Java programming for web

M. Sato

Web programming by Java

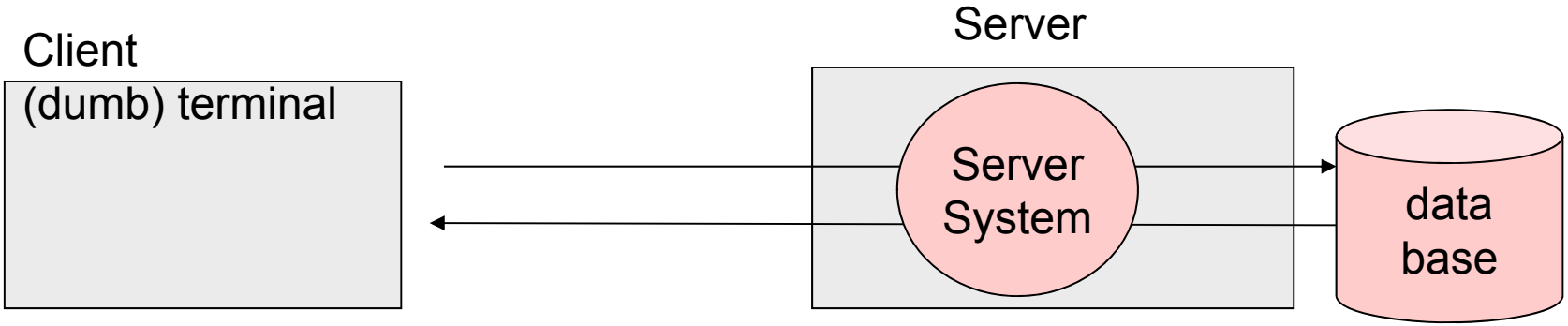
- **Applet - executing Java application on client-side Java VM**
- **Servlet - executing Java application on sever-side Java VM**
  - **3 layers model**
- **JSP (Java Server Pages)**

