プログラミング環境特論

Javaによるwebプログラミング入門

佐藤

筑波大学

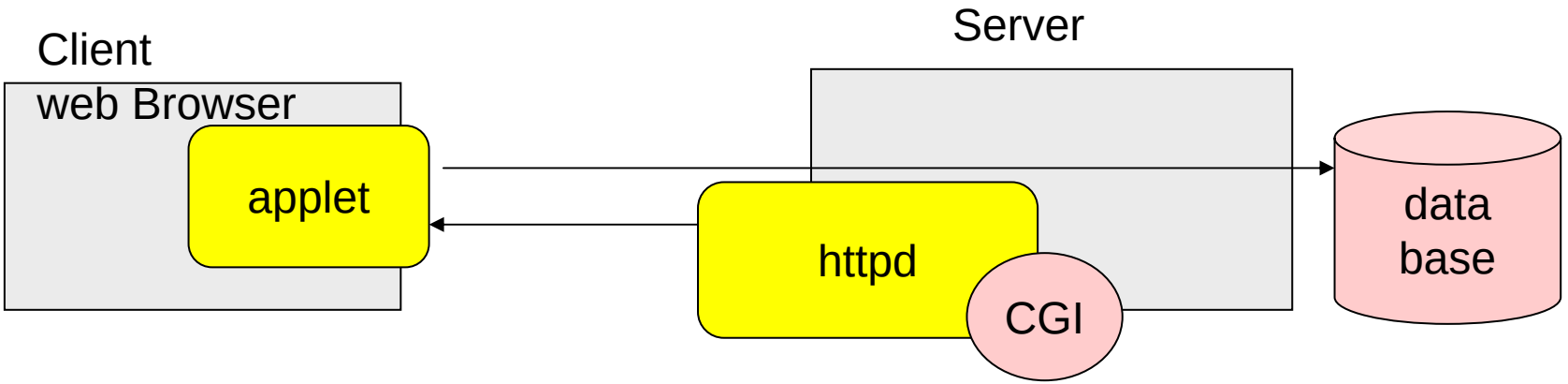
JavaによるWebプログラミング

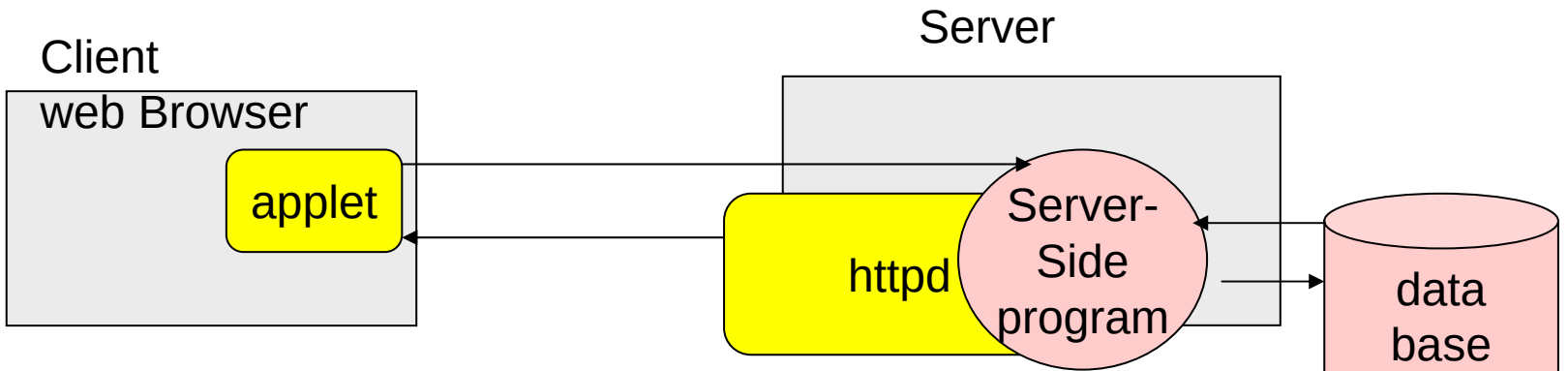
- Applet - クライアントのJVM上でJavaアプリを実行
- Servlet - サーバー側でJVMでJavaアプリを実行
  - ３層モデル
- JSP (Java Server Pages)
- SOAP (Simple Object Access Protocol)
- WebService
- XML (Extended Markup Language)
- J2EE　(Java2 Enterprise Edition)

- JavaScript ⇒ Ajax

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





プログラミング環境特論

- 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
- JavaScript (Client側で動作）との違いに注意

