


XML

Allen Long
Email: allen@huihoo.com
http://www.huihoo.com
2004-04

内容安排

- XML基础
- Java+XML简介
- Java API for XML
  - JAXP (Processing API)
  - JAXB (Project Adelard)
  - JAXM (Messaging API - ebXML)
  - XML和Java 2 平台，企业版

XML是什么？

- XML – Extensible Markup Language
- Based upon HTML
- Describe your own tags
- Uses DTD (Document Type Definition) to describe the data
- XML is not a replacement for HTML
- XML is a language for creating other languages
- Documents follow the custom language a user develops from XML
- Labeled information in XML can be reused
- Need to follow the rules accordingly

HTML存在的问题

- Do not give information about content of the web page
- Hard to be able to reuse this information
- HTML are hard to display from browser's point of view because of HTML's simplicity
- Limited in areas of formatting and dynamic content

JAVA与XML完美结合

1. Java平台是一种跨平台的编程环境
2. XML是一种跨平台的数据格式
3. 几乎所有的XML工具使用的都是Java编程语言
4. 与其他语言相比，Java平台提供了更好的XML支持

JAXP-Java平台上的解析API

JAXP

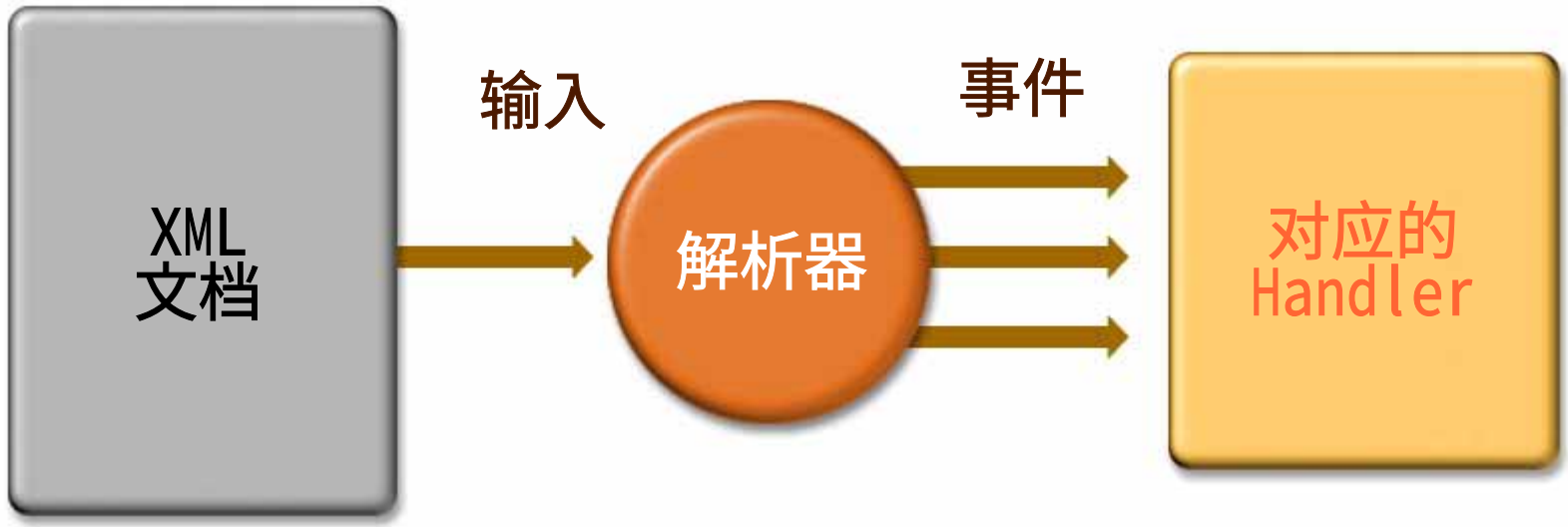



- 用于解析的瘦型、轻量级API
- 用于转换XML文档的API
- 可嵌入式的解析器和XSLT引擎
- 解析XML使用：
  - 事件驱动（SAX）
  - 基于树型结构（DOM）
  - XSL转换

