のリストです。

oserv.forward.clients は、転送要求をするクライアントをリストで指定するため

のプロパティです。 OServ（オブジェクトサーバー） は、サービスの登録を受け付ける毎に、クライアントマシンに情報を通知します。クライアントのリストは，URI表現で、カンマで区切られた文字列で指定します。

oserv.request.refuse は、サービス要求を拒否するネットワーク、マシンを指定するためのプロパティです。ネットワークを指定する場合には、URI表現にクエリー文字列として ‘?netmask=0xffff0000’ を指定してください。Netmaskキーに続く番号は、ネットワークとして識別する場合の有効ビットの指定です。

oserv.object.refuse は、サービスの登録、削除要求を拒否するネットワーク、マシンを指定するためのプロパティです。ネットワークを指定する場合には、URI表現にクエリー文字列として‘?netmask=0xffff0000’を指定してください。Netmaskキーに続く番号は、ネットワークとして識別する場合の有効ビットの指定です。

oserv.forward.refuse は、サービスの転送情報に関する要求を拒否するネットワーク、マシンを指定するためのプロパティです。ネットワークを指定する場合には、URI表現にクエリー文字列として ‘?netmask=0xffff0000’ を指定してください。Netmaskキーに続く番号は、ネットワークとして識別する場合の有効ビットの指定です。

oserv.system.forbiddenは、サービスの実行のために起動されているスレッドを参照、強制終了させるシステムサービスを有効にするかどうかを指定するためのプロパティです。０が指定された場合には、有効で、１が指定されている場合には無効になります。

oserv.message.languageは、システムメッセージの言語を指定するためにプロパティです。日本語jpnと英語engの２つのタイプが指定可能です。

上記の通常のコンフィグレーションの他に、次のプロパティが使用できます。
oserv.work.path=/Users/uewxiun/Java/WORK

Appendix A. カウンタサンプル

カウンタサンプルは、パーシステントな値を扱うサンプルとして、カウンタの整数値を操作する関数群を集めたものです。分散処理に変換されたサービス側とクライアント側の間でint整数型のデータを送受信できる事を確認するために御使用下さい。

インストレーションは、パッケージファイルOServ.jar.zipには、counterディレクトリ内にCounterServer.jarファイルが含まれています。このアーカイブファイルには、OServに登録するためのサービスとして実装されたsample.CounterServerと、そのインタフェースを定義したsample.CounterFunctionクラスのソースプログラムとコンパイルされたclassプログラムがアーカイブされています。
また、counterディレクトリの下のuserclientディレクトリには、そのサービスを利用するCounterClient及びCounterClient2プログラムが含まれており、
oserv.propertiesを含まない場合と含む場合をそれぞれのクライアントプログラムでテストできるようになっています。

クライアントプログラムから呼び出すためのスタッププログラムは、
CounterFunction.jarとsampleディレクトリの中に含まれていますが、
oserv.propertiesをJarファイルが含んでいない場合をテストする場合には、
CounterFunctionOrg.jarを使用します。

サービスの登録

```
# cd counter
# java -classpath ../OServ.jar oserv.ObjectTool sample.CounterServer

# cd userclient
# java -classpath ../../OServ.jar:./CounterFunction.jar:./ CounterClient clear
Value 0

# java -classpath ../../OServ.jar:./CounterFunction.jar:./ CounterClient2 clear

# mkdir oserv
# cd oserv
# jar xvf ../CounterFunction.jar oserv.properties
```

ユーザプログラムの内部での呼び出しは次のようになっています。

```
cs = new CounterServer();

try {
        cs.readPropertyFromJar("CounterFunction.jar");
} catch (Exception ep) {
        System.out.println("CounterFunction");
        return;
}
```

サーバースタブが共通メソッドとして提供されるreadPropertyFromJar()メソッドを使用して、CLASSPATH（-classpathオプション）で指定されたリスト内のJarファイル内部で、oserv.propertiesを検索して読み込みます。上記の例では、oserv.properties内でデバッグモードの設定があるために、実行結果だけでなく、デバッグメッセージが出力されます。

Appendix B. Forward （転送情報）のためのコンフィグレーション

転送要求が 1 段階（直接サービスするマシンだけをサーバーとする構成）の場合

Client side configuration
oserv.forward.tunnel=1
oserv.request.port=2003
oserv.forward.servers=oserv://127.0.0.1:2013/

Server side configuration
oserv.request.port=2013
oserv.forward.clients=oserv://127.0.0.1:2003/