JSP (Java Sever Pages)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。

J2EEとEnterprise Java Beans

- 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

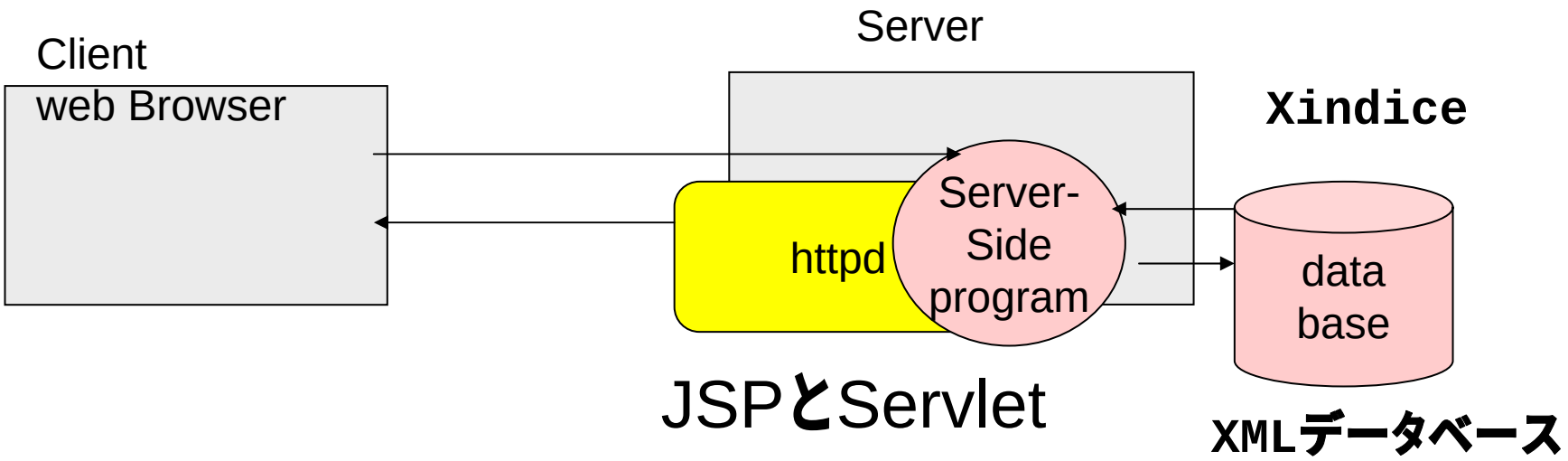
- COBRAのプロトコルであるIIOPをつかったRMIのimplementation
- 今ではこちらのほうが主流
- CORBAのオブジェクトでも呼び出せる
- プログラミングでの違い
  - UnicastRemoteObjectの代わりに、PortableRemoteObjectを使う。
  - RMI registeryでなく、JNDIregistryをつかう。
  - rmicを使うときに、--iiopのオプションを使う

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

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で書く



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する仕組み
- 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
- ブラウザを開いて以下のURLへアクセスとWSDLが見える
  - 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);
   }
}
```

WSDLを用いたWebServiceの開発

- 通常は、WSDL(WebSevice Description Language)でインタフェースを記述して、これからWSDL2Javaというプログラムで、JavaのStab コード（client/server)を生成する。
  - Client側はそのまま
  - Serverは生成されたコードをベースに中身を生めていく。
  - サーバ側のコードは、Deploy(サーバ側にコードを設置すること）という設定をして、使えるようになる。
  - WSのURI?wsdl で、WSDLによるサービスの記述が取得できる。

WSDL (WebSevice Description Language)

- 発見したWebサービスを利用しようとした場合、Webサービスを利用するためのインターフェイス仕様を知る必要ある。この仕様をコンピュータが理解できる形式で記述するためのXML 形式の言語がWSDL（Web Service Description Language）
  - タイプ (types） --- 交換されるメッセージを記述するために使用するデータ型の定義を記述します。
  - メッセージ (message） ---伝送されるデータの抽象定義を表します。メッセージは論理的なパートで構成され、そのそれぞれが何らかの型システムの定義に関連付けられます。
  - オペレーション（operation）--- 操作の抽象的な定義。それぞれの操作は、入力メッセージと出力メッセージを参照します。
  - ポート タイプ (portType） --- 抽象操作のセット。
  - バインディング (binding） --- 特定のportTypeによって定義された操作とメッセージの具体的なプロトコルとデータ形式を指定します。
  - ポート (port） --- 単一の通信端点のアドレスを定義します。通信端点のアドレスとは、実際にWeb サービスを提供するサーバのURLなどを指します。
  - サービス (service）--- 関連する通信端点を集約するために使用されます。

例（hello.jws） 1/2

```
 <?xml version="1.0" encoding="UTF-8" ?>
- <wsdl:definitions targetNamespace="http://localhost:8080/axis/Hello.jw
xmlns:impl="http://localhost:8080/axis/Hello.jws" xmlns:intf="http://loc
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:wsdl="ht
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http:/