- **WebService**
- **SOAP (Simple Object Access Protocol)**

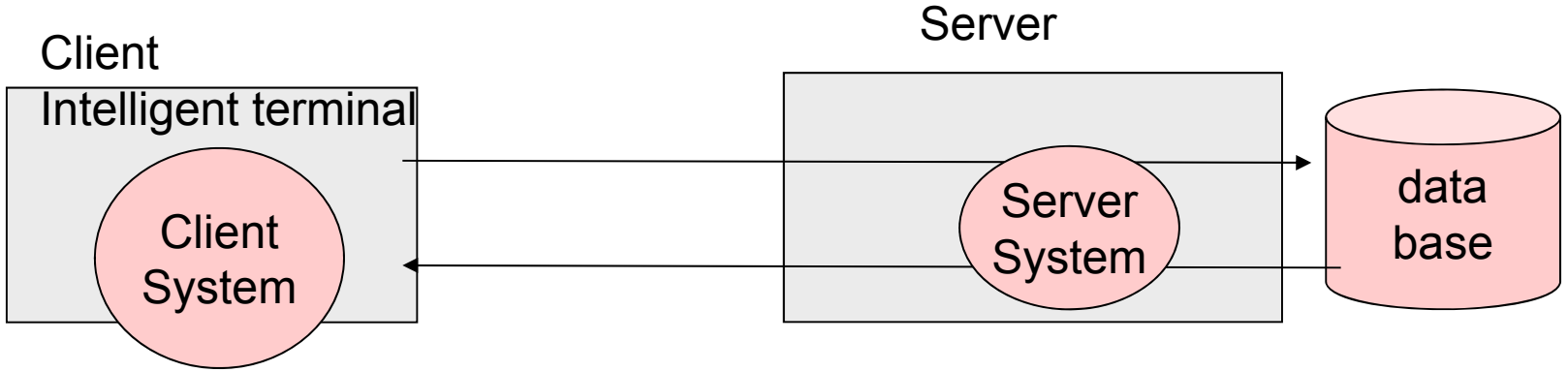
- **JavaScript ⇒ Ajax**

From two layer model to 3 layer model

- **Centralized System (2 layers)**

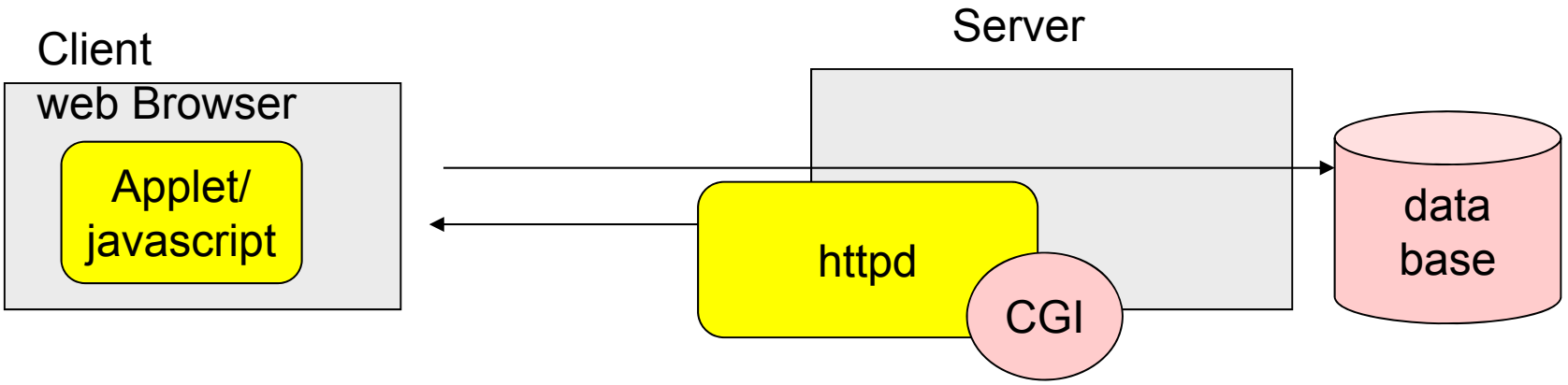


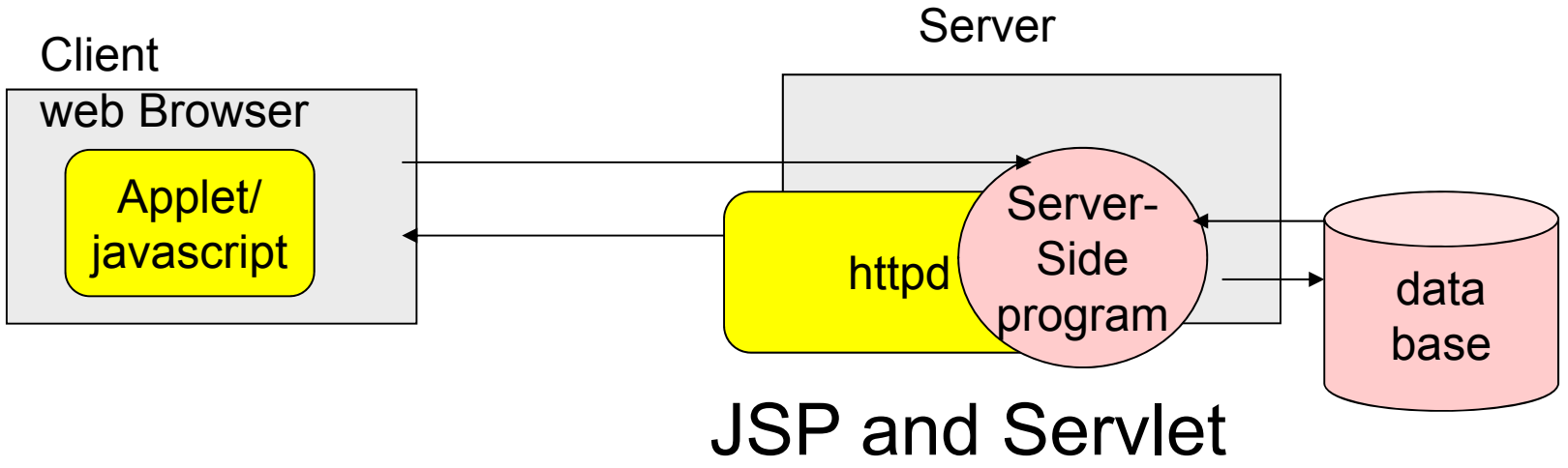
- **Full distributed System (2 layers)**



From two layer model to 3 layer model

- **Server-side program (3 layers)**





JSP and Servlet

Advantages of 3 layer model

- In distributed system (2 layer model), each client system (many!) may be updated if the system update need any modification on client systems.
  - In 3 layer system, only client-side program (only one!) can be modified.

- In distributed system (2 layer model), the intelligent client system have a privileged right to access the main frame database. It may cause a serious security problem if the distributed system run on internet of public use.
  - In 3 layers system, only traffic between server and client is a html data for GUI. The data can be enclosed in server-sided.

Servlet

- A servlet is a Java programming language class used to extend the capabilities of servers that host applications access via a request-response programming model.
- Java program executed in web server-side JavaVM.
- Servlet is usually a Java program which generates HTML documents for the web browser. It may access the back-end database etc …
- How to invoke
  - Specify a servlet name which is registered in web server configuration file.
  - Specify a Servelt by URL
  - By using tag <servlet> …</servlet>, specify a servlet to invoke in HTML file.
- Advantage:
  - Smaller overhead to invoke different programs as in CGI
  - Productive because of Java programming environment
- Prototype implementation： Apache Tomcat

Servlet program using URL parameters

```
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) {
 ...
 }
}
```

Servlet program using URL parameters

- GenericServlet: used in case dependent from http
- Obtain the parameters of URL as in GCI

```
http://host:port/servlet/servlet-name?name=value
```

- Method: javax.servlet.ServletRequest
  - getParameterString(String)
  - getParameterValues(String)

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

Using HTML FORM

- **HttpServlet: Servlet dependent on http**

- **Method:**
  - **The method called at first time access**

```
doGet(HttpServletRequest request,
HttpServletResponse response)
```

  - **Method called at POST request**

```
doPost(HttpServletRequest request,
  HttpServletResponse response)
```

Tomcat

- Tomcat is a formal reference implementation of Java Servlet 2.2 and JavaServer Page 1.1 by Apache project.
- Tomcat has been developed as an open free software under Apache license by many contributor
  - http://www.jajakarta.org/
- All java implementation
- Can be integerated into apache httpd server.

Java Server Pages (JSP)

- A framework to generate dynamic web pages from embedded Java code in HTML.
- Java technology that helps software developers serve dynamically generated web pages based on HTML, XML, or other document types.
- How to use:
  - A fragment of Java program surrounded by <% %> is executed and the output of the program is embedded into HTML text.
  - Other code such as method definition or beans can be defined within <script> </script>
- Implementation: the HTML containing a JSP program is compiled dynamically (at reference time or web serve startup time) into the servlet program which generates the HTML text dynamically.
- Can use JavaBeans technology.
- Related technology: PHP (Hypertext Preprocessor)
- Different from JavaScript technology (executed in Client-side)

Example of JSP (Java Sever Pages)

```
<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>
```

WebService

- Not service using “Web” (browser)!
- Web Service is a framework to describe services (RPC) using WSDL (Web Service Description Language)
  - In almost case, use SOAP.
  - In almost case, use XML.
  - In almost case, use port 80 (HTTP).
  - But, other binding is possible (but none use it?)

