填空 15
名词解释 25（关键词）
简答 30
问答 30
答题标准参照 PPT
没复习的就不考了

第一章

概述
简单说说 J2EE 有哪些东西

企业应用（JEE 中 什么是企业应用） 和他相对应的——web 应用

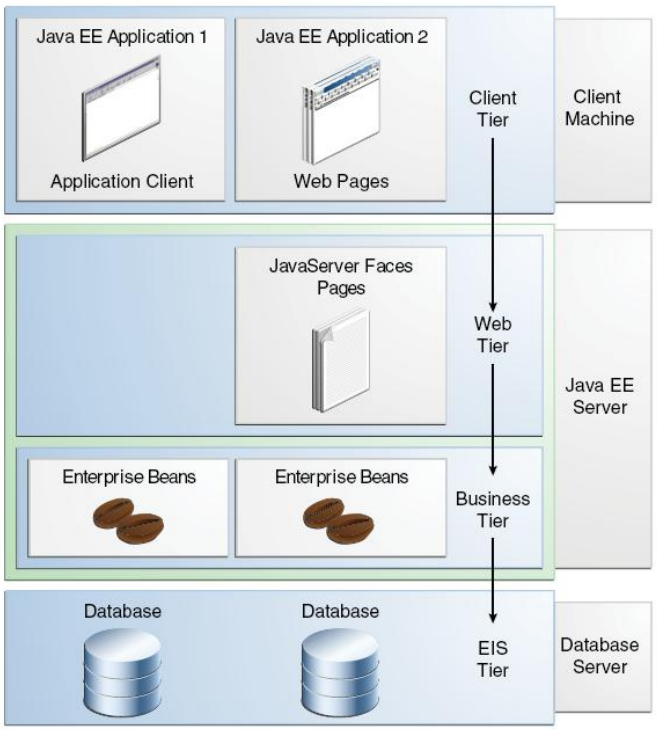
企业应用为企业提供业务逻辑，它是分布式的、有事务处理的部分、便携式的，且在安全、性能和可靠性上有优势。与 web 应用的最大区别在于企业应用的数据量更大，因此所使用的技术也不一样。

java EE 概念：分布式的多层应用模型（四层：客户端 WEB EJB EIS）

Java EE 平台使用分布式多层应用模型来开发企业应用。

应用逻辑根据功能分为不同的组件，应用组件构成 Java EE 应用，根据组件在 Java EE 环境中所处的层将其安装在不同机器上。

Figure 1–1 Multitiered Applications



分布式：EJB 这层可采用分布式技术

分布式多层应用模型都有哪些层，层有哪些组件

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——应用客户端和 applets【小程序】

Web 层：运行在 J2EE 服务器上的 Web 层组件——Java Servlet, JavaServer Faces, and JavaServer Pages (JSP) (might include a JavaBeans component to manage the user input and send that input to enterprise beans running in the business tier for processing)

业务层：运行在 j2EE 服务器上的业务逻辑层组件——Enterprise JavaBeans (EJB)

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——enterprise infrastructure systems

Java 组件和标准 java 类的区别

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，不能发布部署到服务器（容器）中运行

两类客户端

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）

Web 层 orEJB 层都是由容器来提供底层服务——什么是容器（javaee 中的狭义概念），容器提供了哪些服务

1、容器是组件和底层支持该组件的特定平台功能的接口。

2、提供的服务

1）可配置服务 2）不可配置服务

可配置服务

1）Java EE **安全**模块允许用户配置 web 组件或企业 bean 使系统资源仅能被授权用户访问。

2）Java EE **事务处理**模块允许用户指定构成一个事务的方法间的关系，从而一个事务中的所有方法可当作一个单元。

3）**JNDI 查询服务**为企业中的多个命名和目录服务提供一个统一接口，从而应用组件可以访问这些服务。

4）Java EE **远程连接**模型管理客户端和企业 beans 之间的底层通信。在企业 beans 创建后，客户端调用它的方法就像它们在同一虚拟机上。

不可配置服务

1）企业 bean 和 servlet 的生命周期。

2）数据库连接池。

3）数据持久化。

4）对 Java EE 平台 APIs 的访问。

Web 容器提供了哪些服务（EJB 容器和 web 容器不同）

A web container provides such services as request dispatching, security, concurrency, and lifecycle management. 请求分派、安全（可配置）、并发（多线程管理）、生命周期管理（不可配置）
A web container also gives web components access to such APIs as naming, transactions, and email.对 Java EE 平台 APIs 的访问。【不可配置】