- <wsdl:message name="sayHelloResponse">
  <wsdl:part name="sayHelloReturn" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="sayHelloRequest" />
- <wsdl:portType name="Hello">
- <wsdl:operation name="sayHello">
  <wsdl:input message="intf:sayHelloRequest" name="sayHelloRequest" />
  <wsdl:output message="intf:sayHelloResponse" name="sayHelloResponse" /
  </wsdl:operation>
  </wsdl:portType>
- <wsdl:binding name="HelloSoapBinding" type="intf:Hello">
  <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/so
- <wsdl:operation name="sayHello">
  <wsdlsoap:operation soapAction="" />
- <wsdl:input name="sayHelloRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding
```

例（hello.jws）2/2

```
- <wsdl:operation name="sayHello">
  <wsdlsoap:operation soapAction="" />
- <wsdl:input name="sayHelloRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding
  </wsdl:input>
- <wsdl:output name="sayHelloResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding
  </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
- <wsdl:service name="HelloService">
- <wsdl:port binding="intf:HelloSoapBinding" name="Hello">
  <wsdlsoap:address location="http://192.168.153.127:8080/axis/Hello.jws
  </wsdl:port>
  </wsdl:service>
  </wsdl:definitions>
```

クライアントの作成

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

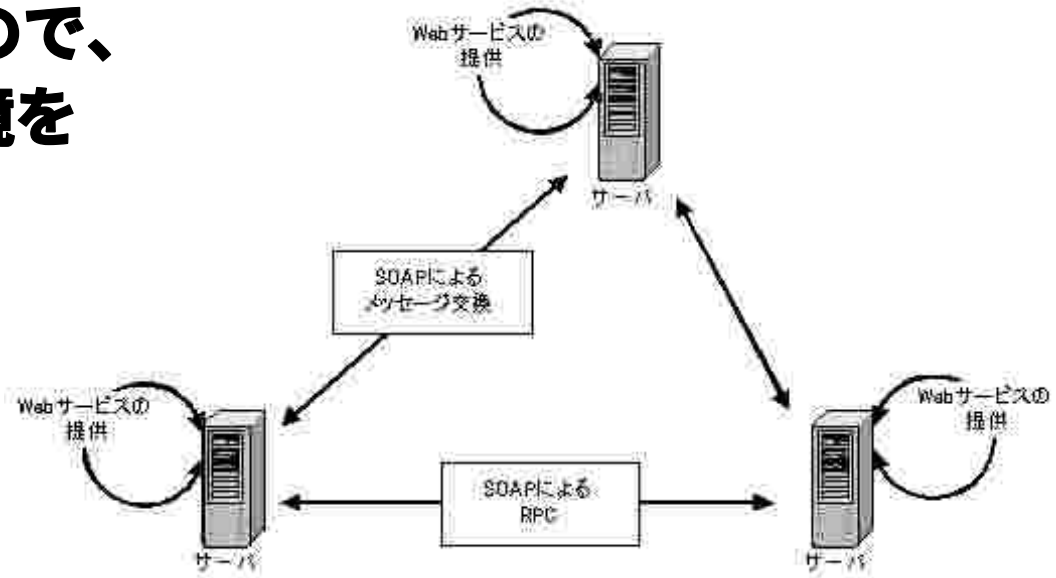
```
import localhost.Echo;//Echoとlocalhost.Echoが衝突する場合の対
import localhost.*;

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);
  }
}
```

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。

SOAPとは

- SOAP: Simple Object Access Protocol
  - ネットワーク上のアプリケーション間（オブジェクト間）の情報を交換し合うための単純で軽量なプロトコルの仕様
- COM（Component Object Model）やCORBA（Common Object Request Broker Architecture） と比較してOpen
- XMLベースでの通信なので、既存のXMLツールや環境を応用することができる



SOAPとは

- 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
- WebServiceでも、下のレイヤーで使われている
- 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();
     …
  }
}
```

SOAPのメッセージ

- ３つの部分
  - エンベロープ構成要素
    - ├ メッセージ構成
    - └ 処理仕様
      - メッセージがどのような構成から成り立っているのか
      - 誰がどの構成要素をどのように処理するべきなのか
      - それらの構成要素や処理は必須か選択可能であるのか、
  - エンコーディング規則
    - └ データのシリアライズメカニズム
  - RPC表現規則
    - └ 要求と応答の規則
      - SOAPを用いてRPCを実現するための要求（コール）と応答（レスポンス）の表現規則を定義
      - SOAP HTTPバインディングではHTTPリクエストとHTTPレスポンスとして表現されることになります。

