Javaによるwebプログラミング入門

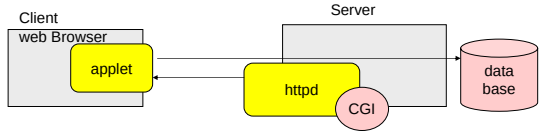
佐藤

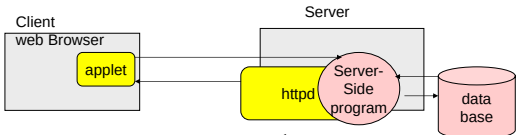
Javaによるwebプログラミング

- Applet
- Servlet – ３層モデル
- JSP
- SOAP
- WebService
- XML
- J2EE

2層モデルから３層モデルへ

- 「サーバサイド」のプログラム





JSPとServlet

- システムを変更して、クライアントに影響を及ぼすような変更があった場合にはクライアント側のプログラムを変更しなくてはならない。（アプレットを使えば、改善されるが）

Server-sideのプログラムの変更のみでよい

- 基幹の大切なデータベースへのアクセスの権限を直接クライアントにももたせなくてはならない。このようなことは特に不特定多数がアクセスする可能性があるインターネットでは問題になる。

Server-sideのプログラムで管理

Servlet

- webサーバ側にJavaVMを組み込み（組み込みの方法はいろいろある）、サーバ側で実行される
- 基本的に、Servletで記述されるプログラムはHTMLを生成するプログラム
- 起動の方法
  - webサーバの起動時にサーバに名前を登録し、その名前で指定する。
  - URLでServletを指定する。
  - <servlet> …</servlet>のタグで、HTMLファイルの中で指定する。
- 利点
  - CGIに比べて、プロセスを起動するオーバーヘッドがないため高速となる。
  - Javaを開発言語に使うことができるため、開発効率を向上させることができる。
- 例： Apache Tomcat

ＵＲＬパラメータを利用するServlet

```
public class UrlParamterMsg extends GenericServlet {
  public void services(ServletRequest request,
                       ServletResponse response)
      throws ServletException, IOException {
      String style = request.getParameter("style");
      response.setContentType("text/html; charset=...");
      PrintWriter pw = response.getWriter();
      pw.println("<html><head>");
      pw.println("<title> ... <title>");
      ....
      pw.println("</html>");
      pw.flush();
      pw.close();
  }
  public String getThisTime(String style) {
 ...
 }
}
```

URLパラメータを取得するServlet

- GenericServlet: httpに依存しない場合
- GCIと同じように、ＵＲＬのパラメータを取ることができる。

  http://host:port/servlet/servlet-name?name=value
- メソッド javax.servlet.ServletRequest
  - getParameterString(String)
  - getParameterValues(String)

  http://host:port/servlet/UrlParameterMsg?style=ja

HTML FORMの取り扱い

- HttpServlet: httpに依存したServlet
- メソッド
  - 最初に呼ばれるメソッド
    doGet(HttpServletRequest request,
    HttpServletResponse response)
  - POSTで呼ばれるメソッドdoPost(HttpServletRequest request,
    HttpServletResponse response)

Java Server Pages (JSP)

- HTMLの中にjavaのコードを埋め込んで、動的にページを作りだす仕組み
- pageに埋め込まれているjavaのコードから、プログラムを生成し、動的にコンパイルし、servletとして実行することによって実現されている。
  - <% %>に囲まれた間をJavaのプログラムとして実行し、その出力をＨＴＭＬの中に埋め込む。
  - <script> </script>で必要なメソッドを定義
- JavaBeans技術を使うことができる
- 類似の技術にPHPがある

ＪＳＰの例

```
<script runat="sever">
private String getThisTime(String sytle){ ...}
</script>
<html><head>
<title> ....</title>
...
<%
 out.println("...");
 String style= request.getParamter("style");
 out.println(getThisTime(style));
 out.println("...");
%>
...
</html>
```


J2EEとEnterprise Java Beans

- J2EE: Java 2 Platform, Enterprise Edition
- MI/IIOP : 従来のRMIに加えて、RPCのプロトコルとして、CORBA(Common Object Request Broker)の標準プロトコルであるIIOP(Internet Inter-ORB Protocol)を使えるようにしたもの。これにより、Javaだけでなく、C／C++などから呼び出したり、その逆も可能になる。
- JBDC : データベースにアクセスするための標準のJava APIを提供するもの。SQL文(Structured Query Languageの略、リレーショナルデータベースの標準的な言語)を簡単に各種のデータベースに送信できる。メジャーなデータベースである、Sybase, Oracle, IBM DB2などに同じAPIでアクセスすることができる。JBDCのAPIとデータベースを結びつける部分をドライバーといい、ドライバーを使ってデータベースと接続することをコネクションという。