在这些服务中哪些是可配置的，哪些是不可配置的（会去用到，但是不可修改的）

不同的容器上面分别管理哪些组件（都是跟层的概念相关）

1）**Java EE 服务器**：Java EE 产品的运行时部分。提供 EJB 和 web 容器。

•**EJB 容器**:为 Java EE 应用程序管理企业 beans 的运行。企业 beans 和它们的容器运行在 Java EE 服务器上。

•**Web 容器**：为 Java EE 应用程序管理 web 网页，servlets 和一些 EJB 组件的运行。web 组件和它们的容器运行在 Java EE 服务器上。

2）**应用程序客户端容器**：管理应用程序客户端组件的运行。应用程序客户端和它的容器运

行在客户端。

3）**Applet 容器**：管理 applets 的运行。包含运行在客户端的 web 浏览器和 Java 插件。

开发好之后，需要进行集成和部署（打包 发送给别的人 部署）

打包之后可以有三种类型（比如常用 web 应用打包类型 war；EJB 打包类型）

Java Archive (JAR) file,
Web Archive (WAR) file
Enterprise Archive (EAR) file.

打包时不仅要编写组件，还要写 xml 文件（部署描述文件：一类是规范中要求的[比如 web 项目的 web.xml]，另一类是厂商需要的（厂商提供的额外服务——不属于标准））

包含的内容

1）一个功能性组件或类似企业 bean，web 页面，servlet，applet 的组件。

2）一个可选的部署描述文件，用来描述单元的内容。

部署描述文件

1）部署描述文件有两种类型：Java EE 和运行时

•Java EE 部署描述文件：由 Java EE 规格说明定义，可用于配置任何兼容 Java EE 实现的部署信息。

•运行时部署描述文件：用于配置 Java EE 特定实现的参数。

打包还有另外一个概念：java ee 模块，一共四类，除了组件和部署描述文件之外，还有哪些文件（每类应用分别包括哪些文件 比如图片 引用的其他 jar 包）

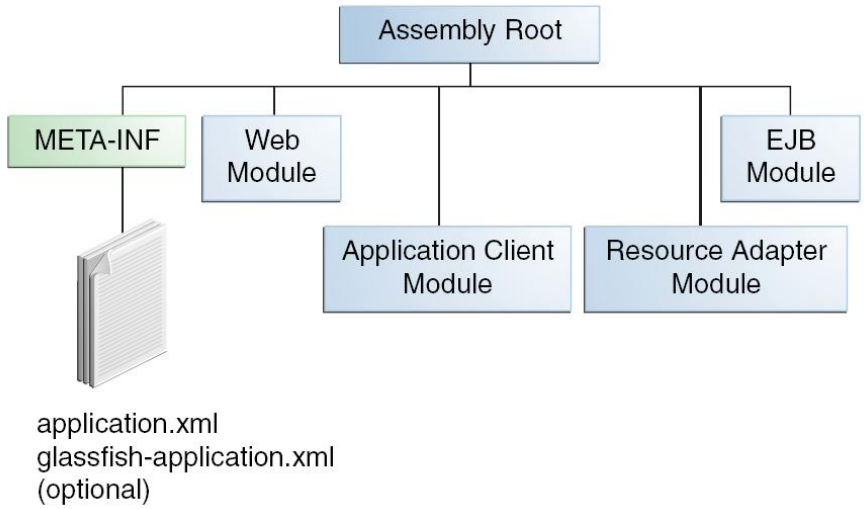
1）**EJB 模块**：包含企业 beans 的类文件和 EJB 部署描述文件——*.jar。