SOAPメッセージの構成

- **HTTPバインディングされたSOAPメッセージの例**


```
POST /StockQuote HTTP/1.1                                          ①ヘッダ
Host: baseball.ufj.co.jp
Content-Type: text/xml; charset="utf-8"
Content-Length: ******
SOAPAction: "http://baseball.azb.co.jp/Apache/DataStor/getResult"

<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
 <SOAP-ENV:Header>
  <t:Transaction xmlns:t="http://baseball.azb.co.jp/Apache/" SOAP-ENV:mustUnderstand="1">
       5
  </t:Transaction>                                                 ③SOAPヘッダ
 </SOAP-ENV:Header>
 <SOAP-ENV:Body>
   <m:getPitchingResult xmlns:m="http://baseball.azb.co.jp/Apache/DataStore/">
     <m:name>Akinobu Yoshida</m:name>
     <m:No>00</m:No>
   </m:getPitchingResult>                                          ④SOAP本体
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>                                               ②SOAPエンベロープ
```

envelopの例

```
<SOAP-ENV:Enve
   "http://sch
SOAP-ENV:encod                                                    ng/
   <SOAP-ENV:Header>
      <t:Transaction xmlns:t="http://baseball.azb.co.jp/Apache/"
         SOAP-ENV:mustUnders
         5
      </t:Transaction>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
      <m:getPitchingResult
            "http://baseba
         <m:name>Akinobu Yos
         <m:No>00</m:No>
      </m:getPitchingResult>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP本体中のメッセージを誰が（どのサーバーが）どのように処理を行うかなどのSOAPメッセージを処理するアプリケーションが解釈すべき情報を記述
つまりSOAPメッセージの宛先

メッセージの受信者が処理を行う情報が記述される。つまりSOAPメッセージの本文
SOAP本体はメッセージの受信者側が理解できるXML形式で記述しなければなりません。
RPCで使用するならば、メソッドやメソッドに必要なパラメータなどの要素を記述します。またレスポンス中では処理結果を記述するのもSOAP本体になります。SOAPの処理が失敗した際のエラー情報もSOAP本体中に記述されることになっています。

RPCの場合

- メソッド呼び出しの場合は、Bodyに以下が入る

```
<メソッド名>
    <パラメータ名1>パラメータ値</パラメータ名1>
    <パラメータ名2>パラメータ値</パラメータ名2>
</メソッド名>
```

- メソッドレスポンス

```
<メソッド名Response>
    <パラメータ名1>パラメータ値</パラメータ名1>
    <パラメータ名2>パラメータ値</パラメータ名2>
</メソッド名Response>
```

実際どのようなメッセージがながれるのか

- ポートモニターでみることができる

```
public class HelloService {
    public String getMessage_ja( String name ) {
        String echo = "名無し";
        if( name != null ) echo = name ;
        return "こんにちは " + echo + "さん ";
    }
}
```

◆ クライアントのコード

```
import java.util.*;
import java.net.*;
public class TestClient {
  public static void main( String [] args ) throws Exception {
    localhost.HelloWORDLocator locator =
      new localhost.HelloWORDLocator();
    URL url =
      new URL("http://localhost:4040/WS-I/services/HelloWORD");
    if( args.length > 1 ) url = new URL( args[1] );
    localhost.HelloService service = locator.getHelloWORD( url );
    String requestMessage = "岩本";
    if( args.length > 0 ) requestMessage = args[0];
    String resultMessage =  service.getMessage_Ja( requestMessage
    System.out.println( resultMessage ) ;
  }
}
```

プログラミング環境特論

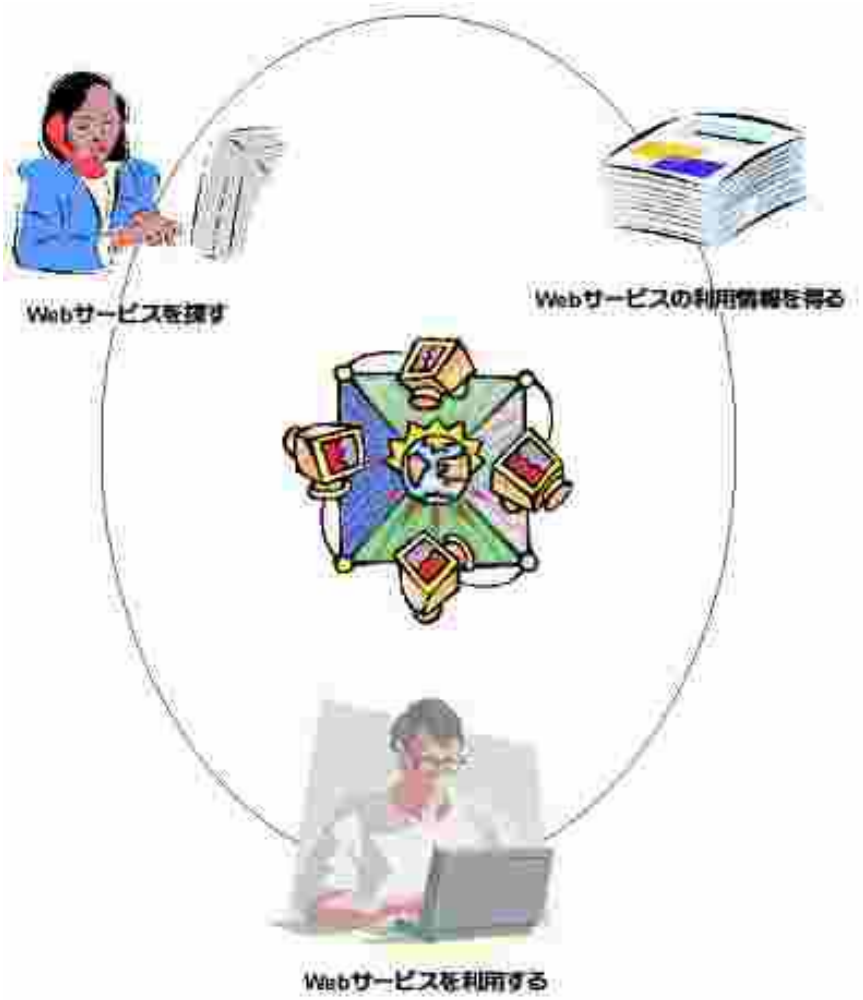
```
[HTTP Headers:]