SAX

- Simple API for XML的缩写
- 串行存取文档
- 轻量级、快速
- 编程较难
- 仅用于串行存取
- org.xml.sax.*

SAX



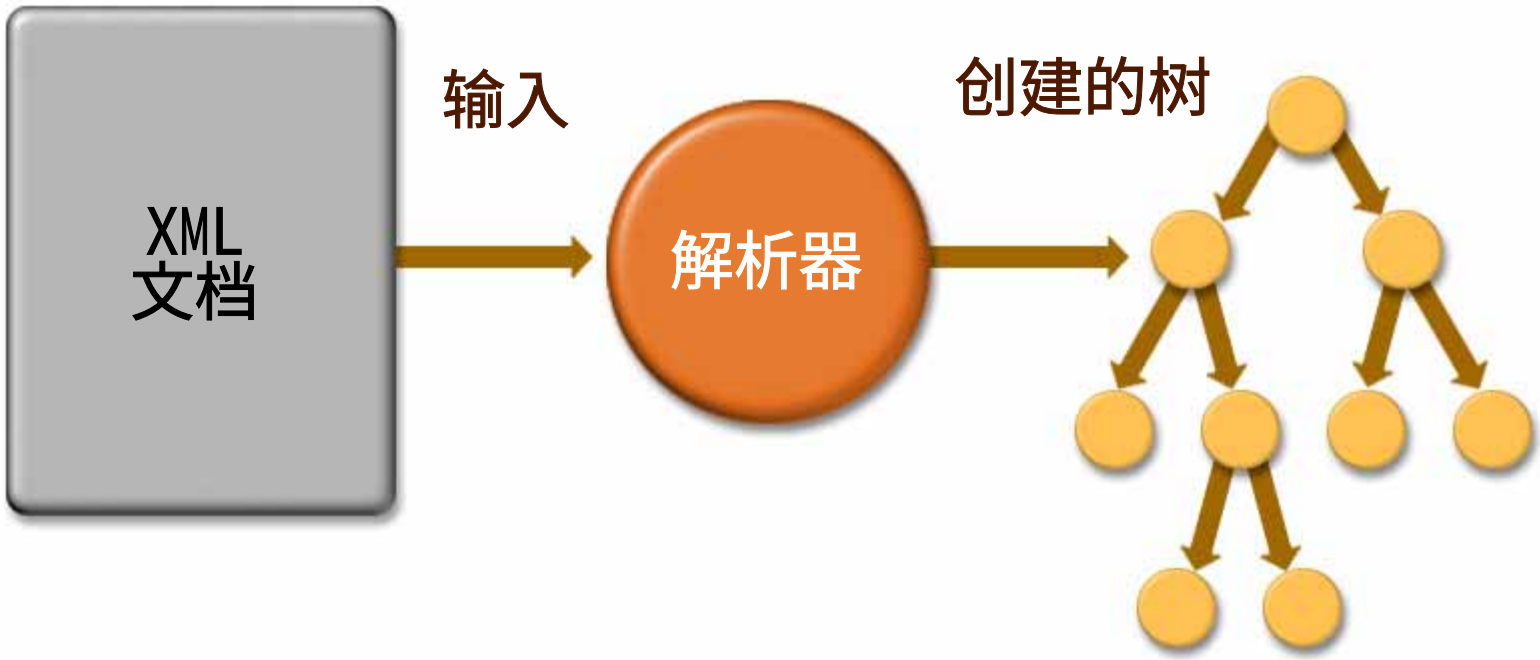
JAXP/SAX 代码例子

```
01 import java.xml.parsers.*;
02 import org.xml.sax.*;
03
04 SAXParserFactory factory =
05                 SAXParserFactory.newInstance();
06 factory.setValidating(true);
07 SAXParser parser = factory.newSAXParser();
08 parser.parse(“config.xml",handler);
09
10 // can also parse InputStreams, Files, and
11 // SAX input sources
```

DOM

- Document Object Model的缩写
- 通过树型结构存取XML文档
- 由元素节点和文本节点组成
- 可以在树的某个节点上向前或向后移动
- 与SAX相比需要更大的内存
- org.w3c.dom.*

DOM



JAXP/DOM 代码例子

```
01 import java.xml.parsers.*;
02 import org.w3c.dom.*;
03
04 DocumentBuilderFactory factory =
05     DocumentBuilderFactory.newInstance();
06 factory.setValidating(true);
07 DocumentBuilder builder =
08     factory.newDocumentBuilder();
09 Document doc =
10     builder.parse(“config.xml");
11
12 // can also parse InputStreams, Files, and
13 // SAX input sources
```

JAXP中的XSLT API

- javax.xml.transform
  - XSLT处理器的基本接口集
  - 定义了TransformerFactory和Transformer类
  - 定义了Templates, Source and Result接口
    - Templates表示处理指令
    - 在Source和Result接口中可以使用 SAX, DOM 和 stream

Transform 的代码例子

```
01 import java.xml.transform.*;
02
03 Transformer trans;
04 TransformerFactor fac = new
05    TransformerFactory.newInstance();
06 try {
07     // Create a tranform for a stylesheet
08     trans = fac.newTransformer(
09        new StreamSource(stylesheet));
10     // Apply transform to System.out
11     trans.transform(new StreamSource(source),
12         new StreamResult(System.out));
13 } catch (Exception e) {
14     // handle error
15 }
```

解析器的指定

- 使用系统属性查找 Factory
  - javax.xml.parsers.SAXParserFactory
  - javax.xml.parsers.DocumentBuilderFactory
  - javax.xml.parsers.TransformerFactory
- 通过改变属性可以使用任意的解析器
- $JAVA_HOME/lib/jaxp.properties文件