- JNDI(Java Naming and Directory Interface): ネットワーク上にあるいろいろなネーミングとディレクトリサービスをまとめたもの。RMIのregistryもこの中の１つになっている。そのほかに、標準的なディレクトリサービスであるLDAP(Lightweight Directory Access Protocol)やCORBAのNamingサービスであるCOS（Common Object Service）もSPI(Service Provider Interface)として統合されている。
- Servlet: httpサーバ側で、Javaプログラムを動かす仕組み。
- JavaServer Page(JSP): HTML中にJavaのプログラムを記述し、動的なWebページを動かすための仕組み。
- Enterprise JavaBeans: ネットワーク上で使えるようにしたBeans.
- Java Transaction: JavaのデータベースなどのtransactionをサポートするAPI
- XMLライブラリ：JavaでXML文書を扱うためのパッケージ。直接は関係ないが、J2EEの中のいろいろなところで使われている。

RMI/IIOP

- COBRAのプロトコルであるIIOPをつかったＲＭＩのimplementation
- 今ではこちらのほうが主流
- CORBAのオブジェクトでも呼び出せる
- プログラミングでの違い
  - UnicastRemoteObjectの代わりに、PortableRemoteObjectを使う。
  - RMI registryでなく、JNDIregistryをつかう。
  - rmicを使うときに、--iiopのオプションを使う

ＳＯＡＰ

- SOAP: Simple Object Access Protocol
- ＸＭＬを使ったRPCのためのプロトコル
  - ＨＴＴＰで、カプセル化できる。
  - 送信側は,ＵＲＬに対するＰＯＳＴメッセージ
  - 受信側は、HTTPからのメッセージ
  - ＵＲＬを指定して、ＲＰＣを行う
- クライアントのライブラリ
  - org.apache.soap.*

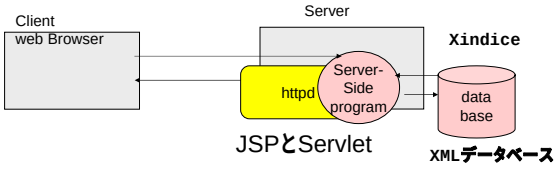
SOAPを使ったＲＰＣの例

```
public class GetPassword {
   public static main(String args[]){
       String urlstring =
                "http://localhost/soap/servlet/rpcrouter";
       Call c = new Call();
       c.setTargetObjectURI("urn:userinfoserice");
       c.setMethodName("getPassword");
       c.setEncodingStyleURI(...);
       Vector v = new Vector();
       v.addElement(...);
       c.setParams(v);
    …
       r = c.invoke(new URL(urlString),"");
       Parameter result = r.getReturnValue();
    …
  }
}
```


- 今日はここまで。

web プログラミングの実際

- Xindiceの付録についていたAddressbookを例に
  - Xindice: native XML データベース
  - servletからアクセス
  - User interfaceは、JSPで書いてある。



tomcatとは

- Tomcat は、Java Servlet 2.2 および JavaServer Page 1.1 技術の正式なリファレンス実装
- Tomcatは、Apacheライセンスに基づいたオープンで誰でも参加できる環境下で、多くの優秀な開発者が協力して開発
  - http://www.jajakarta.org/
- All javaの実装
- apacheにも組み込める（ようだ）

tomcatでのアプリケーションのdeploy

- アプリケーションは、$(TOMCAT_HOME)/webappsにおく。
  - $(TOMCAT_HOME)/webapps/xyzはhttp:<address>/xyzで参照される
- webapps/Addressbookには
  - 直下には、html, jspなど
    - 最初はindex.{html,jsp}が起動される
  - WEB-INFの下がservletのプログラム
    - web.xml servletの環境設定
    - classes servletのプログラムをおく
    - lib servletを動かすためのライブラリ

index.jsp、header.jsp

- はじめに起動されるページ
- <%@ include …%>で、header.jspとfooter.jspをinclude
  - これは、枠を表示
- 実際は、header.jspにあるメニューから飛ぶ
  - メニュー“List Contact”は、

```
<A  HREF=
“/Addressbook/servlet/Task?action=listcontacts”>
  List Contacts</A>
```

- classesの中にあるservletにアクセスするには /Addressbook/servlet/<クラス名>?パラメターでアクセスする

Task.java

- servletのメインプログラム
  - 通常、このＵＲＩが参照された場合にはGETメッセージになるため、doGetは呼ばれる
  - doPostは、HTMLのフォームから呼ばれた場合
    - addContact.jsp
  - doPostとdoGetは同じにして、パラメータとしてわたされるactionで区別している
    - request.getParameter(“action”);
- 最初に、beanを作る
  - このアプリケーションでは、groupとpersonというbeanをつかっている（header.jsp)
  - JSPとservletで共有できる“変数”

task.java

- “action”パラメータで、分岐して
  - gotoPage(“JSPのページ”,request, response);
  - 例えば、addformの場合、/addContact.jsp
- エラーの場合は、error.jsp ?error=… で、response.sendRedirect する。