POST /WS-I/services/HelloWORD HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: Axis/1.0
Host: localhost:4040
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: ""
Content-Length: 479

[Message Content:]
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <ns1:getMessage_ja soapenv:encodingStyle=
"http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://localhost:8080/WS-I/services/HelloWORD">
   <in0 xsi:type="xsd:string">岩本</in0>
  </ns1:getMessage_ja>
 </soapenv:Body>
</soapenv:Envelope>
```

プログラミング環境特論

```
[HTTP Headers:]
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Date: Thu, 04 Sep 2003 12:43:22 GMT
Server: Apache Coyote/1.0
Connection: close

[Message Content:]
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <ns1:getMessage_jaResponse soapenv:encodingStyle=
"http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://localhost:8080/WS-I/services/HelloWORD">
   <getMessage_jaReturn xsi:type="xsd:string">こんにちは 岩本さん
  </getMessage_jaReturn>
  </ns1:getMessage_jaResponse>
 </soapenv:Body>
</soapenv:Envelope>
```

Webサービスを活用するための３ステップ

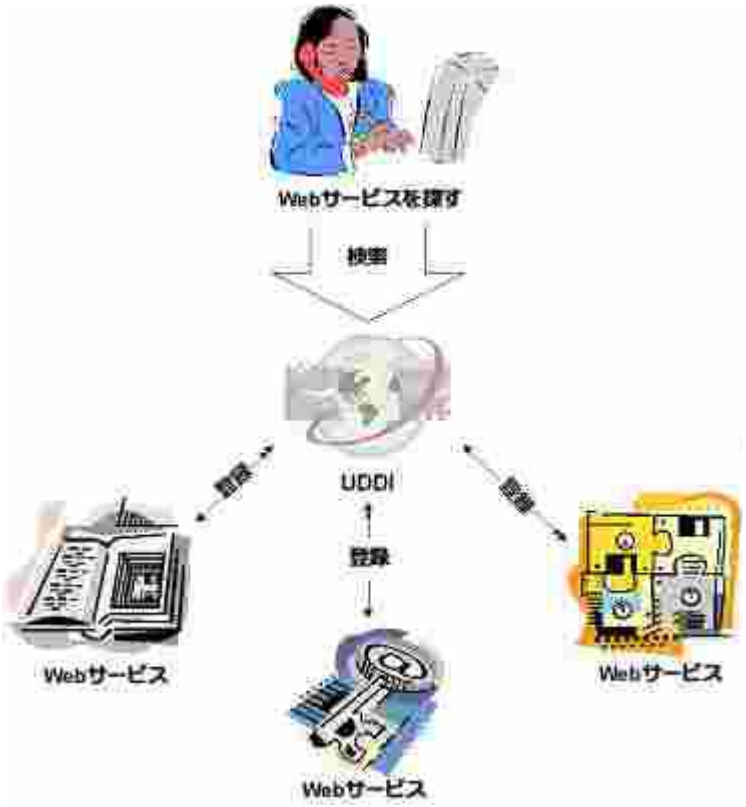
- １．Webサービスを探すこと。
  - ＵＤＤＩ
- ２．Webサービスの利用方法を知ること
  - ＷＳＤＬ
- ３．Webサービスを実際に利用すること



UDDI

- UDDI（Universal Description, Discovery and Integration）
- http://www.uddi.org/

- ホワイトページ
  サービス提供者の企業名、住所、電話番号など、企業名から検索するためのレジストリ
- イエローページ
  サービス提供者のサービスの分類コードなど、業種・サービスの種類から検索するためのレジストリ
- グリーンページ
