**J2EE Review**

# JAVAEE 概述

Java EE是sun公司推出的企业级[应用程序](http://baike.baidu.com/view/330120.htm)版本和标准，这个版本以前称为 J2EE，在Java SE的基础上构建，是一个让企业开发大幅缩短投放市场的时间的体系结构

能够帮助开发和部署可移植、健壮、可伸缩、安全、可以分布式的服务器端 Java 应用程序。

提供 Web 服务、[组件模型](http://baike.baidu.com/view/3852154.htm)、管理和通信 API，可以用来实现企业级的面向服务体系结构和 Web 2.0 应用程序。

## 1.1什么是企业应用？

企业应用程序提供了一个企业的业务逻辑

他们是集中管理的，经常与其他企业软件交互；在信息技术领域的应用，企业必须设计，建造，并产生更少的钱，以更高的速度，和更少的资源。

## 1.2什么是J2EE应用模型？

Java EE应用模型开始于Java编程语言和Java虚拟机。

可移植性，安全性和开发效率形成了应用模型的基础。

## 1.3J2EE是分布式多层应用，包括哪些层？不同的层上分别运行什么组件

**J2EE可以分为4层:**

客户层：运行在客户端机器上的客户端组件

Web层：运行在J2EE服务器上的Web层组件.

业务层：运行在j2EE服务器上的业务逻辑层组件.

企业信息系统层：运行在EIS服务器上的企业信息系统层软件.

1. **客户端组件——客户端组件**

Applets：采用java创建的基于html的程序

应用客户端组件：比网页标记语言提供更丰富的UI，让用户能充分控制任务的执行，

例如GUI（swing），AWT（abstract window toolkit）

Web客户端组件(瘦客户端)：动态网页（html,xml）; 浏览器

Javabean组件：在客户端和服务器之间或server和数据库之间管理数据流

1. **Web层组件——服务器端组件**

Java servlet, JSF(javaserver faces), JSP, web层组件可以直接和数据库交互，也可以通过业务层来和数据库交互

1. **业务层组件**

**EJB——**企业bean，完成业务逻辑处理（会话bean，消息驱动bean，实体bean），EJB负责和EIS（企业信息系统层），数据库交**互**

1. **企业信息系统层（EIS）**

负责EIS，包括企业基础设施系统，例如企业资源计划（ERP——enterprise resource planning (ERP)），大型主机事务处理，数据库系统，遗留信息系统**（举例）**

## 1.4客户端有哪2类，分别运行什么组件？

客户端分为：Web客户端；应用程序客户端

Web客户端组件：动态网页（html,xml）; 浏览器

应用客户端组件：GUI（swing），AWT（abstract window toolkit）

## 1.5什么是javaee 组件，它和标准java类的区别？组件有哪些（分层表述）？

1. **JavaEE组件**

采用java语言编写，像java类一样编译，将相关类和文件打包成的独立的功能单元，可以和其他组件交互，可以集成部署到服务器当中运行，JAVAEE应用是由组件构成。

1. **和java类的区别**

* 组件按照javaEE的规范被编译成javaEE应用
* 可以发布部署到服务器中运行
* 可以提供安全，事务管理，JNDI寻址，远程连接，生命周期管理，数据库连接操作等功能
* 普通java类按j2se的编译规范编译为.class文件，不能发布部署的服务器（容器）中运行

## 1.6什么是容器？提供哪些底层服务？可配置的服务包括哪些？不可配置包括哪些？容器有哪4类？

**什么是容器？**

容器是支持组件和底层平台功能的接口。（javaee组件要编译成模块，发布到容器中运行，通过配置一些文件就可以提供安全，事务管理，JDNI查找和远程连接的服务）

**J2EE容器提供了哪些底层服务？**

Container Services

Configurable Services

Nonconfigurable Services

**可配置的服务**

安全性——配置用户的访问权限；

事务管理——配置由哪些操作来组成一个事务单元

JNDI查找——提供统一的接口让应用查找服务

远程连接——管理客户端和企业bean（EJB）之间的低层通信，一个EJB被创建，client可以invoke其中的方法就像EJB在client的同一个JVM中一样，代理模式

**不可配置的服务**

EJB

servlet 生命周期

数据库连接资源池

javaee平台API

**容器有哪4类？**

Javaee服务器：

EJB container:管理EJB的执行，EJB容器和ejb组件可以运行在javaEE服务器行

Web container：管理web pages，servlets，一些EJB组件的运行，Web容器和web组件可以运行在javaEE服务器上

应用客户端容器：管理应用客户端组件，应用客户端和它的容器可以运行在应用客户端

Applet容器：管理applets的运行，由同时运行在客户端的web浏览器和java插件组成

## 1.7组件要打包部署到服务器上，打包后形成的文件类型有哪些？什么是部署描述文件？

**打包后的文件类型**

通用的java打包文件.jar文件

Web应用打包成.war文件

企业级应用打包成.ear文件

**部署描述文件有哪些**：描述部署内容，两种类型

**Javaee部署描述符：**定义javaEE规范，配置部署设置

**运行时部署描述符：**配置javaee实现时需要的特殊参数

## 1.8javaee模块有哪些类型？（打包后的扩展名？各个模块的内容）

1. web模块， \*.war   包括servlet类文件，JSP页面文件，支持类文件，GIF和html文件，web应用部署描述符文件（xml形式的配置文件）
2. ejb模块， \*.jar     包括ejb文件，ejb配置文件（ejb部署描述符）
3. application client模块； \*.jar   包括相关类文件，应用客户端配置文件（部署描述符）
4. resource adapter模块   \*.rar    包括所有的java接口，类，本地库，文档，资源适配描述文件（资源部署描述符）