Apache Axis

- Apache Axis is an implementation of Web Service by SOAP in java. (SOAP is a communication protocol for Remote procedure call in XML)
  - Implemented as a tomcat servlet
- Simple and quick usage:
  - Java Web Service (.jws), a remote procedure call for Java program
- More practical deployment:
  - Service generated by WSDL

Java web service

- A framework to publish a java code as a web service. (remote procedure call for Java routine)
- Change the file extension of the java program “Hello.java” to .jws, and place it at the root directory of Axis Web application.
- The WDSL description for .jws is seen by a web browser under the following URL:
  - http://localhost:8080/axis/Hello.jws?wsdl

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

Client code of Web service

```
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);
   }
}
```

Development of Web Service using WSDL

- **Usually, the interface of Web Service is described by WSDL (Web Sevice Description Language), and Stub code client/server) is generated from the WSDL by WSDL2Java.**
  - **The generated stub code for Client-side can be used without any modification**
  - **The stub code of Server-side is a skeleton, which give an outline of the sever code and will be modified by adding contents of services.**
  - **The Web Service becomes available by the deployment (install and setup the service code in the server).**
  - **Once deployed, WSDL description are accessed by Web Service URI?wsdl**

WSDL (WebSevice Description Language)

- An XML-based language that is used for describing the functionality offered by a Web service.
- Once a Web service is found, the user needs to know how to use, that is, an interface of the service, which is given in the form of WSDL.
  - Types --- Describes the data used in the message. XML Schema is used (inline or referenced) for this purpose
  - message ---Typically, a message corresponds to an operation. The message contains the information needed to perform the operation
  - Operation -- Defines the SOAP actions and the way the message is encoded.
  - Port Type --- Defines a Web service, the operations that can be performed, and the messages that are used to perform the operation.
  - Binding – Protocol of RPC and data format for a specific port type.
  - Port --- Defines the address or connection point to a Web service. It is typically represented by a simple
  - Service --- Contains a set of system functions that have been exposed to the Web-based protocols.

Example（hello.jws） 1/2

```
<?xml version="1.0" encoding="UTF-8" ?>
- <wsdl:definitions targetNamespace="http://localhost:8080/axis/Hello.jw

- <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
  </wsdl:input>
- <wsdl:output name="sayHelloResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding
```

Example（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>
```

Client code of Web Service

- Stub is generated by WSDL
  - 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);
  }
}
```

Deploy of Web Service (in Axis)

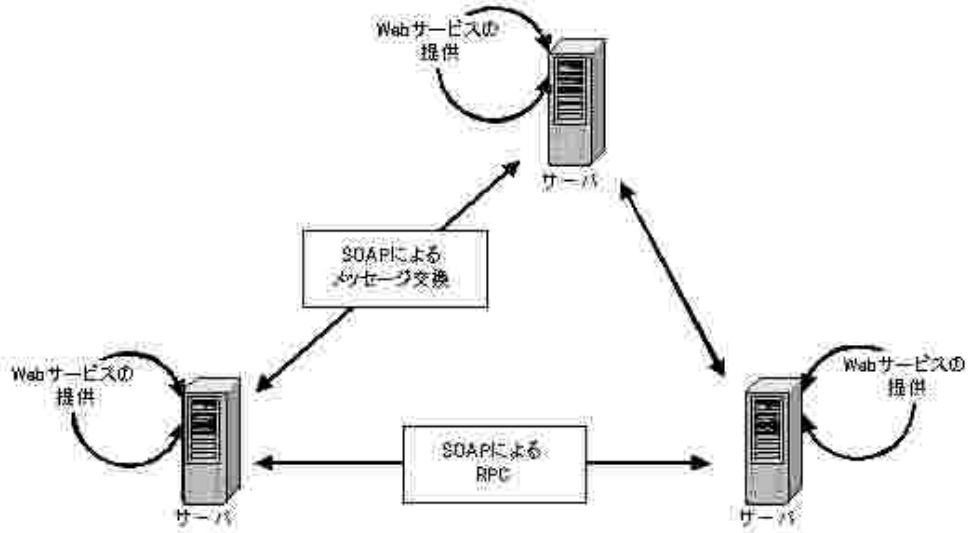
- Place compiled Echo.classfile to the path of Axis Web applications. For exmple:
  - webapps/axis/WEB-INF/classes/
- Create WSDD (Web Service Deploy descriptor) file: Service FQCN (Fully Qualified Class Name) and path, …

```
<service name="Version" provider="java:RPC">
 <parameter name="allowedMethods" value="getVersion"/>
 <parameter name="className" value="org.apache.axis.Version"/>
</service>
```

- Deploy by java.org.apache.axis.client.AdminClient with the WSDD.

SOAP

- **SOAP: Simple Object Access Protocol**
  - **Not “Object” anymore**
  - **XML based RPC protocol**
- **Open and simple, comparing with COM（Component Object Model）and CORBA（Common Object Request Broker Architecture**



What is SOAP

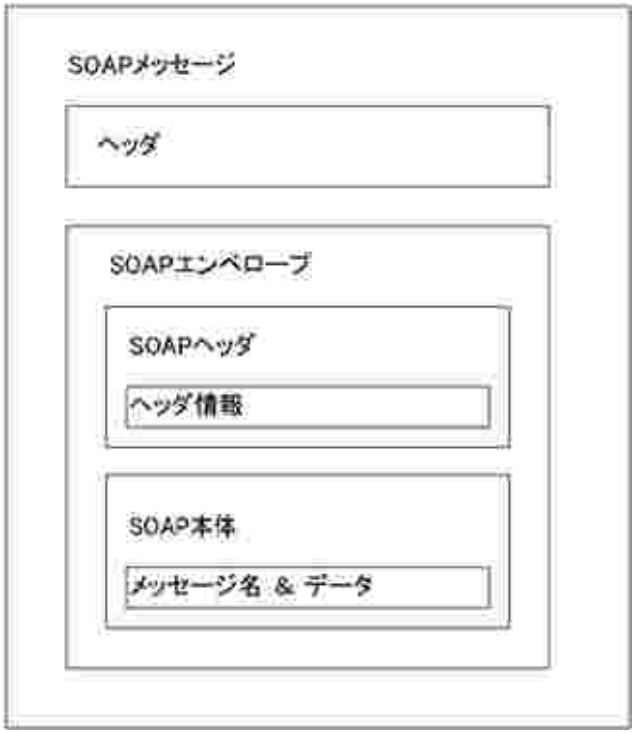
- RPC (remote procedure call) protocol using XML
  - Encapsulated in HTRP
  - Sender send POST message to URL
  - Receiver response as an message to HTTP
  - Perform RPC by specifying URL
- Frequently (mostly) used in Web Service
- Java class library
  - org.apache.soap.*

Example of SOAP RPC

```
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();
     …
  }
}
```