  Web サービスを利用するための技術情報が登録されたレジストリ

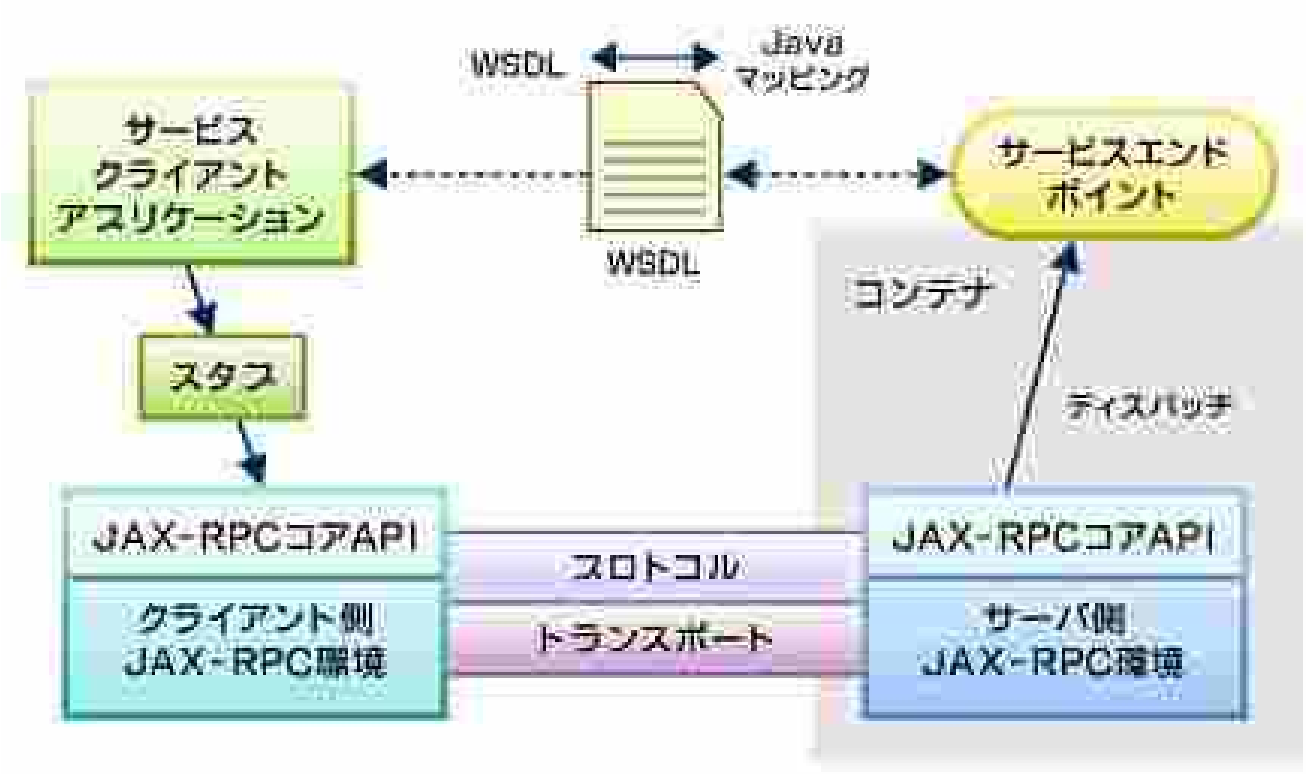


JAX-RPC

- J2EEの一つ
- JAX-RPCは、XMLを使ったRPC（Remote Procedure Call：遠隔操作呼び出し）を実装するためのJavaの標準仕様で、JCPにてJSR-101として策定されたものです。
  - Javaプラットフォームでの XMLベースのRPCをサポートするAPIを定義する
  - 基本レベルのプロトコルバインディングを定義する（ほかのプロトコルバインディングを制限する意味ではない）
  - JAX-RPCベースのサービスを定義（提供）、あるいは利用するJavaアプリケーションを開発するための基本APIを定義する
  - ヘテロジーニアス（混合）な環境での相互接続性をサポートする
  - JAX-RPC実装に対する、適合性と相互接続性の検証のための要件を定義する
  - JAX-RPC APIとメカニズムを拡張可能かつモジュラーに保つ

JAX-RPC

- JAX-RPCは、RPC仕様であるので、いわゆるWebサービス・アーキテクチャ、すなわち、SOAP、WSDL、UDDIによるデルタモデルを実現するものではありません。利用する技術はSOAPとWSDLであり、発見メカニズムにはJNDIを利用することができます



JAX-RPC

- JAX-RPCでは、Webサービスの機能を提供するプログラムコードのことを「サービス・エンドポイント」と呼びます。エンドポイントの機能は、interface句で定義されることになっており、これを「サービス・エンドポイント・インターフェイス（SEI）」といいます。

- クライアント側のＡＰＩ
  - javax.xml.rpc.Stubインターフェイス
  - javax.xml.rpc.Callインターフェイス（動的呼び出しモデル）
  - javax.xml.rpc.Serviceインターフェイス
  - javax.xml.rpc.ServiceFactoryクラス
  - javax.xml.rpc.JAXRPCExceptionクラス

- スタブをつくることもできるし、動的に呼び出すこともできる

WSの応用例：Google API

- http://www.google.com/apis/index.html
- 2004?「米Googleは11日、「Google Web API」を公開した。これはSOAP1.1およびWSDLに基づいたインターフェイスで、Googleがこれまでに収集した20億ものWebページのデータを自分の好きなようにプログラミングして利用することができる。この試みについては多くの開発者が「新たな革命」「Netscapeのリリース以来の出来事」と賞賛している。」

- Amazon.comとかもあるようだ （あった）

プログラミング環境特論

このGoogleAPIがインターネット全体に及ぼす影響は計り知れないものだ。20億ページのデータを自在にプログラミングできることから、どのようなアイデアが生まれてくるのか現在のところ想像もつかないが、Googleではそのためにいくつかのアイディアを FAQ のなかで示している。