## 1.9Java EE应用可重用模型，哪5类开发角色？

The Java EE product provider 。 J2EE产品供应商

The tool provider 工具供应商

Application Component Provider 应用组件供应者

Application Assembler 应用装配

Applicaion Deployer and Administrator 应用程序开发者和管理员

# Servlet

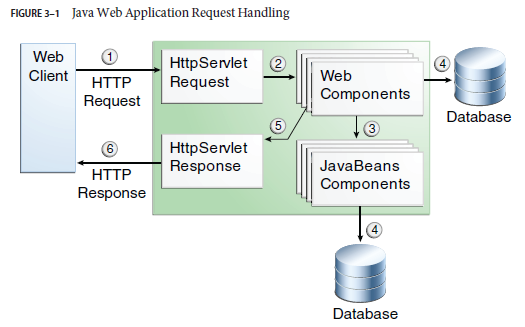
**2.1什么是web应用？**

## 有哪2类的web应用？

Web应用是对web或应用服务器的动态扩展，有两种类型

* **面向表示：**包括用于交互的web pages（html，xhmtl，xml），用于响应用户的动态内容
* **面向服务：**实现了web的端点服务

## 2.2web应用请求处理的过程，6步？



1. client发送HTTP请求给web server
2. 采用servlet和jsp实现的web server转变请求为HTTPServeltRequest object
3. 该object被转发给web组件，web组件和javabean组件或数据库交互产生动态内容
4. Web组件产生HTTPServletResponse，或者把request对象传给另外的web组件处理
5. Web组件生成最终的HTTPServletResponse object
6. Web server将该object转化为HTTPresponse，返回给客户端

## 2.3web容器提供的服务，可以通过哪2种方式进行配置？

提供的服务：请求调度，安全性，并发性，和生命周期管理。也给Web组件提供API，交易，和电子邮件的访问。

1.使用JavaEE注释

2.web应用程序部署描述符

## 2.4web应用包括哪4类内容？开发web应用的过程？

**web应用包含的4类内容？**

web组件，静态资源文件（图像和层叠样式表），辅助类和库

**Web应用开发过程：**

* 开发web组件代码
* 开发web应用部署描述符，有必要的话
* 编译web应用组件和帮助类，帮助类用来引用其他组件
* 打包web应用为一个可部署的单元
* 部署应用到web容器
* 提供一个URL来访问web应用

## 2.5什么是web资源？

**static web content files, such as images, which are called *web resources***

## 2.6web应用会被打包成什么文件？web模块的目录结构？

打包成.war文件

**打包后的web模块的目录结构**

* 最顶层是应用的doucument root：XHTML pages，client-side class and archieves, 静态web资源（images）存放的位置
* Document root包括一个子文件WEB-INF
* WEB-INF包括：
  + classes:一个包括server-side classes的文件夹（包括：servlets,EJB class files,utility class, javabean components）
  + tags：一个包含tag files的文件,tag files用户实现tag libraries
  + lib：一个文件夹，包含必须的jar文件，jar文件中包含ejb，java打包的库文件,叫做server-side classes
  + 部署文件描述符：web应用是web.xml，ejb应用是ejb-jar.xml

## 2.7sevlet的生命周期是什么？

Servlet的生命周期由servlet所部属的容器控制，当一个客户端请求发送到服务器时，容器开始执行以下步骤：

* **如果servlet实例不存在**
* 载入servlet类
* 创建一个servlet的实例，一次只初始化一个servlet实例
* 调用init方法初始化这个实例
* 调用service的方法，传递request和response对象
* 如果容器需要移除这个servlet，那么他就会通过调用servlet的destroy方法来释放这个servlet

## 2.8什么是URL模式？ url包括的组成部分？URL和URI的区别

**URL is a string that identifies a Web component or a static object such as an HTML page or image file.**

**URL**

[**http://[*host*]:[*port*][*request-path*]?[*query-string***](http://[host]:[port][request-path]?[query-string)**]**

* **request-path**

上下文路径：向前的斜线/和servlet的Web应用的上下文根的拼接。

servlet路径：与激活该请求的组件别名相应的路径部分，由向前的斜线/开始。

路径信息：请求路径的部分，不是上下文路径或者servlet路径的部分。

* **查询字符串：**查询的参数

**URL和URI的区别？**

URI 是从虚拟根路径开始的

URL是整个链接

## 2.9Web.xml常见元素的含义?

1. **标题：DOCTYPE声明，告诉服务器适用的servlet规范的版本，指定DTD**

2. **主正文：根元素web-app**

3. **<servlet>…… </servlet>:应包括的Servlet类**

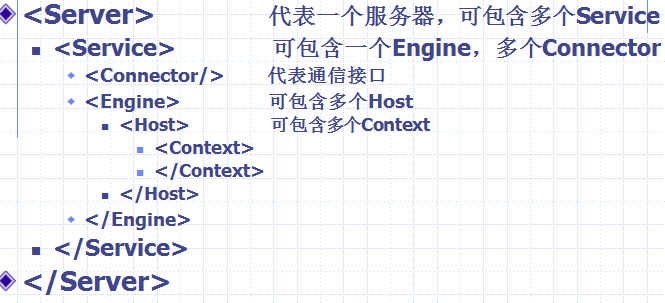
4. **<servlet-mapping>:指定Servlet可以映射到哪种URL模式**