2）**Web 模块**：包含 servlet 类文件，web 文件，辅助类文件，GIF 和 HTML 文件，web 应用程序部署描述文件——*.war。

3）**应用程序客户端模块**：包含类文件和应用程序客户端部署描述文件——*.jar。

4）**资源适配器模块**：包含所有 Java 接口，类，原生类库，其他文档和资源适配器部署描述文件——*.rar。

Figure 5–1 EAR File Structure

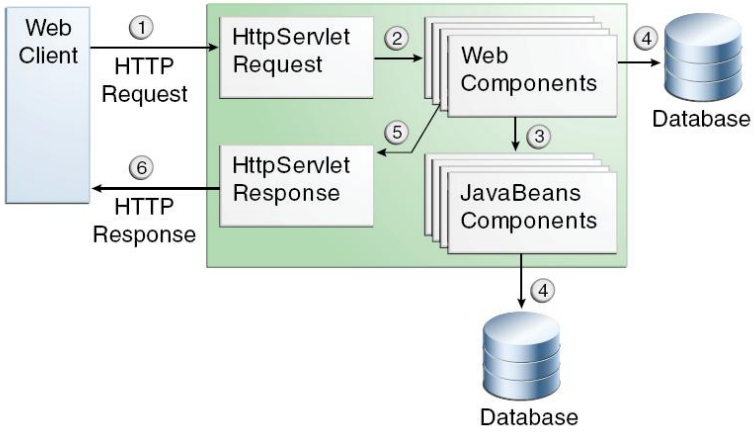


第二章

Web 层 web 应用
Web 应用有两种 我们是面向表示的（通过浏览器交互），还有一类是面向服务的
Web 应用上的 web 组件两种：servlet 和 JSP
原理是相同的，因为 JSP 要翻译成 servlet 再被执行

在 javaee 中 web 应用请求处理的过程是什么

Figure 6–1 Java Web Application Request Handling



1）客户端发送一个 HTTP 请求到 web 服务器。

2）实现了 Java Servlet 和 JSP 技术的 web 服务器将请求转换为 HTTPServletRequest 对象。

3）这个对象被交给 web 组件，该组件可以和 JavaBeans 组件或数据库交互从而生成动态内容。

4）然后 web 组件可能生成一个 HTTPServletResponse，也可能将此请求转发给另一个 web 组件。

5）web 组件最后生成一个 HTTPServletResponse 对象。

6）web 服务器将这个对象转换成一个 HTTP response，然后将它返回给客户端。

开发 web 应用的过程是什么（打包要的文件 就是开发要的那些）

1）编写 web 组件代码。

2）如果需要，编写 web 应用部署描述文件。

3）编译 web 应用程序组件以及组件用到的辅助类。

4）可选地，将应用程序打包到可部署单元。

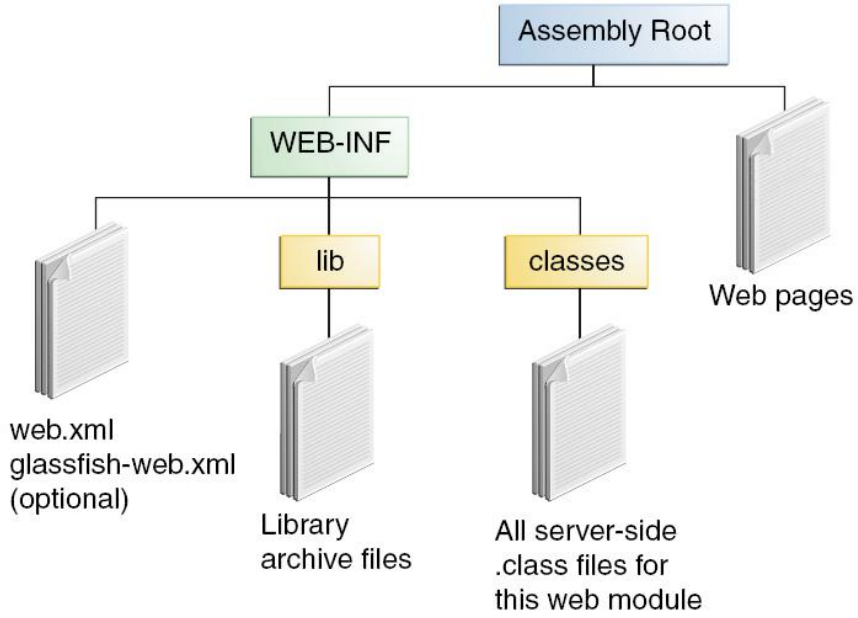
5）将应用部署到 web 容器。

6）访问引用该 web 应用的 URL。

Web 模块包含哪些内容（每个模块包含哪些内容，规范要求必须符合什么目录结构：什么放在哪里）

Web 组件+web 资源

Figure 5–3 Web Module Structure



1）web 模块有特定的结构。

2）web 模块的顶层目录是应用的主目录。

•主目录是 XHTML 网页、客户端类和档案、静态 web 资源（如图像）存储的地方。

3）主目录包含名叫“WEB-INF”的子目录。

4）还可以在主目录或 WEB-INF/classes/目录创建特定于应用的子目录。

Webinfo 包含子目录和文件（有哪些。知道结构）

WEB-INF contains the following files and directories:

- classes: A directory that contains server-side classes: servlets, enterprise bean class files, utility classes, and JavaBeans components
- tags: A directory that contains tag files, which are implementations of tag libraries
- lib: A directory that contains JAR files that contain enterprise beans, and JAR archives of libraries called by server-side classes
- Deployment descriptors, such as web.xml (the web application deployment descriptor) and ejb-jar.xml (an EJB deployment descriptor)

开发后由 web 容器提供服务（提供哪些服务）

A web container provides such services as request dispatching, security, concurrency, and lifecycle management. 请求分派、安全（可配置）、并发（多线程管理）、生命周期管理（不可配置）

A web container also gives web components access to such APIs as naming, transactions, and email.对 Java EE 平台 APIs 的访问。【不可配置】

Servlet

生命周期是怎样：什么时候创建，销毁，方法调用

1.servlet 的生命周期由部署 servlet 的容器控制，分为三个阶段：

1）初始化阶段：调用 init()方法，在 Servlet 的整个生命周期内，init()方法只被调用一次。

2）响应客户请求阶段：调用 service()方法

3）终止阶段：调用 destroy()方法

2.当请求被映射到 servlet，容器会执行以下步骤：

①如果不存在 servlet 的实例：

•载入 servlet 类

•创建 servlet 类的一个实例

•通过调用 init 方法初始化 servlet 实例

②调用 service 方法，传递请求和响应对象

3）如果容器需要移除 servlet，通过调用 destroy 方法来结束 servlet。

访问一个 servlet 的时候要访问 URL，——http 请求的 URL 包含哪些部分

组成

1）HTTP 请求 URL 包含如下内容：**http://[host]:[port][request-path]?[query-string]**

请求路径

1）request 路径由以下元素构成：

•Context path：“/”+ web 应用的上下文根目录。

•Servlet path：“/”+ 激活这一请求的组件别名。

•Path info：属于 request 路径但不属于 context path 或 servlet path 的部分。

查询字符串

1）查询字符串由参数及其值组成。

2）个人参数可通过 geParameter 方法从 request 中获得。

3）生成查询字符串的两种方法：

•查询字符串可明确的显示在 web 页面。

•当带 GET 方法的表单被提交，查询字符串可添加到 URL。

什么是上下文路径 context

web 应用的上下文根目录 web 应用的名字（根路径）

什么是 servlet 路径

Web 组件的名字（可为别名）

URI 包含哪些部分（区分 URL）

request URI：context path+servlet path+path info

定义

1）URI：uniform resource identifier，统一资源标识符，用来唯一的标识一个资源。

2）URL：uniform resource locator，统一资源定位器，是一种具体的 URI，即 URL 可以用来标识一个资源，而且还指明了如何 locate 这个资源。

区别

1）在 Java 的 URI 中，一个 URI 实例可以代表绝对的，也可以是相对的，只要它符合 URI 的语法规则。而 URL 类则不仅符合语义，还包含了定位该资源的信息，因此它不能是相对的，schema 必须被指定。

2）如 URL：http://localhost:8080/tradeload/TestServlet

·ContextPath：/tradeload

·ServletPath：/TestServlet———URL-Pattern

·PathInfo：null

·对应的 URI：/tradeload/TestServlet

Web 容器 tomcat（组件式的容器，包含哪些组件？每个组件的作用？监听、创建连接。。）

3.各组件功能：

1）**Server：**代表一个服务器，只有一个。

2）**Connector：**在某个指定端口上侦听客户请求，并将获得的请求交给 Engine 来处理。

•Java HTTP Connector 在端口 8080 侦听来自客户 Browser 的 http 请求。

•AJP 1.3 Connector 在端口 8009 侦听来自其他 Web Server(Apache)的 JSP/Servlet 代理请求。

3）**Engine：**将获得的请求匹配到某个虚拟主机(Virtual Host)上，并把请求交给该 Host 来处理。

4）**Host：**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(对应不同的 Context)。

5）**Context：**对应一个 Web 应用（由一些 Servlet、HTML 页面、Java 类、JSP 页面和一些其他的资源构成）。

•在创建时根据<Tomcat_home>\conf\web.xml 和<Webapp_home>/WEB-INF/web.xml 载入 Servlet 类。

•在获得请求时，查询映射表，找到被请求的 Servlet 类，并执行以获得请求回应。

http 请求的 URL 发到 tomcat 后，tomcat 如何处理（哪个组件先做什么，再反向）

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。

在编程中用到的细节：

servlet 是多线程的，如何处理线程安全问题

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；
修改 servlet 类，实现 SingleThreadMode1 接口——单线程