例えば、「定期的に特定のキーワードについて新しいページが登録されていないかを走査、通知してくれるプログラム」、「Webの中にある情報の量に応じて市場にどのような需要があるのかを分析する市場調査プログラム」、「コマンドラインや携帯電話などHTML以外のインターフェイスを通してGoogleを利用するプログラム」、「Webにある情報を使った“おもしろい”ゲーム」などを挙げている。
Googleでは、このGoogleAPIの議論のためにニュースグループを設置。開発者たちがアイディアや意見を交換できるようにしており、自分が開発したプログラムを公開することも歓迎するとしている。こうしたアイディアは後々Googleの新たなビジネス戦略にも用いられるものと思われる。

GoogleAPIを利用するためには、開発者キットをダウンロードしなければならない。これには開発者に必要となるJavaライブラリ、WSDLファイル、その他サンプルコードなどが含まれており700KB弱のファイルである。ダウンロードの後、新たに設けられた「Google Account」に登録することでライセンスキーを取得、それによってGoogleAPIを試すことができる。GoogleAPIがサポートしているプログラミング言語は、Java、Perl、Ruby、Microsoft VisualStudio.NETだが、GoogleではWebサービスをサポートしているプログラミング言語ならうまく動作するのではないかと示唆している。

GoogleAPIは、現在ベータテスト中であるため無料で利用できる。その代わり1日に1,000クエリーという制限が設けられている。これはGoogleのリソースを必要以上に消費しないためであると説明されている。またベータテスト中ということもあり、サービスが突然メンテナンスのために停止されたり、APIに将来的に変更があることも考えられるため、利用はあくまで実験的な用途に限られそうだ。

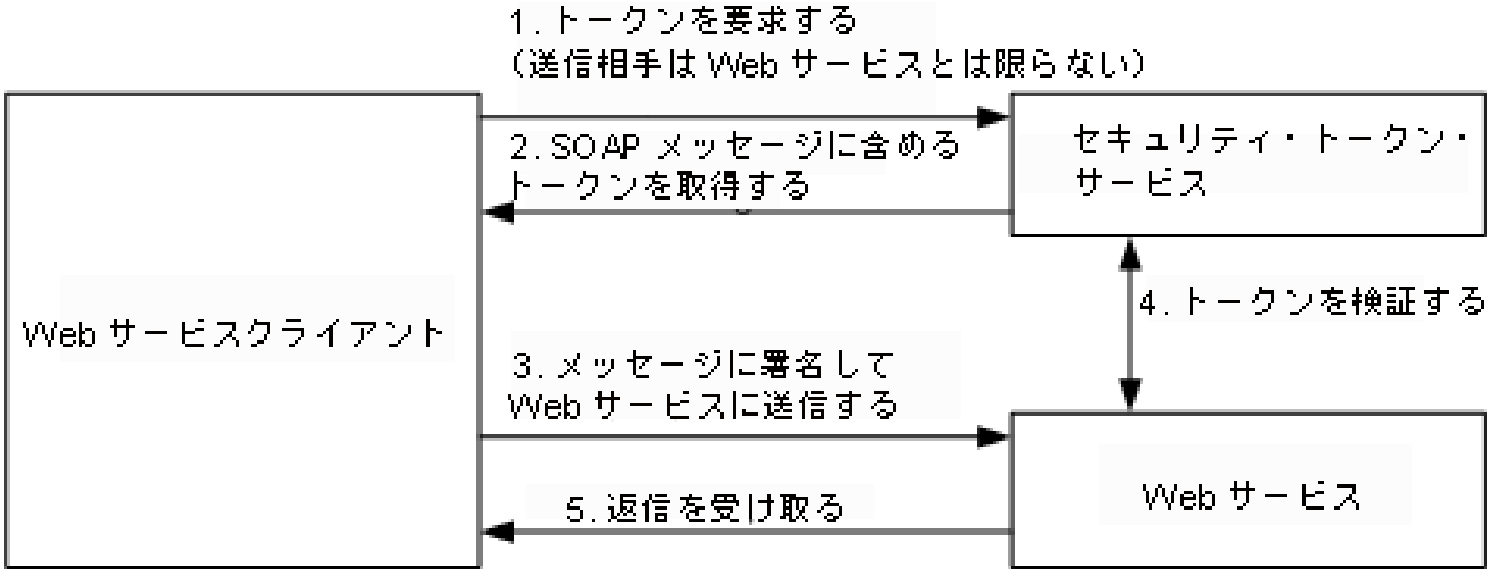
WS-Security

- Webサービスのセキュリティは？
  - SOAP の目的は HTTP を利用して 2 つのエンドポイントの間でメッセージを送受信するためのもの
  - HTTPSを使えばいい？
  - 不十分、なぜ
    - そのメッセージをフォワードして処理する場合とか
    - より複雑な経路を通したり、 HTTP ではない転送プロトコルを利用したりしてメッセージを送受信する場合

- WS-Security は、セキュリティ関連のデータを運ぶための SOAP の Header 要素の内容を定義
  - 既存の仕様に追加して、WS-Security は、これらのメカニズムを SOAP メッセージに、トランスポートに依存しない形で組み込むためのフレームワークを提供します

WS-Security

- 主要なセキュリティ要件
  - 認証
  - 電子署名
  - 暗号
    - 「誰に対してアクセスを許可しようとしているのか。」
    - 「到達する前にメッセージが改ざんされていないか。」
    - 「メッセージは予定通りの相手から送信されたものか。」
    - 「特定の相手にしか公開したくない情報を隠す方法は。」