改变属性

Command:

```
java -
Djavax.xml.parsers.SAXParserFactory=MyParser
Factory MyClass
```

Code:

```
System.setProperty(
  "javax.xml.parsers.SAXParserFactory",
  "foo.bar.MyParserFactory");
```




用于XML绑定的Java API - JAXB

XML数据绑定

- XML = 可移动的数据
- XML 代表的数据没有任何意义
- 模式（Schemas）为XML增加了意义
- 绑定使XML可以容易地在程序中使用

XML = 没有意义的数据

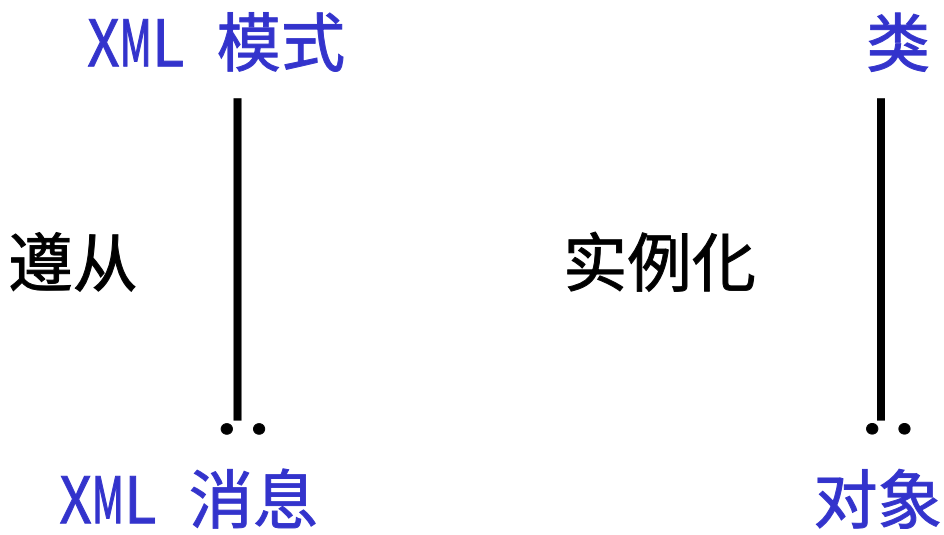
```
<ShoeOrder id="4040458"
           style="Sandal">
  <color>Brown</color>
  <size>9 1/2</size>
</ShoeOrder>
```

它们相同吗?

```
<ShoeOrder id="4040458"
           style="Sandal"
           colour="Brown"
           size="42”>
</ShoeOrder>
```

模式(Schemas)为XML增加了意义

模式 = XML消息的语义和语法描述



在程序中使用XML

对应XML消息的类

```
public class ShoeOrder {
    public ShoeOrder(String id, Style style,
                     String color, String size);
    public String  getId();
    public void    setId(String id);
    public Style   getStyle();
    public void    setStyle(Style style);
    public String  getColor();
    public void    setColor(String color);
    public String  getSize();
    public void    setSize(String size);
}
```

在程序中使用XML

Marshalling/unmarshalling 代码

```
Public void acceptOrder(InputStream in) {
    ShoeOrder so = unmarshal(in);
    WarehouseDB.submit(so);
}
```

编写 unmarshal?

在程序中使用XML

如何编写unmarshal? 使用SAX!

```
private static ShoeOrder newOrder = null;

static class DocHandler implements DocumentHandler {

  public void setDocumentLocator(Locator l) { }
  public void startDocument() { }
  public void endDocument() { }
  public void ignorableWhitespace(char[] cbuf, int offset, int len) { }
  public void processingInstruction(String target, String data) { }

  ShoeOrder so = null;
  String cur = null;

  public void startElement(String name, AttributeList al) {
    if (name.equals("ShoeOrder")) {
      so = new ShoeOrder();
      for (int i = 0, n = al.getLength(); i < n; i++) {
            String an = al.getName(i);
            if (an.equals("id")) {
              so.setId(al.getValue(i));
            } else if (an.equals("style")) {
              so.setStyle(al.getValue(i));
            } else {
              throw new RuntimeException("Unknown attribute: "
                                         + an);
            }
      }
    } else {
      cur = name;
    }
  }
```

在程序中使用XML


```
 public void characters(char[] cbuf, int offset, int len) {
    if (cur == null) return;
    String val = new String(cbuf, offset, len);
    if (cur.equals("color")) {
      so.setColor(val);
    } else if (cur.equals("size")) {
      so.setSize(Integer.parseInt(val));
    } else if (cur.equals("width")) {
      so.setWidth(val);
    } else {
      throw new RuntimeException("Unknown element: " + cur);
    }
  }

  public void endElement(String name) {
    if (name.equals("ShoeOrder")) {
      newOrder = so;
    } else {
      cur = null;
    }
  }

}

public static ShoeOrder unmarshal(InputStream in)
        throws SAXException
{
    InputSource is = new InputSource(in);
    Parser p = ParserFactory.makeParser();
    p.setDocumentHandler(new DocHandler());
    p.parse(is);
    return newOrder;
}
```