Web 容器提供 session 跟踪的机制有两种，分别是怎么做到的（cookie[id 绑定到此]；URL 重写[id 写在 URL 中]?）

会话实现

~~1）隐藏域的使用：在 HTML 表单域中插入隐藏字段，包含客户状态数据。~~

2）Cookie 的使用。

3）URL 的重写：使用 URL 包含客户状态数据。

Cookie 实现

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。

3）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。

6）用户又访问该站点，必须重新登录，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

会话 session 用在哪些场景中？

使用场景及比较

1）存取方式：Cookie 只能存储 ASCII 字符，且不能直接存取 Java 对象，**Session 可存取任何类型的数据，所以复杂数据用 session。（与用户绑定的一次会话的状态信息）**

2）隐私安全：cookie 存储在客户端浏览器上，对客户端可见，可被客户端复制、修改，不安全，session 存储在服务器上，对客户端是透明，更安全。若非要使用 cookie，可将 cookie 信息加密，提交到服务器后再解密。

3）有效期：**cookie 可长久储存信息**，session 不能长久储存信息，如果超时时间设置过长，会导致服务器内存溢出。

4）对服务器的负担：因存储位置不同，所以若并发访问用户很多，应该用 cookie。

5）浏览器支持 浏览器可禁用 cookie。session 只能在本浏览器窗口及其子窗口有效，而 cookie 除此外还可以设为所有浏览器窗口都有效。

6）跨域名：cookie 支持跨域名访问，session 不支持。

具体使用场景：

>>>session

•跟踪用户的购物车

•导航信息，登录状态

>>> cookie

•用户登录 ID

•用户对语言和颜色的选择之类的偏好

•跟踪应用程序的使用情况

•cookie.txt 文件

单独用 cookie 的场景？

长期“记住用户信息”

作用域对象（web 组件共享信息的方法之一）

TABLE 4–3 Scope Objects

Scope Object	Class	Accessible From
Web context	javax.servlet.ServletContext	Web components within a web context. See "Accessing the Web Context" on page 124.
Session	javax.servlet.http.HttpSession	Web components handling a request that belongs to the session. See "Maintaining Client State" on page 125.
Request	subtype of javax.servlet.ServletRequest	Web components handling the request.
Page	javax.servlet.jsp.JspContext	The JSP page that creates the object. See "Using Implicit Objects" on page 145.

- Web Context：作用域为应用程序运行期，工程启动后存在，当容器关闭时被销毁；
- Session：作用域为会话期，从打开一个浏览器窗口开始，关闭窗口，会话关闭，当会话超时，被销毁；
- Request：作用域为用户请求期，只要 Server 向客户端输出内容，就被销毁；
- Page：作用域为页面执行期。

（比如 session）：有四种（最大：应用[启动服务-开始 停掉-结束]，最小：当前页面）分别指的是什么，什么时候会结束

过滤器与 web 组件的区别（与 servlet 的区别）：

1）过滤器是能够对 Servlet 容器的请求和响应对象进行检查和修改的对象。

2）Servlet 过滤器本身并不生成请求和响应对象，只是提供过滤功能。

3）Servlet 可过滤的 web 组件包括 Servlet，JSP 和 HTML 文件等。

不能互换。主要体现在最后在什么地方用

监听器能监听哪些事件（应用启动？Session？还是请求？。。。）能够监听哪些生命周期事件

1）负责监控 ServletContext、HttpSession、ServletRequest 对象的生命周期事件及属性改变事件，并对此作出处理的对象。

2）主要有三类：监听 Servlet 上下文，监听 HTTP 会话，监听客户端请求。

~~Xml~~ 文件：—使用注解和使用部署描述文件的区别是什么

不能完全替代

- 注解往往是类级别的，因此，XML 配置则可以表现得更加灵活
- 在应用中，往往需要同时使用注解配置和 XML 配
  - 对于类级别且不会发生变动的配置可以优先考虑注解配置
  - 而对于那些第三方类以及容易发生调整的配置则应优先考虑使用 XML 配置