```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
    <soap:Header
        xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/07/secext"
        xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">
        <wsu:Timestamp>
            <wsu:Created
                wsu:Id="Id-3beeb885-16a4-4b65-b14c-0cfe6ad26800"
                >2002-08-22T00:26:15Z</wsu:Created>
            <wsu:Expires
                wsu:Id="Id-10c46143-cb53-4a8e-9e83-ef374e40aa54"
                >2002-08-22T00:31:15Z</wsu:Expires>
        </wsu:Timestamp>
        <wsse:Security soap:mustUnderstand="1" >
            <xenc:ReferenceList>
                <xenc:DataReference
        URI="#EncryptedContent-f6f50b24-3458-41d3-aac4-390f476f2e51" />
            </xenc:ReferenceList>
            <xenc:ReferenceList>
                <xenc:DataReference
        URI="#EncryptedContent-666b184a-a388-46cc-a9e3-06583b9d43b6" />
            </xenc:ReferenceList>
        </wsse:Security>
    </soap:Header>
    <soap:Body>
        <xenc:EncryptedData
```

```
<soap:Body>
        <xenc:EncryptedData
            Id="EncryptedContent-f6f50b24-3458-41d3-aac4-390f476f2e51"
            Type="http://www.w3.org/2001/04/xmlenc#Content">
            <xenc:EncryptionMethod Algorithm=
                "http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
            <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
                <KeyName>Symmetric Key</KeyName>
            </KeyInfo>
            <xenc:CipherData>
                <xenc:CipherValue
                >InmSSXQcBV5UiT...  Y7RVZQqnPpZYMg==</xenc:CipherValue>
            </xenc:CipherData>
        </xenc:EncryptedData>
    </soap:Body>
</soap:Envelope>
```

最後に

- Servletは、J2EEの重要なコンポーネントの１つ
  - Javaを中心にいろいろな技術が統合されている環境
- Web Serviceは、分散プログラミング環境ではとても重要な技術
- ＷＳはJavaだけでなく、C++などもある
  - gSOAP
- グリッドコンピューティングとの関連