**Filter filter-mapping welcom-file-list display-name listener**

## 2.10如何编写线程安全的类/servlet线程安全?

**servlet默认是多线程的，Server创建一个实例，用它处理并发请求——编写线程安全的类，避免使用可以修改的类变量和实例变量；**

## 2.11tomcat容器文件的组织结构，组件结构



1. tomcat是基于组件的服务器，构成组件可配置在<tomcat\_home>\conf\server.xml
2. Server代表一个服务器,可以包含多个service
3. Service：代表服务，可以包含一个engine，多个connector
4. Connector代表通信接口，在某一个指定端口监听用户请求，并且将获得的请求交给engine来处理
5. Engine可以包含多个host，将获得的请求匹配到某个虚拟主机上，并且吧请求交给该host来处理
6. Host：可包含多个context，代表虚拟主机，每一个都和某个网络域名相匹配，每一个都可部署多个web应用
7. Context对应一个web应用（由一些Servlet，HTML，Java类，JSP页面和一些其他的资源组成，在**创建时根据**<Tomcat\_home>\conf\web.xml获得和<Webapp\_home>/WEB-INF/web.xml载入Servlet类。

在请求时查询映射表找到被请求Servlet类并且执行以获得请求回应

## 2.12tomcat如何处理http请求，简单描述

[**http://localhost:8080/HelloWorld/**](http://localhost:8080/HelloWorld/)

1.请求被发送到本机端口8080，被Java HTTP Connector获得；

2. Connector将该请求交给它所在的Service的Engine来处理，并等待Engine的回应；

3. Engine获得请求，匹配所有虚拟主机；

4. Engine匹配到名为localhost的主机；

5. localhost主机获得请求，匹配所拥有的所有Context；

6. localhost主机匹配到路径为/HelloWorld的Context

7.路径为/HelloWorld的Context获得请求，在映射表中寻找对应的Servlet；

8.Context匹配到URL PATTERN为/的Servlet;

9.构造HttpServletRequest对象和HttpServletResponse对象，作为参数调用该Servlet的Service方法；

10. Context把执行完之后的HttpServletResponse对象返回给localhost主机；

11.Host把HttpServletResponse对象返回给Engine；

12. Engine把HttpServletResponse对象返回给Connector；

13. Connector把HttpServletResponse对象返回给客户Browser

## 2.13web应用跟踪会话（session）有两种机制，一种是cookie，另一种是URL重写，过程分别是什么。

* Web应用采用session来跟踪应用的状态，因为HTTP是无状态的，有需求要维持状态。
* Session被表示为HTTPSession对象，可以将一个object-valued的属性关联到session中，这样属性可以在同一个web应用的不同地方使用；
* 因为cIient端不会发signal不需要一个session了，所以session有timeout机制，过时失效
* Web容器在client和server端传递一个标识符（session id）来维护session的状态，在client端这个标识符可以实现为cookie机制，或者在服务端采用URL重写机制来维护session，将session信息写入URL中。一般选择URL重写机制，因为cookie在客户端可能不启用
* Session实现两种机制：cookie和url重写
* **1、当用户第一次访问站点→创建一个新的会话对象（Httpsession）， Server分配一个唯一的会话标识号(sessionID)；**
  + **Servlet容器自动处理sessionID的分配**
  + **尽可能长，确保安全**
  + **把sessionID信息放到HttpSession对象中**
* **2、Server创建一个暂时的HTTP cookie**
  + **cookie存储这个sessionID（名:jsessionid）**
  + **Server将cookie添加到HTTP响应中**
  + **Cookie被放置到客户机浏览器中，存储到客户机硬盘**
* **客户浏览器发送包含Cookie的请求；**
* **4、根据客户机浏览器发送的sessionID信息（cookie），Server找到相应的HttpSession对象，跟踪会话**
* **5、在会话超时间隔期间，如果没有接收到新的请求， Server将删除此会话对象**
  + **用户又访问该站点，必须重新注册，确保安全**
* **Cookie被客户禁用时，采用URL重写机制：**
  + **调用reponse.encodeURL(*URL*)方法；**
  + **http://…;jsessionid=….**
* **1、5与Cookie机制相同**
* **2、Server将sessionID放在返回给客户端的URL中；**
* **3、客户浏览器发送的请求将包含sessionID；**
* **4、根据包含请求的sessionID信息（URL），Server找到相应的HttpSession对象，跟踪会话**

## 2.14cookie和session使用场景

* **Cookie**

1. 跟踪会话，也可以独立于http会话使用cookie
2. 长期“记住用户信息”
3. 存储在客户机本地计算机硬盘上

示例：在购物车系统中，使用cookie记录用户id，预填充；使用会话，跟踪登录状态，跟踪应用程序的使用情况，cookie.txt文件，记录用户对语言和颜色的选择之类的偏好

* **Session**

1. 保存在服务器端内存中
2. 使用机制不同

示例：在购物车系统中，跟踪用户的购物车，导航信息，登录状态

## 2.15web组件共享信息有哪四种方法

web组件共享信息方法：

1. private helper objects :javabean,可以在public函数的区域共享信息
2. request，response绑定属性：

四种作用域对象共享信息——application,page,request,session

1. 数据库共享信息：数据库绑定属性
2. 其他web资源

## 2.16servlet有哪四种作用域对象

页面域（page scope）

请求域（request scope）

会话域（Session scope）

应用域（Application scope）

## 2.17共享信息会带来并发访问的情况，并发访问的场景有哪些（至少三种）

* **产生并发的场景**
  + 多个web组件访问web context（web上下文）中的对象
  + 多个web组件访问session中的对象
  + Web组件多线程访问实例变量（可以采用实现SingleThreadModel interface来避免）

