J2EE Review

1. JAVAEE 概述

Java EE 是 sun 公司推出的企业级应用程序版本和标准，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、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

2. **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
3. **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**
4. **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。
2. **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

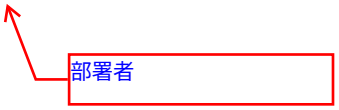
Javaee 部署描述符：定义 javaEE 规范，配置部署设置
运行时部署描述符：配置 javaee 实现时需要的特殊参数

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



2. Servlet

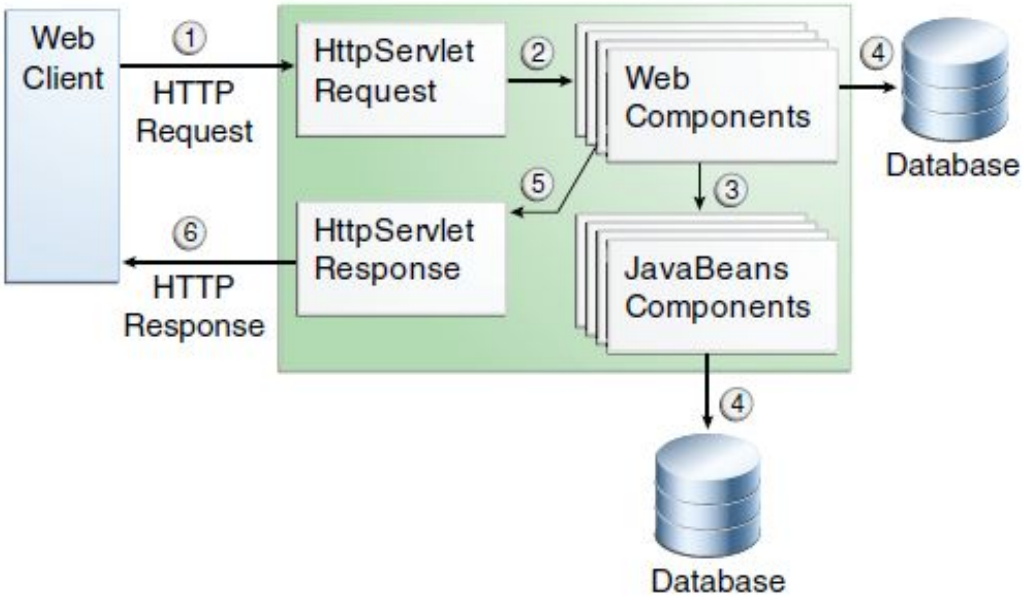
2.1 什么是 web 应用？有哪 2 类的 web 应用？

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？

FIGURE 3–1 Java Web Application Request Handling



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*]

- ✓ **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**

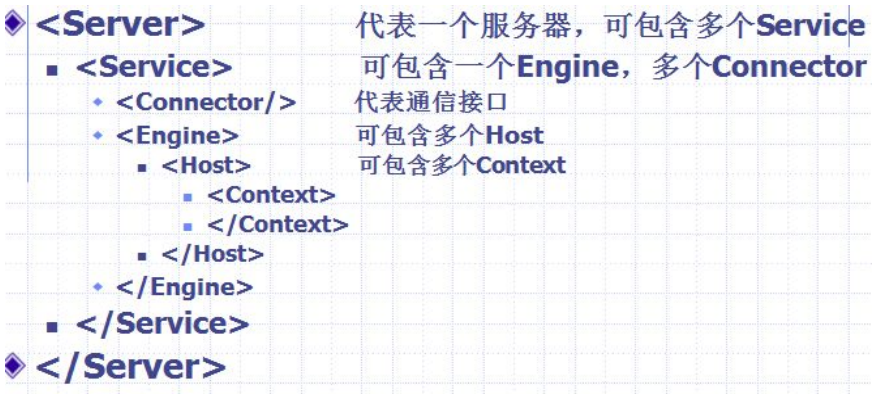
Filter filter-mapping welcom-file-list display-name listener

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；

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/

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 client 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
3. 数据库共享信息：数据库绑定属性
4. 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？

2） **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

3. 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**

	XML语法	标准语法
page指令	<jsp:directive.page attribute list/>	<%@ page property-attribs %>
include指令	<jsp:directive.include file="filename"/>	<%@ include file="filename" %>
Taglib指令	xmlns:prefix="tag library URL"	<%@ taglib uri="uri" prefix="tagPrefix" %>

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

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

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

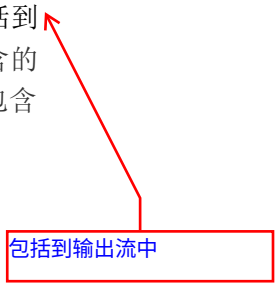
include 指令：<%@ include file="filename" %>，静态包含，不能有重复内容

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
```

2) 局部变量 在脚本和表达式中被创建和使用