JSP

用更好表达的方式

什么是 jsp 页面，能干什么，包含什么？（两类：静态，动态（称为 jsp 元素））

两部分内容

1）静态数据：可用任何基于文本的格式如 HTML,WML,XML 表示。

2）JSP 元素：决定页面如何构造动态内容。

Jsp 的声明周期（由容器提供管理），与 servlet 的区别——有一个翻译的过程（jsp 是怎样被翻译的）

1）JSP 页面服务可作为 servlet 来响应请求，因此 JSP 页面的生命周期和许多功能（尤其是动方面）由 Java Servlet 技术决定。

2）当一个请求被映射到 JSP 页面，web 容器先检查该 JSP 页面的 servlet 是否比 JSP 页面早，如果早于 JSP 页面，web 容器就将该 JSP 页面翻译成 servlet 类并编译它。在开发阶段，JSP

页面的优势之一是它的构建流程是自动执行的。

3）当页面被翻译并编译后，JSP 页面的 servlet 遵照以下的生命周期：

•如果 JSP 页面的 servlet 不存在实例变量，则容器：

•载入 JSP 页面的 servlet 类。

•实例化出一个 servlet 类的实例。

•调用 jspInit 方法初始化这个实例。

•容器调用_jspService 方法，传递请求和响应对象。

•如果容器需要移除 JSP 页面的 servlet，就调用 jspDestroy 方法。

哪些 jsp 元素，每种如何被翻译？

分类知道如何翻译
1、jsp 指令（常用的）
页面指令（要点：有很多属性，常用的由 session，还有 import 什么的。这些属性每一个如何被翻译）和 servlet 一一对应
2、还有 include 包含指令（包含什么 怎么翻译）
3、脚本（[所有的 servlet 都可以用脚本来描述]三种创建和使用对象的方法：声明中创建的对象；表达式；小脚本 创建的是什么样的对象。线程安全？，这些创建的对象对应的放到 service 的什么方法中去）

提供的元素

1）脚本元素：声明、表达式、scriptlet：

2）指令元素：page 指令、 include 指令、tablib 指令：

3）动作元素：

<jsp:include>,<jsp:param>,<jsp:forward>,<jsp:useBean>,<jsp:getProperty>,<jsp:setProperty>,<jsp:plugin>

翻译

1）web 容器负责对 JSP 页面的翻译。

2）在翻译阶段，对 JSP 页面的每种数据类型有不同的处理方式：

•静态数据：被翻译成代码，并被传入响应流。

•JSP 元素被如下处理：

•**指令（Directives）**：被用来控制 web 容器如何翻译并执行 JSP 页面。

•**脚本元素**：被插入 JSP 页面的 servlet 类中。

•**表达式语言**：表达式被当做参数传到 JSP 表达式求值程序。

•**jsp:[set|get]Property 元素**：被转换成对 JavaBeans 组件的方法调用。

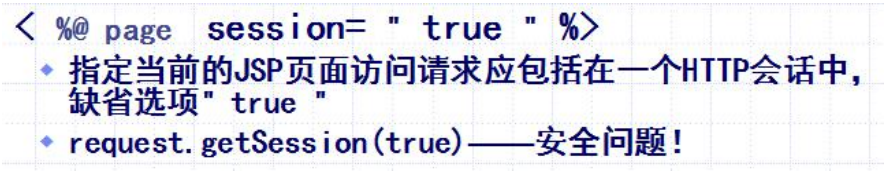
•**jsp:[include|forward]元素**：被转换为对 Java Servlet API 的调用。

•**The jsp:plugin 元素**：被转换为激活 applet 的特定浏览器标记。

•**Custom tags**：被转换为对实现定制标签的标签处理器的调用。

3）翻译和编译阶段都可能会产生错误，这些错误仅在页面第一次被请求时可观测到。

4）如果在任一阶段遇到错误，服务器会返回 JasperException 和一条包含该 JSP 页面名字及错误行号的消息。



脚本：放在 JSP 对应的 service 方法中
表达式：<%= %>把表达式装换成 string 对象并插入到隐式输出流中

隐式对象主要有哪些：比如 session out。。。

- Out 作用域是当前页面
- request/response 对象
- session 对象：javax.servlet.http.HttpSession 的实例
- application 对象：从 web.xml 获取初始化参数、访问 RequestDispatcher