在程序中使用XML 使用DOM!


```
Public static ShoeOrder unmarshal(InputStream in)
    throws IOException, SAXException
{
    XmlDocument xd = XmlDocument.createXmlDocument(in, false);
    Element r = xd.getDocumentElement();
    ShoeOrder so = new ShoeOrder();
    so.setId(r.getAttribute("id"));
    so.setStyle(r.getAttribute("style"));
    for (Node n = r.getFirstChild(); n != null; n = n.getNextSibling()) {
        if (n instanceof Element) {
            Element e = (Element)n;
            String tn = e.getTagName();
            if (tn.equals("color")) {
                String val = ((CharacterData)e.getFirstChild()).getData();
                so.setColor(val);
            } else if (tn.equals("size")) {
                String val = ((CharacterData)e.getFirstChild()).getData();
                so.setSize(Integer.parseInt(val));
            } else if (tn.equals("width")) {
                String val = ((CharacterData)e.getFirstChild()).getData();
                so.setWidth(val);
            } else {
                throw new RuntimeException("Unknown element: " + tn);
            }
        }
    }
    return so;
}
```

在程序中使用XML

不使用SAX,DOM

使用这些方法产生的问题

- 需要编写代码
- 因为涉及模式，需要维护代码

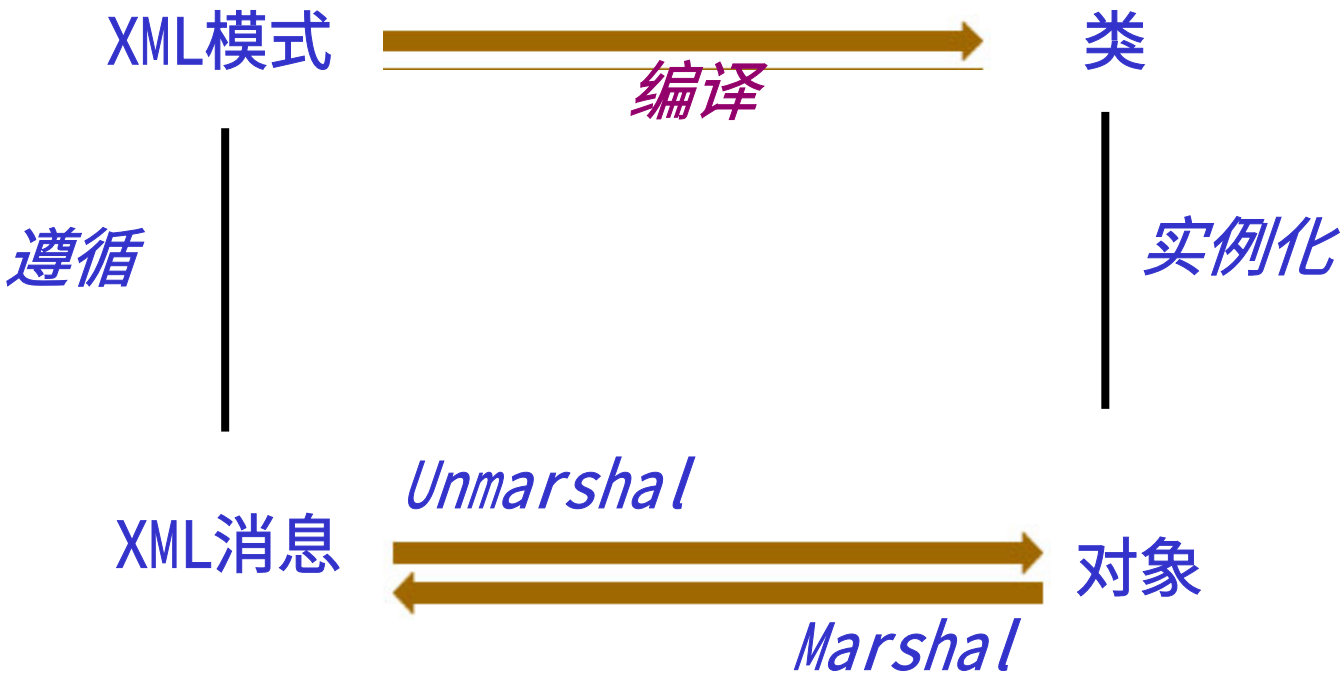
是否有更好的方法…

将XML绑定到程序

```
public void acceptOrder(InputStream in) {
    ShoeOrder so = hoeOrder.unmarshal(in);
    WarehouseDB.submit(so);
}
```

将XML绑定到程序

绑定(Binding)将模式编译成类



将XML绑定到程序

- 绑定产生的类包含：
  - Marshalling/unmarshalling代码
    - 完全错误和有效性检查
  - 组件存取方法（get/set）
    - 确保与模式的一致性
- 最大优势：简化了创建和维护