## 2.18什么是过滤器？过滤器和其他web组件的区别有什么？过滤器使用场景

1. **过滤器**

是一个object，可以修改request或者response的header和context

在不修改servlet代码的情况下向servlet添加功能；如：身份认证

可用于跨多个servlet执行一些功能，创建可重复使用的功能

在servlet处理请求之前，截获请求 如：在调用servlet之前，截获请求，验证用户身份，未经授权的用户遭到拒绝，而servlet不知道曾经有过这样的请求

具体使用场景：代码重用；应用安全策略；日志；为特定目标浏览器传输XML输出；图像转换、加密；动态压缩输出；解决请求和响应中中文乱码的情况

## 2.19什么是监听器，场景？

1. **监听器**

是一个对象，监听servlet的生命周期，而做一些操作；可以监听webcontext，才初始化时做一些操作；监听session；监听request

# JSP

## 3.1JSP与servlet的区别？

JSP技术可以容易的创建web context（静态+动态），sevlet技术在jsp中可用，也提供了更加自然的方式创建静态内容；特性：

* 一种开发JSP pages的语言，基于本文描述如何请求和响应
* 提供表达式语言，访问server-side的对象
* 定义了jsp语言上的扩展

## 3.2JSP页面的生命周期？

JSP的生命周期和许多JSP页面的能力（特别是动态方面）一样是由Java Servlet技术测定。

当一个请求映射到一个JSP页面，Web容器首先检查是否JSP网页的servlet比JSP页面的老。如果servlet是老年人，Web容器将JSP页面到servlet类和编辑类。在开发过程中，在JSP页面servlet的一个优点是，生成过程是自动执行。

**解析阶段：Servlet容器解析JSP文件代码，如果有语法错误，就会向客户端返回错误信息**

**翻译阶段：Servelet容器把JSP文件翻译成Servelet源文件**

**编译阶段：Servelet容器编译Servelet源文件，生成servelet累**

**初始化阶段：加载与JSP对应的Servelet类，创建其实例，并调用它的初始化方法**

**运行时阶段：调用与JSP对应的Servelet实例的服务方法**

**销毁阶段：调用与JSP对应的Servelet实例的销毁方法，然后销毁Servelet实例**

## 3.3jsp包含哪两种类型的文本？

* **静态数据：HTML，SVG（可伸缩矢量图），WML（无线标记语言），XML**
* **动态数据：**jsp 元素（标准jsp语法，xml语法），如java代码，指令标签等

## 3.4jsp元素包括哪些类型？

* **指令元素**

<%@page ...%>;

<%@include ...%>;

<%@taglib ...%>;

* **脚本元素：声明（Declaration）、表达式（Expression）、脚本程序（Scriptlet）**

<%...%>

<%=…%>

<%!...%>： <%! int foo=3; %>声明只在当前页面可用，声明变量和方法

* **行为元素**

jsp:[set|get] Property;

jsp:[include|forward]

jsp:plugin

jsp:param

jsp:useBean

## 3.5这些类型是如何被处理，即翻译成servlet

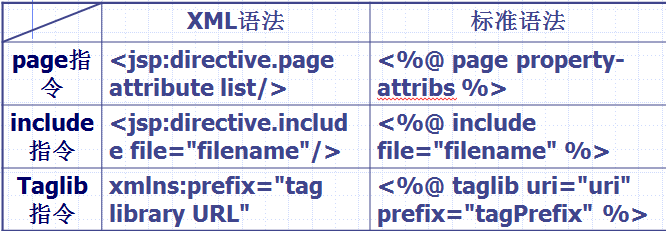
1.**静态数据：**转化为代码，排放到响应流中的数据。

2.**jsp元素**

* **指令标签**：是用来控制Web容器转换并执行JSP页面；
* **脚本元素：**插入到JSP页面的servlet类中；
* **表达式语言**：作为参数传递来调用JSP表达式解释器
* **jsp:[set|get]Property：**被转换成方法来调用JavaBeans组件。
* **jsp:[include|forward]**： 被转换成Java Servlet API的调用，即servlet采用dispatcher的方式include和forward的方式。
* **jsp:plugin**被转换成浏览器的特定标记来激活一个applet
* **Custom tags自定义标签：**转化为调用标记处理程序，该程序实现自定义标签调用
  + **JSP elements are treated as follows:**
    - **Directives are used to control how the web container translates and executes the JSP page.**
    - **Scripting elements are inserted into the JSP page’s servlet class.**
    - **Expression language expressions are passed as parameters to calls to the JSP expression evaluator.**
    - **jsp:[set|get]Property elements are converted into method calls to JavaBeans components.**
    - **jsp:[include|forward]elements are converted into invocations of the Java Servlet API.**
    - **The jsp:plugin element is converted into browser-specific markup for activating an applet.**
    - **Custom tags are converted into calls to the tag handler that implements the custom tag.**

## 3.6常见的JSP指令？

* **指令标签**



Page指令（session，import，extends，contentType，buffer，ThreadSafe，errorPage）；

例如：<% @ page import= " package.\* " %>；

<% @ page contentType= "text/html,charset=GBK" %>

include指令：<%@ include file="filename" %>，静态包含，不能有重复内容

taglib指令: <%@ taglib prefix="tlt" uri="/tlt"%> uri指明标签

## 3.7include指令和include动作的区别

**include指令**：包括其他页面，编译时把其他页面的内容加进来，比include动作快；静态包含，不能有重复内容。

**include标准动作**：使用RequestDispatcher，运行时把其他页面的内容加进来（包括到输出流中），调用servlet API的include，不是静态包含，是运行时包含，是被包含的jsp执行完毕后，包含到本页面，被包含页面中有与本页面重复内容不影响，可以包含静态和动态内容

## 3.8jsp脚本元素创建和使用对象的方式

**三种创建和使用脚本元素对象的方法：**

1. 类实例和类变量 在声明中创建，在脚本和表达式中被使用

<%! int foo=3; %>

< %!

private int inc(int x){

return x++;

}

%>声明方法

1. 局部变量 在脚本和表达式中被创建和使用

<%

foo=int(foo);