- pageContext 对象：对页面作用域的属性的访问

不用额外写代码创建

动作：

标准动作（常见的有哪些？Include 动作[包含] 怎么翻译？和前面 include 指令区别在于翻译不同；forward 动作[提交]怎么翻译）

1）include 指令和 include 动作都能实现将外部文档包含到 JSP 文档中的功能。

2）语法

•include 指令：<%@ include file="Relative Url"%>（如果内容是静态的，则用此更快）

•jsp:include 动作：<jsp:include page="Relative path to resource" flush="true">

3）区别

·根本性区别在于它们被调用的时间。jsp:include 动作在请求期间被激活**（在动态执行过程中，使用请求分派器去包含）**，而 include 指令在页面转换期间被激活。**（把内容加进来在编译——静态）**

·使用 include 指令的页面要比使用 jsp:include 动作的页面难于维护。

·使用 JSP 指令，如果包含的 JSP 页面发生变化，那么用到这个页面的所有页面都需要手动更新，这就需要记住所有包含该页面的其他页面，或者重新编译所有的页面，以使更改能够生效。

·jsp:include 在每次请求时重新把资源包含进来。在实现文件包含上，应该尽可能地使用 jsp:include 动作。

·include 指令的优势在于其功能更强大，执行速度也稍快。include 指令允许所包含的文件中含有影响主页面的 JSP 代码，比如响应报送的设置和属性方法的定义。

Forward 与重定向的区别？

1.区别：

1）HTTP 重定向：response.sendRedirect(myNewURL);

·发送的请求信息又回送给客户机，让客户机再转发到另一个资源上，新的 URL 出现在 Web 浏览器中，需要在服务器和客户机之间增加一次通信。

·在用户的浏览器端工作，通过修改 http 协议的 head 部分。

·可带参数传递，可重定向至不同的主机，可重定向相对路径和绝对路径。

2）forward 标准动作

·使用 RequestDispatcher，在服务器端起作用。

·无法重定向至有 frame 的 jsp 文件，可以重定向至有 frame 的 html 文件。

·无法在后面带参数传递。

·重定向后浏览器地址栏 URL 不变。

Java bean 动作：如何创建，使用，输出属性 编写组件的设计规范（普通的那些）

1.JavaBeans 组件的属性可以是：

1）读/写，只读，只写

2）简单：只包含一个值

3）有索引的：表示一组值

2.它仅在使用符合如下规范的公共方法时可用：

1）对每个可读属性，bean 必须有一个 getProperty()方法。

2）对每个可写属性，bean 必须有一个 setProperty()方法。

3.除了属性方法外，JavaBeans 组件还必须定义无参数的构造方法。

MVC

MVC 在 web 应用中的控制流是什么：用什么作为 M、V、C？

有了这些之后客户端怎么请求，控制的是谁，怎么封装业务和数据，怎么获得数据，怎么返回

一、MVC 三部件各对应什么技术、组件？

1）HTML、JSP、XML/XSLT

1）JavaBean，EJB，Java 对象

1）Servlet，StrutsAction

二、MVC 的流程？

控制流：

1）客户端浏览器发送请求。

2）Servlet 获取客户端请求。

3）Servlet 决定哪个元素（JavaBeans，EJBs 或其他对象）来执行指定的请求。

4）JavaBeans 或 EJBs 执行业务逻辑操作，并封装结果。

5）Servlet 使用 RequestDispatcher 的 forward 方法将请求转发到 JSP 页面

6）JSP 通过 JavaBeans 访问结果内容，生成特定的响应。

Struts 框架【前端】

Struts 中的 action 是线程安全的吗？（和 servlet 作对比）（每个请求是产生一个单独的实例还是？）

线程安全 每个请求都会产生一个新的实例

加一个 Service 层：可使用 spring 框架 or ejb 技术

Spring

从 bean 工厂取得的实例，由 spring 提供生命周期的管理。从 bean 工厂取得的实例有两类:一种默认是单例模式，另一种是原型（每一个都产生一个实例）——在 EJB 中可以理解为有状态？无状态？
单例：适合在 service 层（无状态的，和具体的请求无关系）
Spring+Struts 结合，可以把 Struts 中的 action 也作为一个 bean，生命周期也由 spring 来管理创建

EJB

两类：会话 bean，消息驱动 bean

什么是会话 bean：

会话 bean 有三类（多了一个无状态——有状态 无状态 单例。在 EJB 中不要用单例。原因——EJB 的场景[会话 bean 的场景]：分布式）
（EJB 的类型）