将XML绑定到程序

可能创建一个无效的ShoeOrder吗? 不可能!

```
ShoeOrder so = ShoeOrder.unmarshal(in);

so.setColor(“Red”);    // 抛出异常
so.setSize("5 3/4");   // 抛出异常
so.setWidth("Z");      // 抛出异常
```

将XML绑定到程序

```
public class ShoeOrder {
    public void                marshal(OutputStream);
    public static ShoeOrder    unmarshal(InputStream);
    public ShoeOrder(String id, Style style,
                     String color, String size);
    public String  getId();
    public void    setId(String id);
    public Style   getStyle();
    public void    setStyle(Style style);
    public String  getColor();
    public void    setColor(String color);
    public String  getSize();
    public void    setSize(String size);
 }
```




用于消息处理的Java API--- JAXM

Java API for XML Messaging

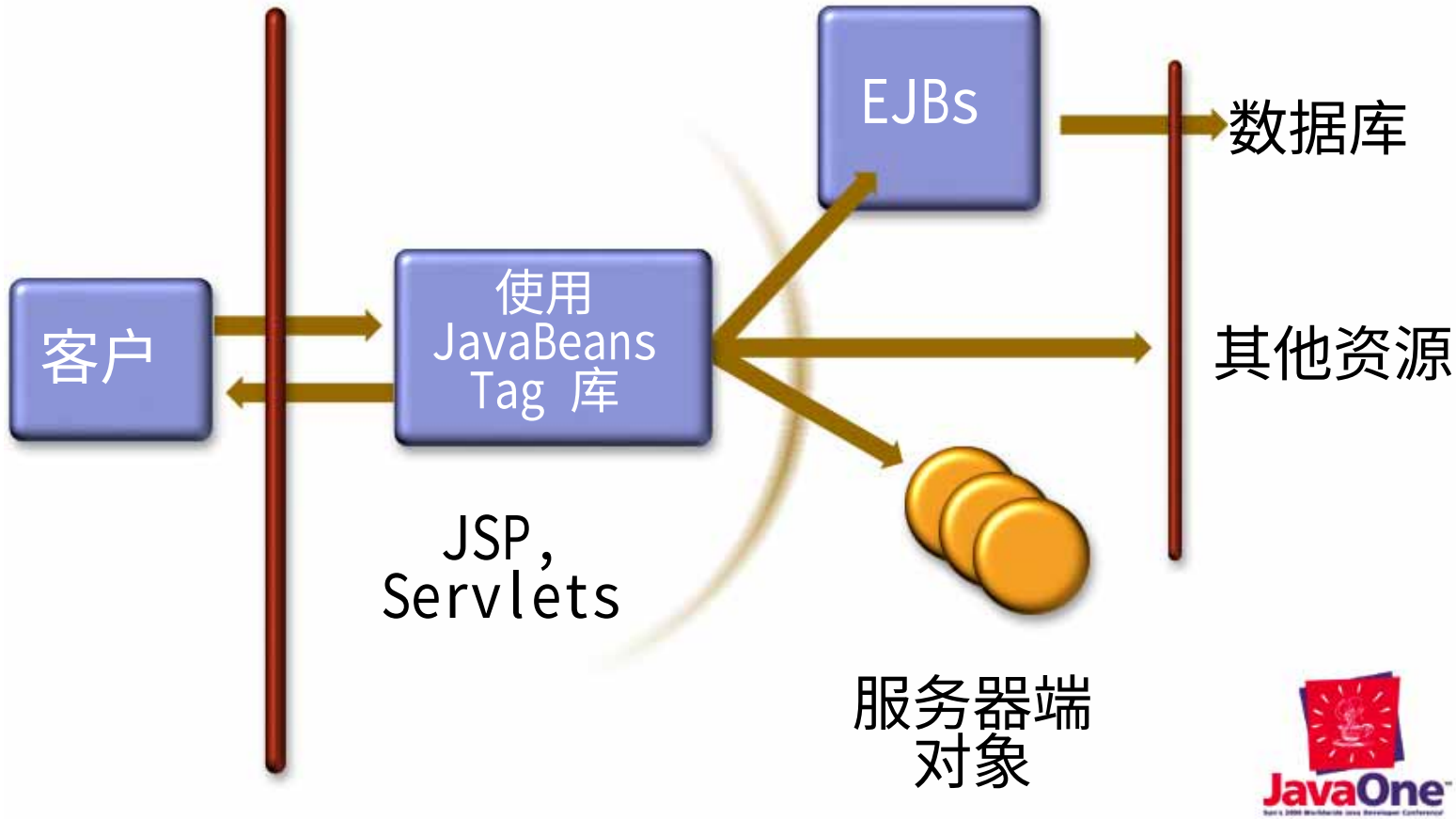
JAXM和ebXML

- ebxml.org
- JAXM
  - 用于消息处理的API
- M项目
  - 参考实现

什么是ebXML?