% >

1. 作用域对象的属性 在脚本和表达式中被创建和使用：四种域对象：application，page，request，session



## 3.9jsp隐式对象，常见的有哪些

application,session,request,respose,out,pageContext

不常见 page config exception

## 3.10常见的JSP标准动作？forward动作与http重定向的区别？

* **动作标签**
  + <jsp:include>:调用servlet API的include，不是静态包含，是运行时包含，是被包含的jsp执行完毕后，包含到本页面，被包含页面中有与本页面重复内容不影响，可以包含静态和动态内容
  + <jsp:forward>
  + <jsp:plugin>
  + <jsp:useBean>
  + <jsp:setProperty>
  + <jsp:getProperty>
  + **<jsp:param name= " itemid " value= " %=itemId" />**

**例如：参数只在被访问的页面有效**

**<jsp:include page= " /itemdetail.jsp" >**

**<jsp:param name= " itemid " value= " %=itemId" />**

**</jsp: include >**

**forward动作与http重定向的区别**

转发是服务器行为，重定向是客户端行为。

转发过程：客户浏览器发送http请求——》web服务器接受此请求——》调用内部的一个方法在容器内部完成请求处理和转发动作——》将目标资源发送给客户；在这里，转发的路径必须是同一个web容器下的url，其不能转向到其他的web路径上去，中间传递的是自己的容器内的request。在客户浏览器路径栏显示的仍然是其第一次访问的路径，也就是说客户是感觉不到服务器做了转发的。转发行为是浏览器只做了一次访问请求。

重定向过程：客户浏览器发送http请求——》web服务器接受后发送302状态码响应及对应新的location给客户浏览器——》客户浏览器发现是302响应，则自动再发送一个新的http请求，请求url是新的location地址——》服务器根据此请求寻找资源并发送给客户。在这里location可以重定向到任意URL，既然是浏览器重新发出了请求，则就没有什么request传递的概念了。在客户浏览器路径栏显示的是其重定向的路径，客户可以观察到地址的变化的。重定向行为是浏览器做了至少两次的访问请求的。

## 3.11javabean组件设计的规范

javabean是java的class，是符合一定规范的value object，是可以复用和组合使用的组件，所有java的class只要符合javabean规范，就是一个javabean，jsp支持javabean的创建和初始化，设置属性。

**规范：**

**1. 属性Read/write, read-only, or write-only**

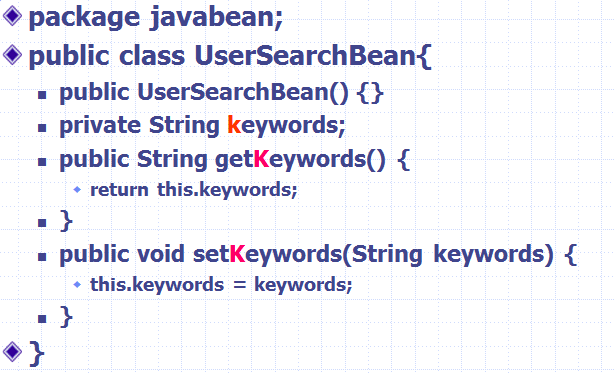
**2. 属性可以简单的单值也可以复合（数组）**

**3. 每一个属性有get和set方法，可以设值是可读，可写，或同时可读/可写**

**4. 命名规则：属性第一个字母小写，Set/Get第一个字母大写**

**5. 要位于一个package中**

**6. 要有一个无参数的构造器**



## 3.12javabean动作标签中usebean执行过程，

1. <jsp:useBean id="beanName" class="fully-qualified-classname" scope="scope"/>

Scope可以是application,session,request,page

1. <jsp:useBean id="beanName" class="fully-qualified-classname" scope="scope">

<jsp:setProperty .../>（initializing bean properties）

</jsp:useBean>

用usebean标签声明要使用javabean，并指明作用域，如果这个bean不存在，则statement创建一个bean，存在scope中，id决定了bean的名称，class决定了具体的class

容器将自动创建bean实例，容器自动处理javabean的清理事项

## 3.13使用定制标签的作用

自定义标签，可以消除jsp页面的冗余标签，

* **在简单的JSP标签后面隐藏复杂的功能**
* **在一定程度上实现了模块化**
* **JSP程序员把程序的基本功能用自定义的标签库来实现；美工人员使用这些标签，专注于数据的表达**
* **实现了重用性**
* **将复杂的功能封装在HTML风格的标签中**

# JDBC

## JDBC API包含哪两方面内容？

**由一组类和接口定义的方法构成，为Java开发人员提供了一个行业标准API，提供了数据库的调用层接口**

**The JDBC API has two parts: an application-level interface used by the application components to access a database, and a service provider interface to attach a JDBC driver to the Java EE platform.**

**译：**应用组件用来连接数据库的应用层接口和连接上Java EE平台的JDBC驱动程序的服务提供商接口

## 两种建立数据库连接的方式区别？为什么推荐DataSource

DriverManager机制和DataSource机制：向数据库提交查询请求，读取查询结果，处理结果，释放连接。

DriverManager机制：

* **1.注册驱动程序**
  + **隐式注册：加载数据库驱动程序类（把驱动加载到内存中），自动向DriverManager注册**
    - **Class.forName(“*JDBCDriverName*”);**
  + **显示注册：**
    - **DriverManager.registerDriver (new *JDBCDriverName*());**
* **2. 建立数据库连接：**
  + **Connection con=DriverManager.getConnection(*URL*,*username*,*password*);**
  + **//按照注册顺序，找到第一个可以成功连接到给定URL的驱动程序，返回一个Connection对象**
  + **JDBC URL的语法：**
    - **jdbc:driver:databasename**
* **3. 使用连接进行查询、插入、删除的操作**