Structure of SOAP Message

- **Example of SOAP messages in HTTP binding**




```
POST /StockQuote HTTP/1.1                                   Header      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                                          envelop
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>                                  SOAP header      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>                           SOAP body       ④SOAP本体
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>                                             ②SOAPエンベロープ
```

Example of envelop

```
<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 in SOAP

- **The body of Method call message is:**

```
<method name>
    <parameter_name1>value</parameter_name1>
    <parameter_name2>value</parameter_name2>
 </method name>
```

- **The body of method reponse is:**

```
< method name Response>
    <parameter_name1>value</parameter_name1>
    <parameter_name2>value</parameter_name2>
 </ method name Response>
```

What messages are exchanged?

- You can check by port monitor

```
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 ";
    }
}
```

Client code:

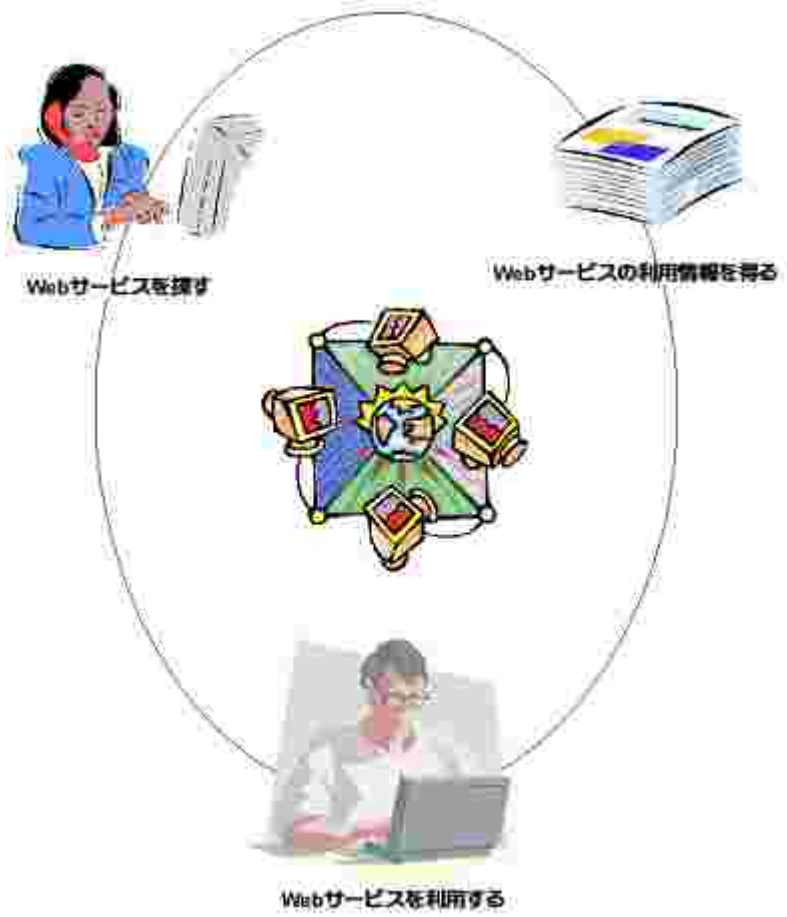
```
import java.util.*;
import java.net.*;
public class TestClient {
  public static void main( String [] args ) throws Exception {
    localhost.HelloWORDLocator locator =
      new localhost.HelloWORDLocator();
    URL url =
      new URL("http://localhost:4040/WS-I/services/HelloWORD");
    if( args.length > 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 ) ;
  }
}
```

```
[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>
```

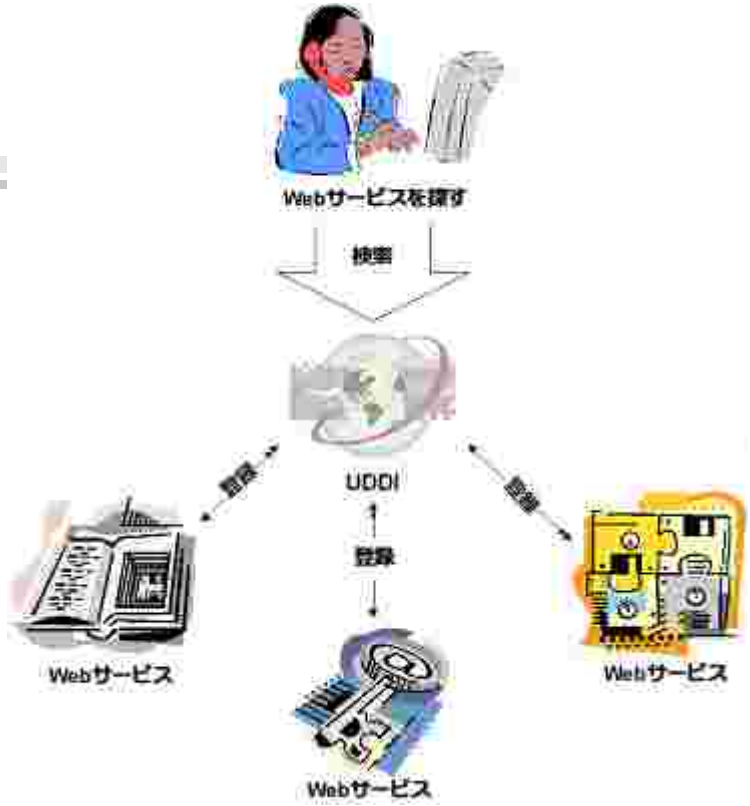
```
[HTTP Headers:]
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Date: Thu, 04 Sep 2003 12:43:22 GMT
Server: Apache Coyote/1.0
Connection: close