- ebXML是两个组织共同努力的结果
  - OASIS (Organization for Advancement of Structured Information Standards的缩写)
  - UN/CEFACT (United Nations Center for Trade Facilitation and Electronic Business的缩写)
- 关注于电子商务XML
- 目标是建立一个标准的电子商务平台
- ebXML.org消息服务工作组已经公布了下列文档
  - 简介和需求文档
  - 消息信封规范
  - Strawman消息头规范

J2EE和XML

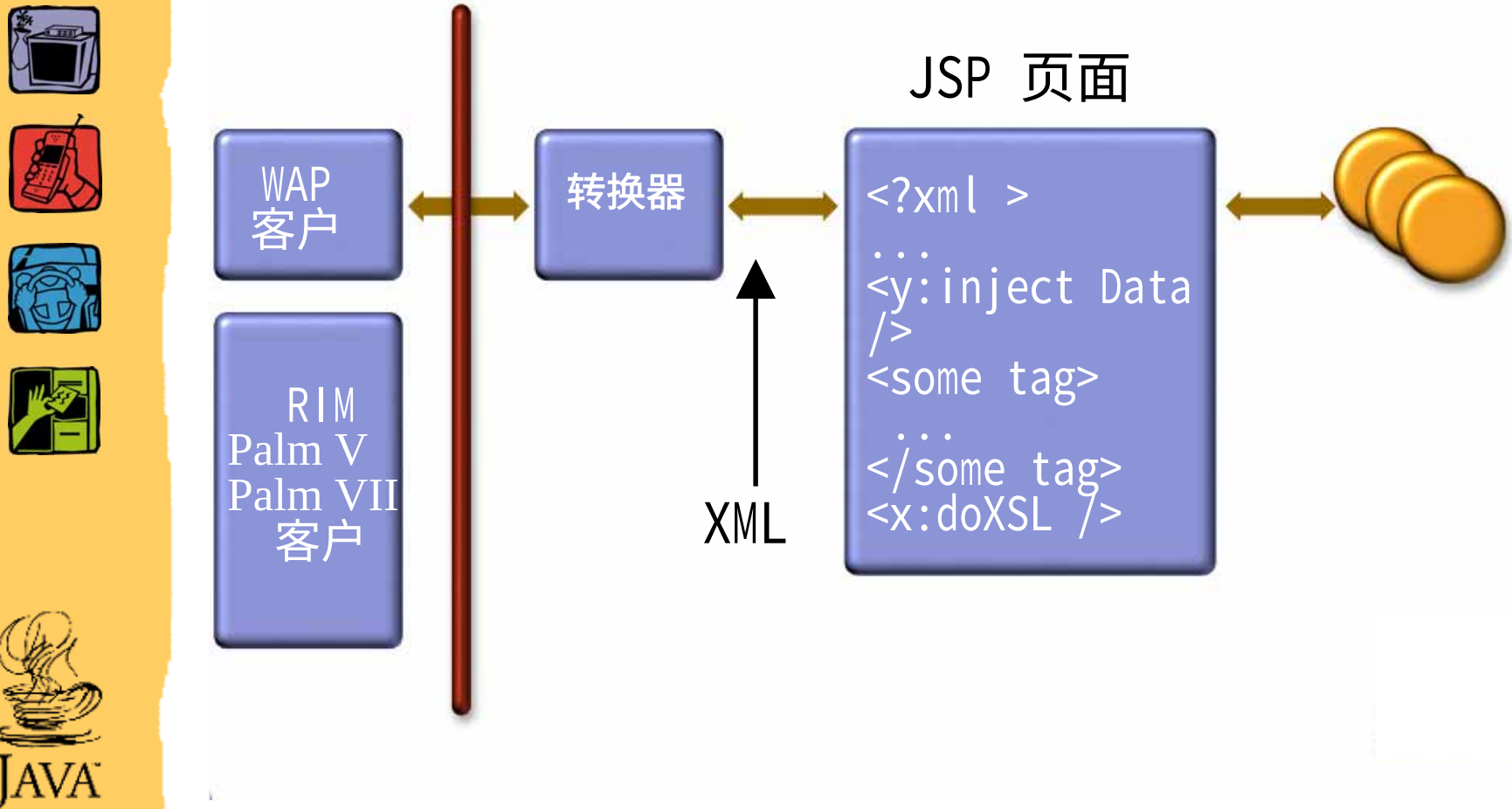


XML和JSP技术

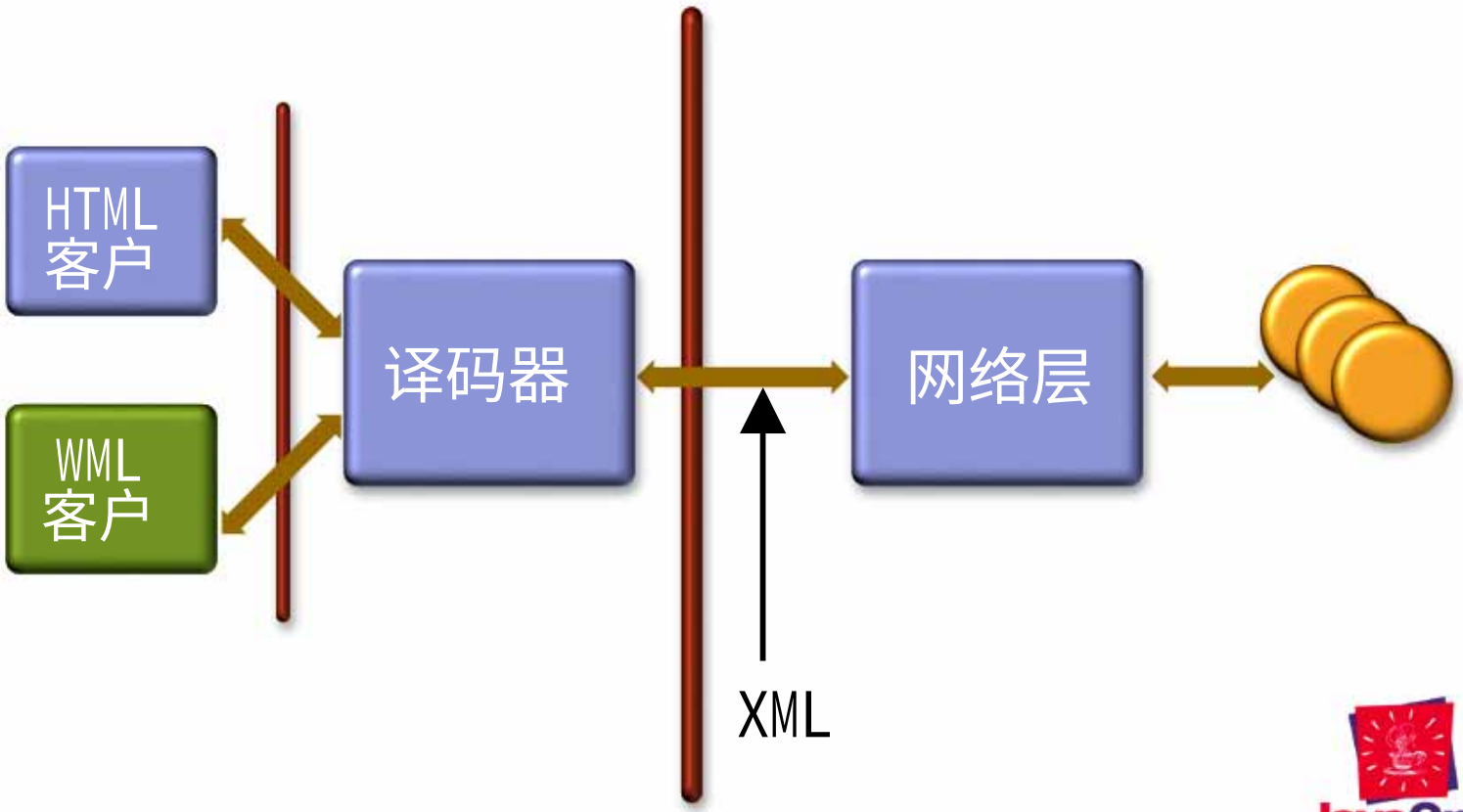



- 互相补充
  - 操作XML以完成不同的任务
  - 来自XML的数据
    - 数据库查询
  - 通用数据 + XSLT 作为表现层

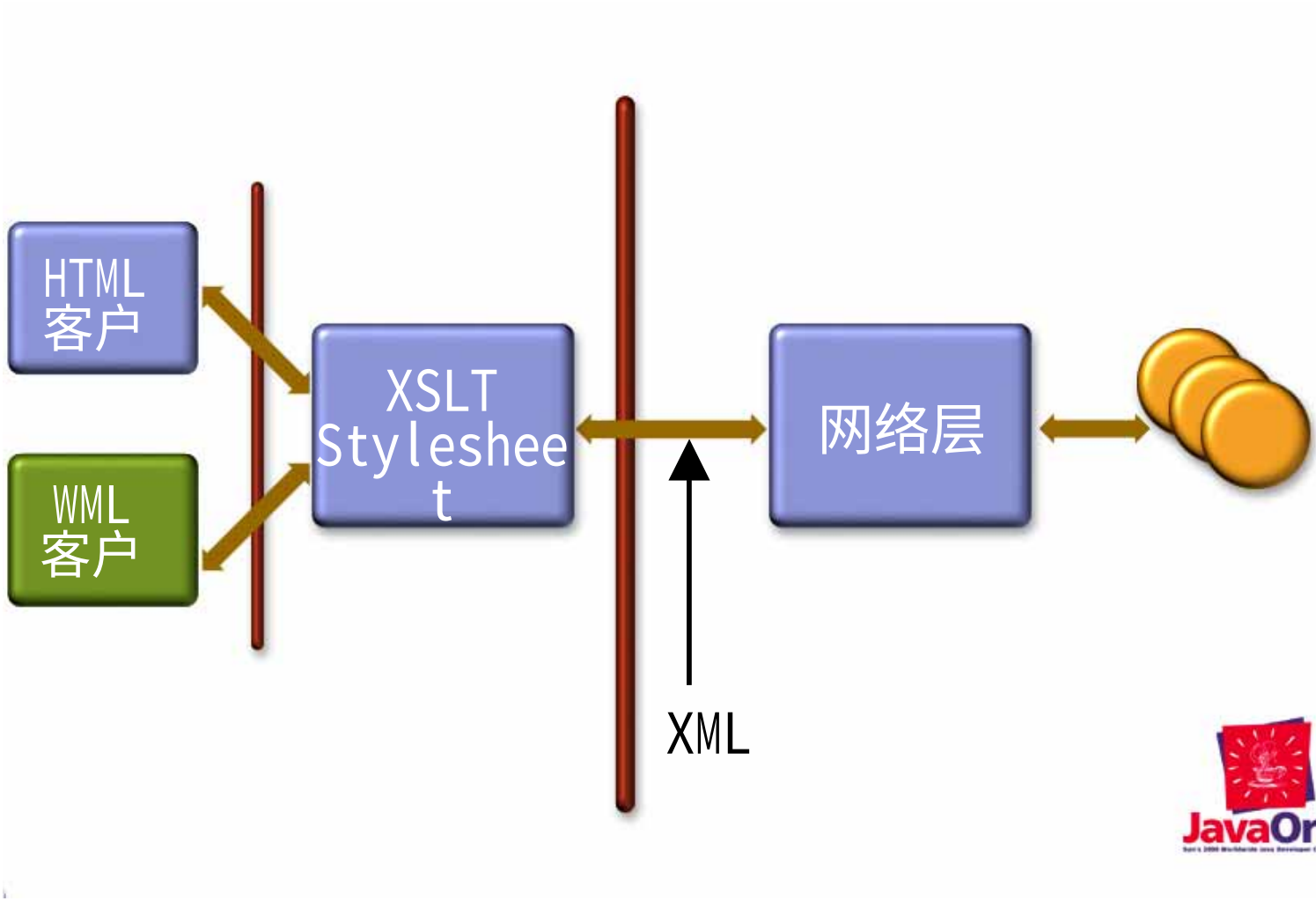
产生XML


JSP 页面
WAP
客户
RIM
Palm V
Palm VII
客户
转换器
XML
<?xml >
...