DriverManager弊端：

* + 是一个同步的类，一次只有一个线程可以运行
  + 与数据库相关的连接信息都包含在类中，如果用户更换另一台计算机作数据库服务器，就需要重新修改URL变量、重新编译、部署；
  + 用户的用户名、口令也包含在类中，丧失了安全性

**DataSource机制——JNDI**

1. **JNDI：注册到JNDI，使用JNDI服务向程序隐藏了登录细节**

JNDI：Java命名和目录接口(Java Naming and Directory Interface)，为开发人员提供了查找和访问各种命名和目录服务的通用、统一的方式。（中央注册中心，储存了各种对象、用户和应用的变量及其值，开发大型的分布式应用，使分布式的Java程序找到分布式的对象）

**分布式应用程序**：通过RMI或CORBA向JNDI注册对象，其他任何客户机上的应用程序只需知道数据源对象在服务器JNDI中的逻辑名称，就可以通过RMI向服务器查询数据源，然后与数据库建立连接

1. **连接池**

Application Server启动时，创建通向数据库资源的连接池。连接池包含多个JDBC连接。

当应用程序需要访问数据库时，从连接池中取得一个连接，使用该连接与数据库通信；一旦工作完成，关闭数据库连接，释放回连接池中，重用共享数据库连接，最好的连接方式

**DataSource机制:**

* **DataSource是JDBC Connection对象的一个工厂**
* **允许使用已经在JNDI命名服务中注册的DataSource对象建立连接，由驱动程序供应商实现**

## 事务： Java事务类型有哪三种？

JDBC事务；JTA（Java Transaction API）事务，容器事务

## java事务有几种类型，转账事务可以有哪些实现（选看）

事务：标准一系列数据库操作能够完成，特性：原子性，一致性，隔离性，持久性

转账事务的实现方式：JDBC事务,JTA事务，容器事务

* **JDBC事务**
* 用Connection对象控制
* 提交事务两种模式：自动提交和手工提交
* 使用 JDBC 事务界定时，可以将多个 SQL 语句结合到一个事务中。
* 缺点:事务的范围局限于一个数据库连接。一个 JDBC 事务不能跨越多个数据库
* **JTA（Java Transaction API）事务**
* 允许应用程序执行分布式事务处理–在两个或多个网络计算机资源上访问并且更新数据，这些数据可以分布在多个数据库上。提供事务的默认自动提交来控制事务的提交和回滚
* 两阶段提交:事务管理器和资源管理器之间使用的协议是XA；
* XA：资源和事务管理器之间的标准化接口，参与JTA事务，XA连接不支持JDBC自动提交功能
* **容器事务**
* 是J2EE应用服务器提供的，容器事务大多是基于JTA完成。
* 与编码实现JTA事务管理相比，可以通过EJB容器提供的容器事务管理机制（CMT）完成同一个功能。
* 可以简单的指定将哪个方法 加入事务，一旦指定，容器将负责事务管理任务。

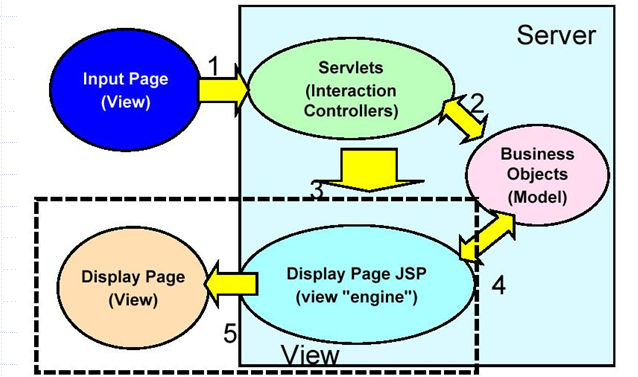
# MVC

## MVC在web应用中的控制流程（6步）

View层：Browser（HTML tags;JSP tags XML/XSL）用户界面

Controller层:Servlet，接受用户动作，并对数据做适当处理

Model层:Bean/JavaBeans(InventorManager,InventorItem,ShipmentReceived)封装应用数据



1. client端（web浏览器）发出请求
2. servlets获取客户请求
3. servlets决策由哪一个web组件来处理请求（javabean，EJB，或者其他对象）
4. javabean或者EJB处理来自servlets的业务请求，封装结果
5. servlet选择一种表示模板（JSP），将内容返回给client
6. jsp根据结果中的javabean产生具体的jsp页面，返回给client端；jsp页面不创建对象，只是从javabean中获取内容，对象都由servlet创建

# EJB

## 什么是企业bean

**an enterprise bean is a server-side component that encapsulates the business logic of an application**

企业bean是一个服务器端组件,封装了应用程序的业务逻辑

## EJB(企业bean)有哪三类？

1. **Session bean：**执行客户端请求的任务，可以用web server实现

封装业务逻辑，代表为一个用户执行的操作，可以被本、远程client，webservice调用。访问分布式的server端应用，client调用session bean的方法，session bean屏蔽了服务器端内部业务逻辑的细节。Session bean 不是持久化的，不保存到数据库中

1. **Message-driven bean**：监听某种类型的消息， 例如Java Message Service API（JMS）

* 让javaee应用能处理异步消息，JMS消息或其他消息
* 是一种JMS消息监听器，接受jms消息，类似于事件的监听器
* JMS消息可以由javaee应用组件发出（应用client，另外的ejb，web组件），JMS应用发出，或者非java技术的应用发出
* Sessionbean 主要是同步接收或发出JMS消息，但是不能异步；当需要降低同步消息带来的进程阻塞时，用 Message-dreiven bean，异步消息传输
* Message-driven bean不通过接口来访问bean
* 无状态：可以处理多个client的消息
* 单线程：一个message-driven bean一次只能处理一个消息
* Message-driven bean：class要是pulic的，添加**@MessageDriven**注解，class不能是abstract或final，要有无参数构造器，不能定义final方法，最好实现message listener接口