データベースのアクセス

```
ackage org.apache.xindice.examples;

import org.xmldb.api.base.*;
import org.xmldb.api.modules.*;
import org.xmldb.api.*;

public class Example1 {
   public static void main(String[] args) throws Exception
      Collection col = null;
      try {
         String driver =
            "org.apache.xindice.client.xmldb.DatabaseImpl";
         Class c = Class.forName(driver);
         Database database = (Database) c.newInstance();
         DatabaseManager.registerDatabase(database);
         col =
 DatabaseManager.getCollection("xmldb:xindice:///db/address
        String xpath = "//person[fname='John']";
```


データベースのアクセス

```
      col = DatabaseManager.getCollection(
               "xmldb:xindice:///db/addressbook");
      String xpath = "//person[fname='John']";
      XPathQueryService service
  (XPathQueryService) col.getService("XPathQueryService",
       ResourceSet resultSet = service.query(xpath);
       ResourceIterator results = resultSet.getIterator()
       while (results.hasMoreResources()) {
          Resource res = results.nextResource();
          System.out.println((String) res.getContent());
       }
    }
    catch (XMLDBException e) {
       System.err.println("XML:DB Exception occured " + e
    }
    finally {
       if (col != null) {
```

データベースについて

- データベース　Database
- コレクション　Collection
- サービス　XPathQueryService
- リソースセット　ResourceSet
- リソースIterator

addContact.jsp

- 名前の入力のためのFORM
  - POSTで、送る
  - そのときに、ACTION=ADDCONTACT
  - それを/Addressbook/servlet/Taskに送る。

addContact.java

- HttpServletRequest, HttpServletResponseを引数として、requestからgetParameterで、入力を作る
- Actionをextendして作られている

Action.java

- 他のactionのベースクラス
  - AddContact.java
  - DeleteContact.java
  - …
- getCollection
  - 初期化時に作られるDBconnectionから、collectionを返す
  - DBConnectionは、もしもDBとつながっていなかったらつなげる

Group, Person

- Person.java Servlet上の一人分のデータ
- Group.java Servlet上のPersonデータの列

ListContact.java、listContact.jsp

- /personでXpathQueryをして、groupを作る。
- listContact.jsp
  - ListContact.javaが呼ばれてから、表示
  - groupというbeansを共有しながら、表示している。

WebService

- “ Web ”(browser)を使ったサービスではない！
- WSDL (Web Service Description Language)を媒介として、サービス（RPC)を記述する枠組み
  - 多くの場合、SOAPが使われる
  - 多くの場合、ＸＭＬが使われる
  - 多くの場合、80 (ＨＴＴＰ)ポートが使われる
  - が、他のバインドも可
    - だけど、あまりない。

Apache Axis

- Apache AxisとはJavaで書かれたSOAP実装の一つです (SOAPというのはWebサービスで使われる通信プロトコルです。XMLで書かれており.NETとJavaでさえ通信可能です)。
  - tomcat servlet で実装
- 簡単な使い方
  - Java Web Service (.jws)
- でもちゃんとした使い方が必要…
  - セッション
  - WSRF (web service resource Framework) by GGF

Java web service

- 作成した.javaをウェブサービスとしてpublishします。
- 作成したHello.javaの拡張子を.jwsにして、Axis Webアプリケーションのルートにコピーします。
- ブラウザを開いて以下のURLへアクセスします。
- http://localhost:8080/axis/Hello.jws?wsdl

```
public class Hello{
  public String sayHello(){
    System.out.println("call sayHello");
    return "hello!";
  }
}
```


WSのクライアントのコード

```
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import javax.xml.namespace.QName;

public class HelloClient {
   public static void main(String [] args) throws Exception{

        String endpoint = "http://localhost:8080/axis/Hello.jws";

        Service  service = new Service();
        Call     call    = (Call) service.createCall();

        call.setTargetEndpointAddress( new java.net.URL(endpoint) );
        call.setOperationName(
         new QName("http://localhost:8080/", "sayHello"));
        String ret = (String) call.invoke( new Object[0] );

        System.out.println(ret);
   }
```


ちゃんとしたDeploy

- コンパイルされたEcho.classファイルをAxis Webアプリケーションが認識できる場所に置きます。例えば、以下にEcho.classをコピーします。
  - webapps/axis/WEB-INF/classes/
- WSDDファイルを作成します。以下の内容のファイルをdeploy.wsddとして保存します。
- デプロイを実行します。java.org.apache.axis.client.AdminClientを使います。引数は今作ったwsddファイルのパスです。

クライアントの作成

- WSDLから自動生成
  - org.apache.axis.wsdl.WSDL2Java

```
import localhost.Echo;//Echoとlocalhost.Echoが衝突する場合の対
import localhost.*;

public class EchoClient{

  public static void main(String[] args) throws Exception{
    EchoService locator = new EchoServiceLocator();
    Echo echo = locator.getecho();
    String s = echo.sayEcho("hoge");
    System.out.println(s);
  }
}
```

つぎは

- ＷＳの詳細？