<y:inject Data
/>
<some tag>
...
</some tag>
<x:doXSL />

多客户译码器



多客户XSLT Stylesheet



用于XML注册的Java API --- JAXR
Java API for XML Registries




- 注册是B2B协作中的一个第三方工具
- JAXR是用于XML注册的统一的应用编程接口
  - ebXML，UDDI等等.

基于RPC方式处理XML的Java API
Java API for XML based RPC




- 用途
  - Marshalling和Unmarshalling参数
  - 将基于XML的调用定义映射为Java接口、类和方法，或者进行反向映射
- 将会成为W3C的XML协议(XP)

JDOM

- JDOM设计体系
  - 隐藏了XML的复杂性
  - 利用了Java 2语言的强大功能
  - 利用了方法过载、Collections APIs、 Reflection、弱引用
  - 提供类型转换

JDOM与DOM的区别就在于代表Document(文档)、Elements(元素)和Attributes(属性)的JDOM的类是模块化的，更像传统的JAVA类

包	说明
org.jdom	DOM的JDOM实现
org.jdom.adapters	处理XML解析器的JDOM适配器
org.jdom.input	内含使用DOM或SAX创建文档的类
org.jdom.output	内含向流发送DOM树或创建SAX2事件的类

内容回顾

- XML基础
- Java+XML简介
- Java API for XML
  - JAXP (Processing API)
  - JAXB (Project Adelard)
  - JAXM (Messaging API - ebXML)
  - XML和Java 2 平台，企业版

总结

- Java + XML代表了可移植的数据和行为
- Java+XML关注
  - 标准体
  - 通过JCP的Java APIs
  - 鼓励标准的实现
  - 利用已存在的平台—Java 2，J2EE，JSP等等

参考资料

- http://www.w3.org/XML/
w3c的xml站点

- http://www.xml.org/
xml站点

- http://java.sun.com/xml
sun公司的xml站点

- http://www.huihoo.com
国内一个关于中间件的专业站点

结束

谢谢大家！

Allen@huihoo.com

http://www.huihoo.com