```
<%
foo=int(foo);
% >
```

3) 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

request	请求对象	类型 javax.servlet.ServletRequest	作用域 Request
response	响应对象	类型 javax.servlet.SrvletResponse	作用域 Page
pageContext	页面上下文对象	类型 javax.servlet.jsp.PageContext	作用域 Page
session	会话对象	类型 javax.servlet.http.HttpSession	作用域 Session
application	应用程序对象	类型 javax.servlet.ServletContext	作用域 Application
out	输出对象	类型 javax.servlet.jsp.JspWriter	作用域 Page
config	配置对象	类型 javax.servlet.ServletConfig	作用域 Page
page	页面对象	类型 javax.lang.Object	作用域 Page
exception	例外对象	类型 javax.lang.Throwable	作用域 page

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>

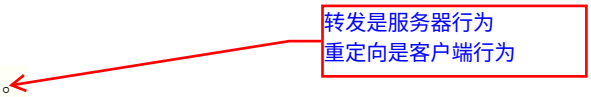
- <jsp:getProperty>
- **<jsp:param name= " itemid " value= " %=itemId" />**

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。

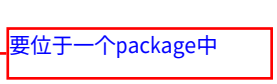
重定向过程：客户浏览器发送 http 请求——》web 服务器接受后发送 302 状态码响应及对应新的 location 给客户浏览器——》客户浏览器发现是 302 响应，则自动再发送一个新的 http 请求，请求 url 是新的 location 地址——》服务器根据此请求寻找资源并发送给客户。在这里 location 可以重定向到任意 URL，既然是浏览器重新发出了请求，则就没有什么 request 传递的概念了。在客户浏览器路径栏显示的是其重定向的路径，客户可以观察到地址的变化的。重定向行为是浏览器做了至少两次的访问请求的。

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**