3.实体bean

## 什么是远程方法调用？

## 其中的会话bean三种类型是什么？(对比sping)

* **Stateful（有状态session bean）；**
* 有状态的会话 bean在方法调用时可保持对话状态，**维持实例变量的状态**；
* 不同的client有各自的session bean，**不共享**。Client端请求session bean，系统就给该client创建一个session bean实例，并维持组件的状态
* client端关闭，session bean失效；要指定容器在某个方法完成后删除有状态的会话bean实例，只要为该方法添加注释@Remove
* **什么时候使用：a）**client跨越多个方法调用需要维持相关信息 **b）**client需要和其他应用组件交互 **c)** 该session bean需要多个EJB交互才能实现，需要维护一个工作流

例如：购物车

* **stateless（无状态会话bean）；**
* 不维持会话状态，不跟踪记录从一个方法调用传递到另一个方法调用的信息，每次调用无状态的业务方法都独立于前一次调用。
* 一个bean可以被多个client共享，client可以从容器的stateless session bean的实例池中获取实例
* 可以实现web service，但是stateful session bean不行。
* 容器管理stateless session bean更简单
* 指定Java Bean作为无状态的会话bean加以部署及管理，只需要为该bean添加注释@Stateless。
* 例如：发email
* **singleton（单例bean）**

应用的生命周期中，只会被实例化一次。

可以被所有客户共享，并发访问

可以实现为web serveive端点

用途：应用的初始化工作，关闭时的清理工作，singleton在应用的整个生命周期存在

## 企业bean的客户端，有哪两种方式获得bean实例？这两种方式的区别？

* **依赖注入**：client端有ejb的实例引用，采用java语言注解的方式；ejb和client在同一个jvm中。最简单的获取ejb引用的方式。

Jsp，web application，JAX-RS web service, 其他的EJB， java ee应用客户端，都支持依赖注入，需要javax.ejb.EJBannotation包

@EJB(“beanName”)

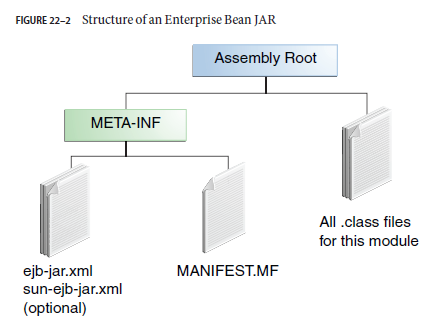
* **JNDI查找**：ejb和client在不同的jvm或者同一个jvm中，通过JNDI来查找ejb实例。

Client端可以是简单的J2SE应用，JNDI支持识别java ee组件的全局语法，简化了显示查找的语法

## 企业bean的客户端有三种类型，是哪三种？（会话bean允许哪些client访问？）

* **远程romote**
  + Client和ejb在不同的jvm上，client可以是web组件，应用client，另外的ejb
  + 对于远程client，**Ejb的位置透明**
  + 必须**通过business interface** **访问ejb，不能no-interface-view访问：采用JNDI查找**
  + 客户端通过remote接口调用ejb：会用到RMI的stub和skeleton，网络，参数整理功能，生成bean很慢，效率不高
* **本地 local**
  + Client和EJB在同一个jvm，client可以是web组件或其他的ejb，
  + **ejb的位置不透明**
  + **采用no-interface view的方式**访问ejb，依赖注入或JNDI查找都行，不用使用new操作符来创建EJB实例
  + **采用business interface view的方式访问ejb：**依赖注入或JNDI
  + 快速高效的访问，没有使用stub、skeleton代理，以更快的方式生成bean
* **服务 web service**

## EJB打包后的模块目录结构如何？



* Enterprise bean class

实现业务方法的的ejb，生命周期回调方法

* Business interface

定义业务接口方法

当ejb是本地，且no-interface-view访问时，就不需要了

* Helper class

Ejb的辅助类，例如：exception 或者 utility class

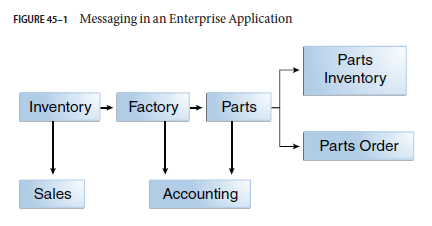
## 使用消息服务（JMS）的应用场景？

JMS技术：Java Message Service；智能的交换机，用于路由分布式应用中的应用程序组件和进程中的消息；异步传递消息

**应用场景：**

**进程间通信**

* + web前端为顾客订货录入系统
  + 仓库管理系统：接受订单、配送货品、把订单转交发货系统
  + 发货系统：更新顾客的账号纪录，开始发货

****

各个组件之间可以发送消息来通信

## JMS的哪两种消息域？点对点，发布订阅模式应用场景？

* **点对点**
* JMS把消息传递给一个消息消费者，

例如：web前端发送包含订货信息的消息，仓库管理系统接收这个消息，处理订货，即使有多个仓库管理系统，也不会同时处理一个订单

* 异步通信：不必等到所有的处理工作都完成，成功将消息插入队列后可回复处理
* 按照发送的顺序把消息写入并保存到队列中；消息消费者处理队列中的信息
* **发布/订阅**
* 把消息发送给一个主题（Topic），每个主题有多个订阅者，由JMS把消息的副本传递给主题的每个订阅者
* 当客户购买过几次之后，一个消息就会发送给一个“常客”主题
* 站点为这些常客发送一个“特惠待遇”消息