[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>
```

3 steps to make use of Web Service

- １． Looking for Web Service
  - ＵＤＤＩ
- ２． How to use Web Service
  - ＷＳＤＬ
- ３． Use Web Service
  - SOAP
  - ….



UDDI



- UDDI（Universal Description, Discovery and Integration）
- A platform-independent, Extensible Markup Language (XML)-based registry for businesses worldwide to list themselves on the Internet and a mechanism to register and locate web service applications.
- http://www.uddi.org/
  - White pages give information about the business supplying the service.
  - Yellow pages provide a classification of the service or business, based on standard taxonomies.
  - Green pages are used to describe how to access a Web Service, with information on the service bindings

An application example of WS: Google API

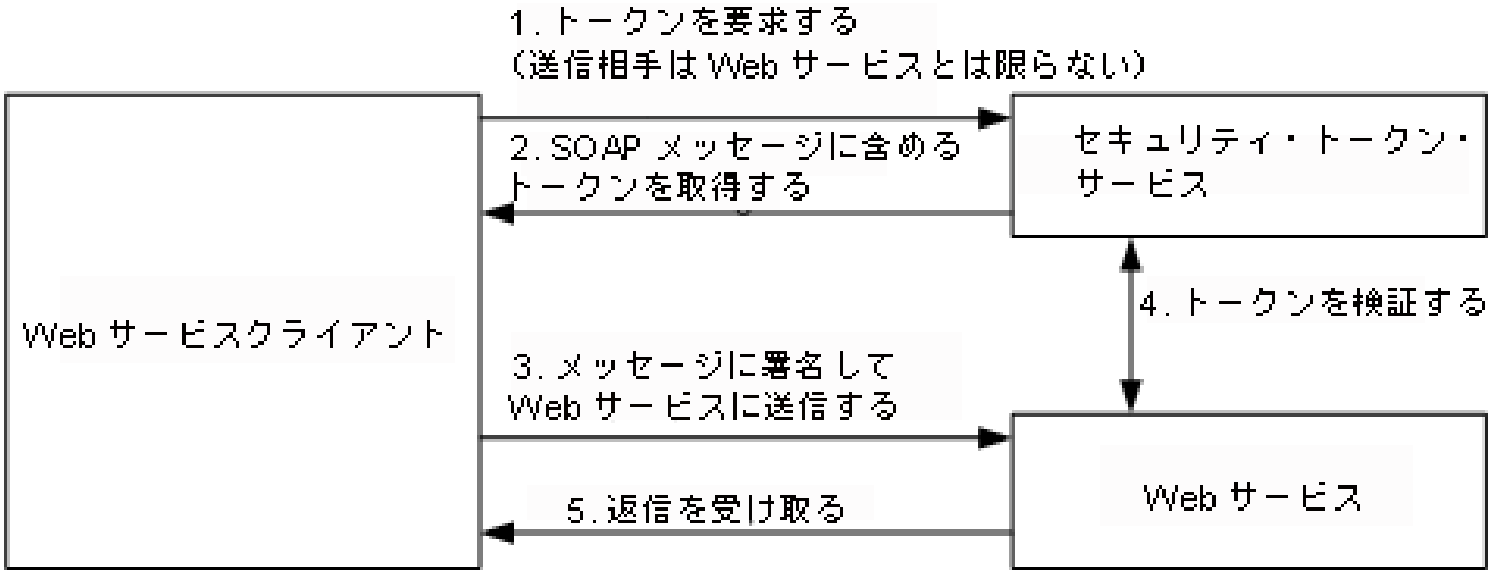
- http://www.google.com/apis/index.html
- Google is used to provide their search engine via web service, which was called “Google API” (2004)
- Google expected that someone would create/invent a new application using their search engin via web service.
- But, … stopped … unfortunately.
- Move to more RIA (Rich Internet Application) frame work such as google map, … via Ajax, javascript.

WS-Security

- **Security of Web Service**
  - **SOAP is a protocol to transfer messages in the form of HTTP between two end points.**
  - **How about using HTTPS for secure communication?**
  - **It not enough, Why?**
    - **The messages of WS may be forwarded to other severs.**
    - **It may be routed via complicated network or via non-HTTP protocol.**
- **WS-Security defines the header elements to contain security data in SOAP header**
  - **By extending the existing specification, WS-Security provides a framework to include security mechanism in SOAP messsages without depending transport layers.**

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>
```

参考

- 「Ajaxを勉強しよう」
  http://www.openspc2.org/JavaScript/Ajax/Ajax_study/index.html

- Life is beautiful ：Ajaxの本質、「非同期メッセージ型ウエブ・アプリケーション」のススメ
  http://satoshi.blogs.com/life/2005/06/ajax.html

What is Ajax

- Ajax is「Asynchronous JavaScript + XML」
- Programming with JavaScript and XML using asynchronous communications, that is, JavaScript program which communicates XML data asychronously to servers.

- Main technology for RIA (Rich Internet Application)
  - Google Maps, Gmail Google are RIAs using Ajax technology.

A history

- **Before DHTML**
  - **Only static HTML pages, any texts and images don't move. These cannot be move or modifiy.**

- **In DHTML, the program in web browser maniplate the contest in form of HTML. The text or images can be handled as an object in the program in web browser.**
  - **HTML data is handled as a DOM(Document Object Model)**
  - **Using JavaScript, DOM including style sheets can be accessed.**
  - **Available from IE4. Too many bugs, still many problems.**

Related Technology

- DHML
- Flash
- Java Applet
  - Only working in Applet
  - DHML/Ajax can process a whole HTML document,
  - and perform communication to server.

Sample Program

- test1: 簡単な例
- test2: HMLの表示
- test3：XMLの表示
- test4：画像(gif)の表示
- test5: HTMLの書き換え
- test6: HTMLの書き換え
- map-test: google mapの最初のサンプル
- map-test1: ズーム、情報ウインドウ
- map-test1: イベント、マーカー

Basic example (1/2)

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;charse
<title> test1 </title>
<script type="text/javascript"> <!--
function loadTextFile(){
httpObj = new XMLHttpRequest();
httpObj.onload = displayData;
httpObj.open("GET","data.txt",true);
httpObj.send(null);
}
function displayData(){
document.myform.result.value = httpObj.responseText;
}
// --> </script>
</head>

<body>
```

Basic Example (2/2)

```
</head>

<body>
<h1> test1 </h1>
簡単な例 <br>
<form name="myform">
<input type="button" value="read" onClick="loadTextFile()"

<textarea name="result" cols="40" raws="5"></textarea>
</form>
</body>
</html>
```

Asynchronous communication with Sever

- Create communication objects