```
package javabean;
public class UserSearchBean{
    public UserSearchBean() {}
    private String keywords;
    public String getKeywords() {
        return this.keywords;
    }
    public void setKeywords(String keywords) {
        this.keywords = keywords;
    }
}
```

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
2. <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

模块化

美工专注数据表达

重用 标签多处使用

4. JDBC

4.1. 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

4.2.两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**

注册驱动程序

建立连接

执行操作

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

重新编译

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

2. 连接池

通过RMI或者CORBA向JNDI注册对象

JNDI和连接池协作

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**

4.3. 事务： Java 事务类型有哪三种？

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

4.4. java 事务有几种类型，转账事务可以有哪些实现（选看）

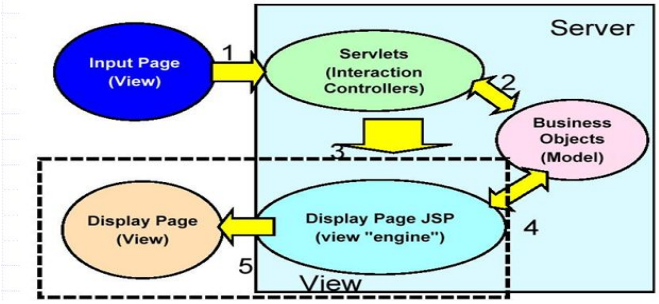
事务：标准一系列数据库操作能够完成，特性：原子性，一致性，隔离性，持久性
转账事务的实现方式：JDBC 事务,JTA 事务，容器事务

- **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。

5. MVC

5.1. 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

jsp不创建对象

6. EJB

6.1. 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

服务器端的组件，封装了应用程序的业务逻辑

6.2. 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
2. **Message-driven bean**：监听某种类型的消息， 例如 Java Message Service API（JMS）

session bean不是持久化的

监听器 实现异步处理消息

- ✓ 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

无状态

单线程

3.实体 bean

什么是远程方法调用？

6.3. 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 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

和其他应用组件交互和多个EJB交互有什么区别

- ✓ **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

6.4. 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？

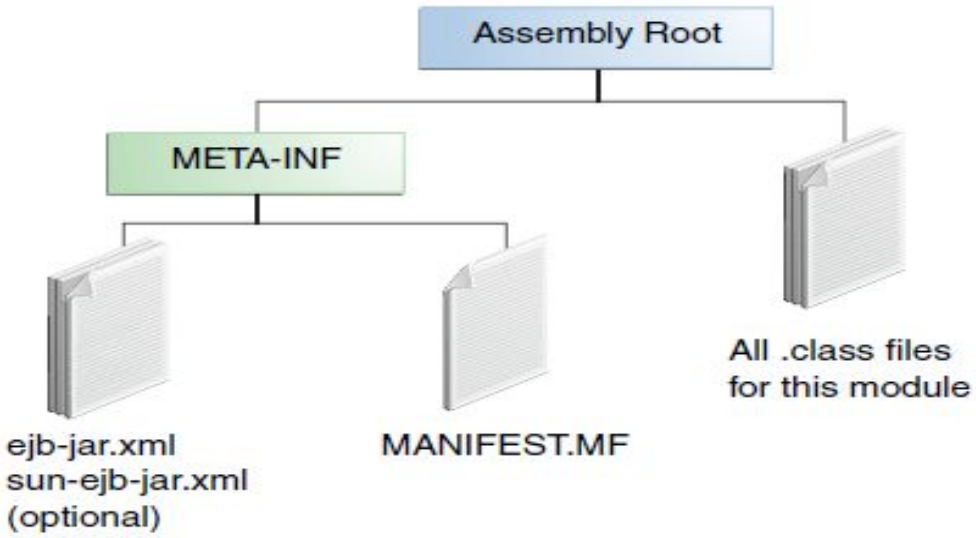
- **依赖注入**：client 端有 ejb 的实例引用，采用 java 语言注解的方式；ejb 和 client 在同一个 jvm 中。最简单的获取 ejb 引用的方式。
  Jsp，web application，JAX-RS web service，其他的 EJB， java ee 应用客户端，都支持依赖注入，需要 javax.ejb.EJBannotation 包
  @EJB("beanName")
- **JNDI 查找**：ejb 和 client 在不同的 jvm 或者同一个 jvm 中，通过 JNDI 来查找 ejb 实例。Client 端可以是简单的 J2SE 应用，JNDI 支持识别 java ee 组件的全局语法，简化了显示查找的语法

6.5. 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**

6.6. EJB 打包后的模块目录结构如何？

FIGURE 22–2 Structure of an Enterprise Bean JAR



- 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

6.7. 使用消息服务（JMS）的应用场景？

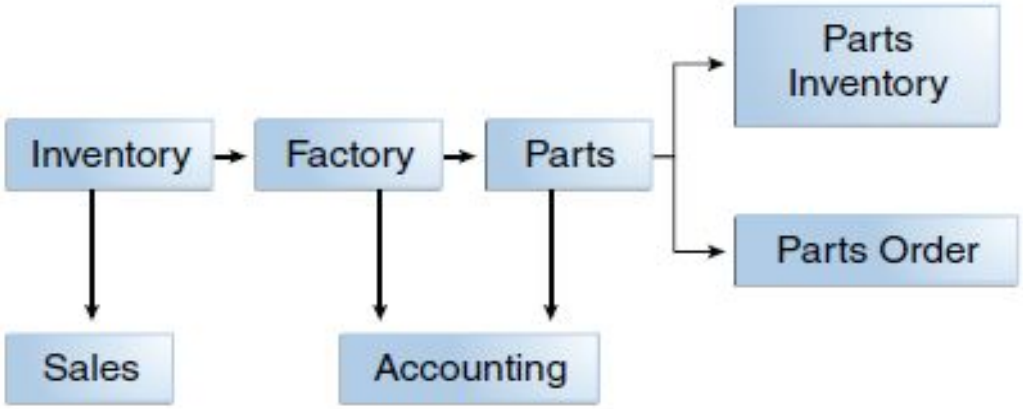
JMS 技术：Java Message Service：智能的交换机，用于路由分布式应用中的应用程序组件和进程中的消息；异步传递消息

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

FIGURE 45–1 Messaging in an Enterprise Application



各个组件之间可以发送消息来通信

6.8. 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

6.9. 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

在消费者端注册消息监听器

调用监听器的onMessage方法来转发消息

✓ 在消息到达前可以做其他事情

消息驱动 bean 属于哪种？

7. 对象关系映射（java 持久性 API/Hibernate）

7.1. 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”)
2. 要有 public 或者 protected，无参数的构造器
3. Class 不能声明没 final，不能有 final 类型的持久化实例变量或方法
4. 如果对象会被传递，例如通过 romote business interface 传递，要实现 Serializable interface
5. 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
6. Entity 可能扩展 entity class 和 non-entity class， non-entity class 可能扩展 entity class
7. 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
8. 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

添加对象关系映射

构造器

没有final

序列化接口

ge/set

啥

可用注解方式映射持久化变量

ID

bean 的四种状态？

7.2. 配置文件主要内容有哪些？

定义持久化单元

persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd
" version="1.0">
  <persistence-unit name="nju">
    <jta-data-source>java:/DefaultMySqlDS</jta-data-source>
    <properties>
      <property name="hibernate.dialect"
        value="org.hibernate.dialect.MySQL5InnoDBDialect"/>
      <property name="hibernate.hbm2ddl.auto" value="update"/>
    </properties>
  </persistence-unit>
</persistence>
```

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

8. 安全服务（了解）

8.1. 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

  2) **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

8.2. 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）

2) **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”

3) **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

4) **审计和日志记录**

a) 收集、存储和分发整个系统中安全事件信息

b) 查看执行的活动；

c) 帮助检测和调查应用程序环境中的潜在弱点

5) **防火墙**

a) 禁止任何不需要的协议或客户类型访问应用程序

6) **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。