転送要求が多段階（中継サービスするマシンもサーバーとして構成）の場合

Client side configuration
oserv.forward.tunnel=1
oserv.request.port=2003
oserv.forward.servers=oserv://127.0.0.1:2013/
oserv.forward.clients=oserv://127.0.0.1:2023/

Server side configuration
oserv.request.port=2013
oserv.forward.clients=oserv://127.0.0.1:2003/

Second client side configuration
oserv.forward.tunnel=2
oserv.request.port=2023
oserv.object.servers=oserv://127.0.0.1:2003/

Appendix C. オブジェクトディレクトリサンプル

オブジェクトディレクトリサンプルは、実用的なアプリケーションでの使用例として、ファイルの管理機能を実現したシステムを分散システムに変換するサンプルとして作成されました。

ここでは、ローカルで動作するシステムを分散処理システムに0Serv（オブジェクトサーバー）を使用して移行するサンプルとして作成されました。

```
# cd ODIR
```

ローカルでの動作

```
# mkdir WORK
# cd WORK
```

（ユーザアカウントとパスワードを登録します。）

```
# java -classpath ../odir.jar odir.Register ttanaka 'password'
# ls
ttanaka        user_database
# java -classpath ../odir.jar odir.User ttanaka 'password'
Entry
ttanaka
# java -classpath ../odir.jar odir.Register koizumi 'koizumi33'
# java -classpath ../odir.jar odir.User ttanaka 'password'
Entry
koizumi
ttanaka
# java -classpath ../odir.jar odir.Erase ttanaka 'password'
# java -classpath ../odir.jar odir.User koizumi 'koizumi33'
Entry
koizumi
```

0Servを使用した分散処理の実現

```
# cd ODIR
```

（サーバーにサービス処理を登録）

```
# java -classpath ../OServ.jar oserv.ObjectTool odir.DirectoryServer
# java -classpath ../OServ.jar oserv.ObjectTool -l
Entry
odir.DirectoryServer
```

（ネットワーク接続用スタブの作成／コンパイル）

```
# mkdir DIST
# cd DIST
# jar xvf ../DirectoryServer.jar odir/DirectoryService.class
```

```
# java -classpath ../../OServ.jar oserv.ObjectGenerator DirectoryService
odir.DirectoryServer
# javac -classpath ../../OServ.jar:./ odir/DirectoryServer.java
```

（ユーザディレクトリと名簿はサーバー側に作成）

分散処理で実行

```
# java -classpath ../../OServ.jar:./:../odir.jar odir.Register ttanaka 'password'
# ls
odir

# java -classpath ../../OServ.jar:./:../odir.jar odir.User ttanaka 'password'
Entry
ttanaka
```

Command Reference

```
# cd ODIR
# cp ../OServ.jar .
```

ユーザの登録

```
# java -classpath ./OServ.jar:./client:./odir.jar odir.Register ttanaka 'password'
```

ユーザの参照

```
# java -classpath ./OServ.jar:./client:./odir.jar odir.User ttanaka 'password'
```

ユーザの削除

```
# java -classpath ./OServ.jar:./client:./odir.jar odir.Erase ttanaka 'password'
```

ディレクトリの作成

```
# java -classpath ./OServ.jar:./client:./odir.jar odir.Directory odir://localhost/TEST ttanaka
'password'
```

ディレクトリの参照

```
# java -classpath ./OServ.jar:./client:./odir.jar odir.List odir://localhost/TEST ttanaka
'password'
```

ディレクトリの削除

```
# java -classpath ./OServ.jar:./client:./odir.jar odir.Delete odir://localhost/TEST ttanaka
'password'
```

ファイルの保存
java -classpath ./OServ.jar:./client:./odir.jar odir.Put ttanaka 'password' Check.class odir://localhost/Check.class

ファイルの取り出し
java -classpath ./OServ.jar:./client:./odir.jar odir.Get ttanaka 'password' odir://localhost/Check.class Check.class

ファイルの削除
java -classpath ./OServ.jar:./client:./odir.jar odir.Remove odir://localhost/Check.class ttanaka 'password'
　(ファイル名として ‘odir://localhost/PP.*’ を指定する事で、*を任意の文字に置き換えたファイルを一括して削除する事ができます。）

ファイルの実行（クラスファイルの場合）
java -classpath ./OServ.jar:./client:./odir.jar odir.Execute ttanaka 'password' odir://localhost:2001/Check.class Check outValue1 2 3 2.2 3.3

ファイルの実行（Jarのアーカイブファイルの場合）
java -classpath ./OServ.jar:./client:./odir.jar odir.Execute ttanaka 'password' odir://localhost/Check.jar Check outValue1 1 5 2.4 3.4