```
httpObj = new XMLHttpRequest()
```

  - In IE, `ActiveXObject("Microsoft.XMLHTTP");`

- Send the request to get data to Server

```
httpObj.open("GET","data.txt",true);
httpObj.send(null);
```

- When a reply to the request is returned, response it. To do this, set a handler:

```
httpObj.onload = displayData;
```

When a reply to load request is received, the handler is called.

Programming in JavaScript

- Set Javascript handler for the action when GUI such as a button is modified (pushed).

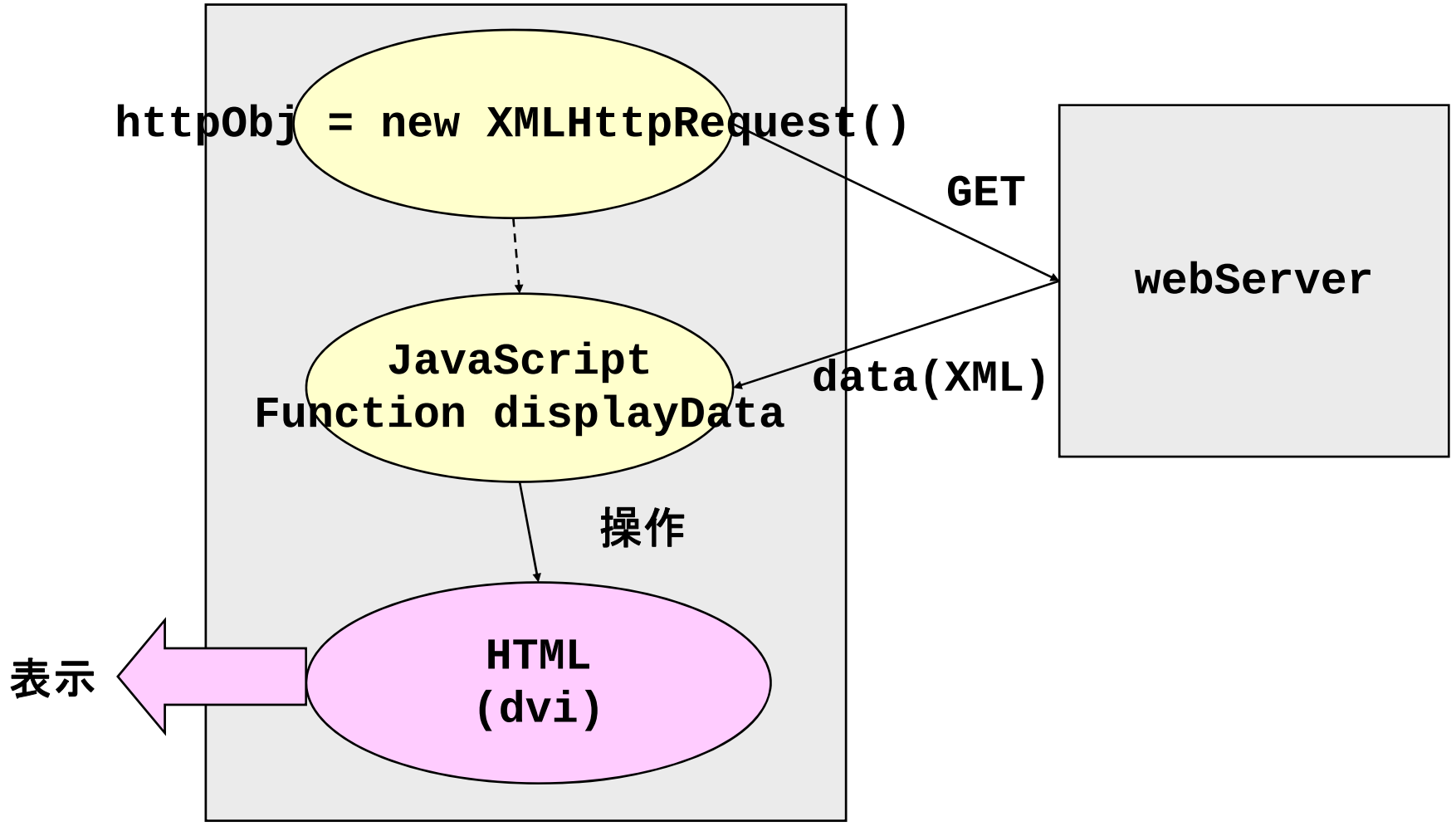
```
<form name="myform">
<input type="button" value="read"
  onClick="loadTextFile()"><br>
  ….
</form>
```

Modify HTML:test1.html

- Set value from JavaScript

```
function displayData(){
document.myform.result.value =
  httpObj.responseText;
}

<textarea name="result" cols="40"
  raws="5"></textarea>
```

Model of Asynchronous communications



Using div : test2.html

- Put div-tag in HTML source

```
<div id="resultData"> </div>
```

- document.getElementById(tagId); Find tag and rewrite it.
- Access the element of HTML using id

```
function displayData(){
if((httpObj.readyState == 4) &&
    (httpObj.status == 200)) {
   $("resultData").innerHTML = httpObj.responseText;
} else {
   $("resultData").innerHTML = "<b> Loading ... </b>";
}
```

Mylib.js

```
<script type="text/javascript" src="mylib.js"></script>

// library
function createXMLHttpRequest(cbFunc)
{
 var XMLhttpObj = null;
 XMLhttpObj = new XMLHttpRequest();
 XMLhttpObj.onreadystatechange=cbFunc;
 return XMLhttpObj;
}

function $(tagId)
{
  return document.getElementById(tagId);
}
```

Display XML : test3.html

- Receive XML data by responseXML
- Display it as modify DOM.