## 消费者可以同步，可以异步，区别  ？

消费者处理消息

* **同步消息**
* 发布者或者订阅者调用receive方法，显式地从目的端获取消息
* receive方法在消息到达之前会阻塞，或者设置一个timeout在时限内没有到达，就失效返回null
* **异步消息**
* 在消息消费者端注册消息监听器，类似于事件监听器
* 消息到达，JMS provider调用监听器的onMessage方法来转发消息，该方法处理消息的内容
* 在消息到达前可以做其他事情

消息驱动bean属于哪种？

# 对象关系映射（java持久性API/Hibernate）

## 实体bean的设计规范?

实体是轻量的持久化对象，映射关系数据库表的字段，每个实例就是表的一行

持久数据组件——内存中的对象，对应到数据库中的一个视图；一种持久性的、事务性的以及可以共享的组件，多个客户机可以同时使用其中的业务数据

例如：顾客、订单、产品、信用卡

1. 添加对象关系映射注解

@Entity

@Table(name=”tbl\_user”)

1. 要有public或者protected，无参数的构造器
2. Class不能声明没final，不能有final类型的持久化实例变量或方法
3. 如果对象会被传递，例如通过romote business interface传递，要实现

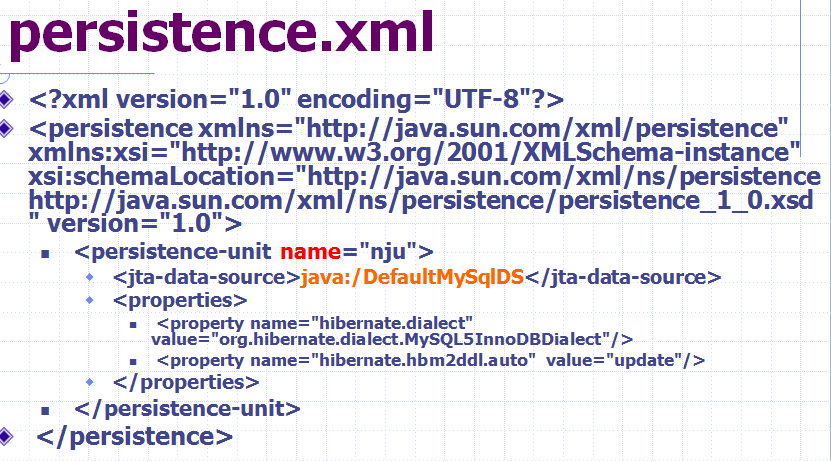
Serializable interface

1. 持久化实例变量要是private，protected，或者paskage-private,通过entity的业务方法或get/set访问器访问，要设置entity的get/set方法
2. Entity可能扩展entity class 和non-entity class， non-entity class可能扩展entity class
3. 采用注解方式来映射持久化实例变量,可以注解实例变量，或者javabean风格的getter方法
4. 用@Id注解单个primary key；符合主键要定义在primary key class中，采用javax.persistence.EmbeddedId and javax.persistence.IdClass annotations 标明

## bean的四种状态？

## 配置文件主要内容有哪些？

定义持久化单元



Persistence-unit:可以有一个或多个，定义持久化内容名称，使用的数据源名称和Hibernate属性，name属性设置持久化名称

Jta-data-source:指定entity bean使用的数据源名称，哪一个数据库；”java:/”前缀不能少，数据源名称大小写敏感

Properties:指定 Hibernate的各项属性

# 安全服务（了解）

## javaEE的安全模式可以通过哪2种方式实现？

* **实现机制**

Javaee安全服务由组件容器来提供，

可以采用声明和编程技术来实现

1. **声明安全**

采用部署描述符文件或者注解的方式实现

部署描述符文件在应用的外部，包括了安全角色，访问需求的定义，安全角色和需求被映射成具体环境的安全角色、用户和策略

1. **编程技术实现**

绑定在应用内部，用来做出安全决策

适用于当声明安全的实现形式不足以充分表达应用所需的安全模型的情况

## 安全性包括哪些内容？

1. **用户和组**

用户——应用程序终端用户的账户名（ID）

组——命名的用户集合，包含0个和多个用户；通常用户表示具有类似系统资源访问权限的应用用户，例如：雇员（企业组），管理员（雇员组子集，可以访问敏感的工资数据）

1. **认证和授权**

认证（Authentication）：用户向系统证明“我是谁”

授权（Authorization）：应用服务器授予某个用户访问哪些资源的权限，“我能访问什么样的服务”

1. **角色和策略**

**角色(role)**：一种抽象的逻辑用户分组；代表相同资源访问权限的用户组或者特定用户；

在部署时，角色被映射为授权的用户或组

**策略**：回答“特定角色能够访问什么样的服务”的问题

**应用服务器**：使用角色和策略为请求者访问资源提供授权，如

具有客户服务角色的成员在8:00 AM到5:00 PM间访问web应用程序

1. **审计和日志记录**
   1. 收集、存储和分发整个系统中安全事件信息
   2. 查看执行的活动；
   3. 帮助检测和调查应用程序环境中的潜在弱点
2. **防火墙**
   1. 禁止任何不需要的协议或客户类型访问应用程序
3. **数据保密和安全套接字**

安全套接字层（Secure Sockets Layer, SSL）：通过在网络传输之前对数据加密，保证数据的机密性

SSL结合了几种加密技术：数字证书、标准加密（对称密钥加密）、公钥加密

## 8.3用户认证有哪四步？

步骤1。应用程序开发人员编写代码来提示用户输入用户名和密码。

步骤2。应用程序开发人员讨论如何通过使用一个部署描述符来为部署的应用程序设置安全。

步骤3。服务器管理员在应用程序服务器设置授权用户和组。

步骤4。应用程序部署比对应用程序和定义在应用服务器上的安全角色的用户，组，和策略。