```
xmlData = httpObj.responseXML;
 userListTags = xmlData.getElementsByTagName("user");
 numberListTags = xmlData.getElementsByTagName("number");
 usernameListTags = xmlData.getElementsByTagName("username");
 userLen = userListTags.length;
 resultText = "";
 for(i=0; i<userLen; i++){
 num = numberListTags[i].childNodes[0].nodeValue;
 uname = usernameListTags[i].childNodes[0].nodeValue;
 resultText = resultText + num + " : " + uname + "<br>";
 }
 document.getElementById("resultData").innerHTML = resultText;
```

Display image：test4.html

- Set image by
- Web Browser get and display image by <img src= ..">

Rewrite HTML:test5.html

- **Rewrite attribute in HMTL**
  - **Ex: Modify background color**

```
function setBGColor(){
 document.body.bgColor = "#ffffef";
}
```

- **It is possible to add the tag**

```
function addImage(){
 imgObj = document.createElement("img");
 imgObj.setAttribute("src","mark1.gif");
document.getElementById("subContent").appendChild(imgObj)
  ;
}
```

- **Style sheet CCS can be modified.**

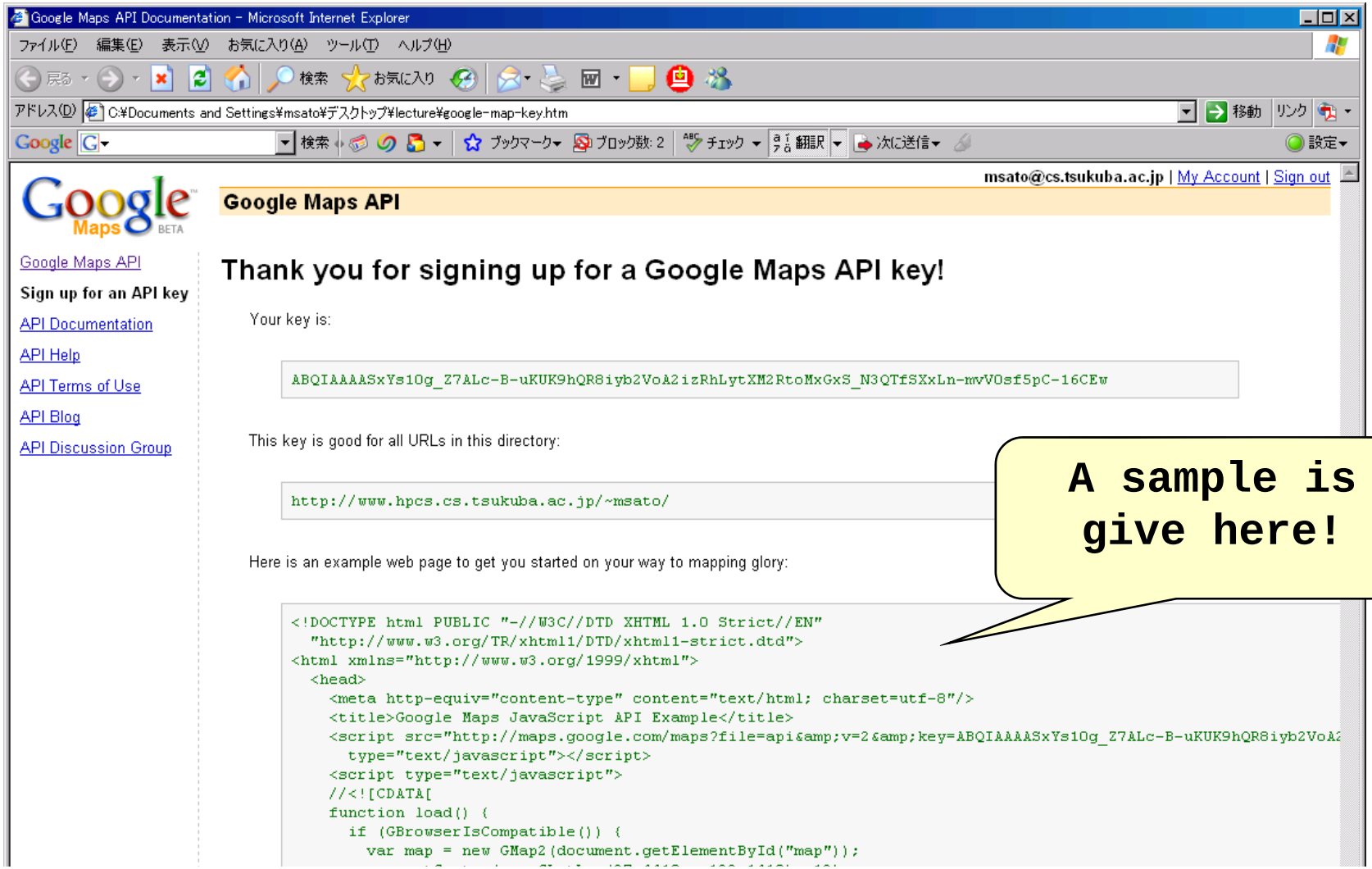
Generate and modify HMTL：test6.html

- Create images
- Modify HTMLattribute
  - Src attribute of Img tag

```
function changeContent(){
 document.getElementById("subContent").
    childNodes[1].setAttribute("src","mark1.gif");
}
```

GoogleMaps

- As a first step, register and obtain a key.



GoogleMaps

```
<!DOCTYPE html>
<html>
  <head>
    <title>Simple Map</title>
    <meta name="viewport" content="initial-scale=1.0">
    <meta charset="utf-8">
    <style>
      /* Always set the map height explicitly to define the size o¥
f the div
       * element that contains the map. */
      #map {
        height: 100%;
      }
      /* Optional: Makes the sample page fill the window. */
      html, body {
        height: 100%;
        margin: 0;
        padding: 0;
 }
    </style>
  </head>
  <body>
```

Map-test1.html (1)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=ut¥
f-8"/>
    <title>Google Maps JavaScript API Example</title>
    <script type="text/javascript">
      //<![CDATA[
var map;
function initMap() {
 map = new google.maps.Map(document.getElementById('map'),
     { // #mapに地図を埋め込む
     center: { // 地図の中心を指定
          lat: 36.10, // 緯度
          lng: 140.10 // 経度
       },
function newPoint(x,y,z){
   var latlng = new google.maps.LatLng(y, x);
   map.setCenter(latlng);
   map.setZoom(z);
}
```

Map-test1.html (2)

```
function setFuji() {
  newPoint(138.73123168945312, 35.3565762019612 1,10);
}

function dispInfo()
{
   setFuji();
  var latlng = new google.maps.LatLng(35.3565762019612 1,138.731231¥
68945312);
  var iwopts = {
    content: 'this is Mt. Fuji',
    positon: latlng
  };
  var infowindow = new google.maps.InfoWindow(iwopts);
  infowindow.open(map);
}
 <script src="https://maps.googleapis.com/maps/api/js?key=AIzaS¥
yC_ZHg9MnTADt0odSy8o3Rzdyc1i5_rRPU&callback=initMap"
    async defer></script>
  </head>

  <body>
```

Map-test1.html (3)

```
  <body>
    <div id="map" style="width: 1000px; height: 600px"></div>
    <form>
      <input type="button" value="(137,36) zoom 13" onClick="newPo¥
int(137,36,13)" />
      <br />
      <input type="button" value="(137,36) zoom 10 " onClick="newP¥
oint(137,36,10)" />
      <br />
      <input type="button" value="fuji" onClick="setFuji()" /> <br¥
 />
      <input type="button" value="set fuji Info" onClick="dispInfo¥
()" /> <br />
 </form>
  </body>
</html>
```

Create map : map-test1.html

- **initMap**

```
var map;
function initMap() {
 map = new
  google.maps.Map(document.getElementById('map'),
     { // #mapに地図を埋め込む
     center: { // 地図の中心を指定
          lat: 36.10, // 緯度
          lng: 140.10 // 経度
       }, }
```

- **Load and initFunc**

```
<script
 src=https://maps.googleapis.com/maps/api/js?key=A
 IzaS¥yC_ZHg9MnTADt0odSy8o3Rzdyc1i5_rRPU&callback=
 initMap async defer></script>
```

Zoom, information window : map-test1.html

- Zoom

```
function newPoint(x,y,z){
   var latlng = new google.maps.LatLng(y, x);
   map.setCenter(latlng);
   map.setZoom(z);
}
```

- Display information window:

```
var iwopts = {
  content: 'this is Mt. Fuji',
  positon: latlng
};
var infowindow = new
google.maps.InfoWindow(iwopts);
infowindow.open(map);
```

Map-test2.html (1)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=ut¥
f-8"/>
    <title>Google Maps JavaScript API Example</title>
    <script src="https://maps.googleapis.com/maps/api/js?key=AIzaS¥
yC_ZHg9MnTADt0odSy8o3Rzdyc1i5_rRPU&callback=initMap"
    async defer></script>
    <script type="text/javascript">
var map;
function initMap() {
 map = new google.maps.Map(document.getElementById('map'),
     { // #mapに地図を埋め込む
     center: { // 地図の中心を指定
          lat: 36.10, // 緯度
          lng: 140.10 // 経度
 zoom: 15 // 地図のズームを指定
   });
map.addListener('click', function(e) {
     getXY();
  });
}
```

Map-test2.html (2)

```
function getXY()
{
var latlng = map.getCenter();
document.getElementById("mapX").innerHTML = "<b>"+latlng.lat()+"</¥
b>";
document.getElementById("mapY").innerHTML = "<b>"+latlng.lng()+"</¥
b>";
}
function setMarker()
{
  var latlng = map.getCenter();
  var marker = new google.maps.Marker({
    position: latlng,
    map: map});
}
    </script>
  </head>
  <body>
    <div id="map" style="width: 1000px; height: 600px"></div>
<form>
<input type="button" value="set marker" onClick="setMarker()" />
</form>
```

Map-test2.html (3)

```
<body>
    <div id="map" style="width: 1000px; height: 600px"></div>
<form>
<input type="button" value="set marker" onClick="setMarker()" />
</form>
<ul>
<li> Longitude : <div id="mapY" /> </li>
<li> Latitude : <div id="mapX" />  </li>
</ul>
</ul>
</body>
</html>
```

Mouse , marker : map-test2.html

- Setting mouse event

```
map.addListener('click', function(e) {  getXY(); });
```

- Event handler

```
function getXY()
{
var latlng = map.getCenter();
document.getElementById("mapX").innerHTML =
  "<b>"+latlng.lat()+"</¥
b>";
document.getElementById("mapY").innerHTML =
  "<b>"+latlng.lng()+"</¥
b>";
}
```

- Display region

```
<ul>
<li> Longitude : <div id="mapY" /> </li>
<li> Latitude : <div id="mapX" />  </li>
</ul>
```

Mouse, marker : map-test2.html

- Display marker

```
function setMarker()
{
  var latlng = map.getCenter();
  var marker = new google.maps.Marker({
    position: latlng,
    map: map});
}
```

- Setting

```
<form>
<input type="button" value="set marker" onClick="setMarker()"
   />
</form>
```

What is new in Ajax and RIA

- The essential point of the second generation of Web applications, Google is developing, is not using XHTML or XML, Javascript. 、

① No need of explicit installation

② Don't block User interface due to waiting the response, by asynchronous communications with server.

③ Use massage passing explicitly, not RPC with servers.

④ Data binding (how to display) should be implemented in client-side, not sever-side.

⑤ User interface should have intelligence to allow interaction with users without communicating servers.

- （１）No need of explicit installation
  - This is essential point in web-based applications
  - Trends to Web OS

- （２）Don’t block User interface due to waiting the response, by asynchronous communications with server.
  - First generation of web application often makes the user wait on every click on hyper-link until contents is coming.
  - By using asynchronous communication, the latency (the time for communication) should be hided so that it provides easy-to-use user interface.

- （３）Use massage passing explicitly, not RPC with servers.
  - If you use RPC, it must be asynchronous.

- （４）Data binding (how to display) should be implemented in client-side, not sever-side.
  - In First generation of web applications, all view is decided by server-side. That means binding of view to data is done in sever-side.
  - In Second generation of web applications, view is decided in client-side by using Javascript. Javascript access the server to get data, and change view.
  - It improve the quality and usability of user-interface.
  - これはただしい！

- （５）User interface should have intelligence to allow interaction with users without communicating servers.
  - Rich library and tools of javaScript

Summary and comments

- Now, Web become main user interface for computers. The concept of programming is changing…
  - What is “programming”?
    - Wiki, … web 2.0
  - Programming environment also change …
- Everything will be a network or web …
  - Cloud …