1）会话 bean：代表与客户端的一次短暂会话，封装了业务逻辑，可被本地客户端、远程客户端或 web 服务客户端调用。不是持久化的。

·有状态会话 bean

·无状态会话 bean

·单例会话 bean

2）消息驱动 bean：结合会话 bean 和消息监听器的特点，运行业务逻辑组件异步接收消息。一般有 JMS 消息。

适用场景

1）会话 bean：只要求一个客户端访问一个 bean 实例，不要求 bean 状态持久，要求用 EJB 实现 web 服务时。

·有状态会话 bean：需要保存客户端状态、需要考虑线程安全、维护的灵活性及事务处理时。

·无状态会话 bean：对性能有要求又不需要考虑特定客户端信息，实现 web 服务。

·单例会话 bean：它从应用程序启动到服务器停止一直存在，用来存储常见应用程序状态。

4）消息驱动 bean：需要异步接收消息。

与 spring 的区别

1、（spring 不强调分布式，因此强调用单例[默认模式]，而 EJB 强调最好不要用单例，而是有状态/无状态，二者都是多实例，解决分布式问题）
2、另一个区别：spring 中的 action 层或 servlet 想访问 spring 的 bean 时，spring 进行配置很容易；但是在 EJB 中客户端想要获得会话 bean 的实例有两种方法，更复杂。使用会话 bean 的两种方法：依赖注入【前提：在一个 JVM 中】 使用 JNDI（支持远程客户端）

（so EJB 会话 bean 有多种客户类型：本地 远程 web service。支持多种客户类型）

dao 层：

使用 hibernate 或者标准提供的 JPA（基本上完全类似，也有小的区别）

什么是实体（二者都要进行实体关系映射）

轻量级的持久领域对象
An entity is a lightweight persistence domain object.
Typically an entity represents a table in a relational database, and each entity instance

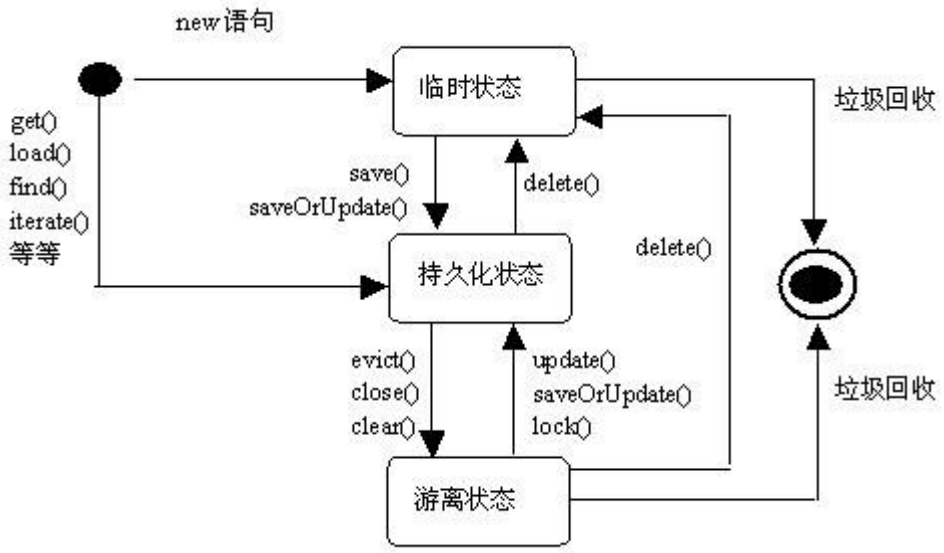
corresponds to a row in that table.
持久数据组件——内存中的对象，对应到数据库中的一个视图；一种持久性的、事务性的以及可以共享的组件，多个客户机可以同时使用其中的业务数据
一个实体的实例是一个对应到数据库中的视图：
更新内存中的实体实例，数据库也自动被更新——java 对象与数据库的同步是由容器自动完成的
一个实体的几个实例可能代表同一底层数据：
例如：许多不同的客户端浏览器同时访问一个产品目录——保证每一个 Bean 实例的数据都是最新的

如何管理实体：实体有哪些状态（管理，调用实体上的什么方法）

Hibernate 有三种状态（哪三种，分别代表什么含义[new save 。。。]）

- 瞬时/临时状态 (Transient)：当通过 new 生成一个实体对象时，这个实体对象就处于自由状态。并没有通过 Session 对象的 save()方法保存进数据库，还没有纳入 Hibernate 的缓存管理中，也就是说 user 对象现在还自由的游荡于 Hibernate 缓存管理之外。在数据库中不存在一条与它对应的记录
- 持久状态 (Persistent)：持久化对象,已经被保存进数据库的实体对象，处于 Hibernate 的缓存管理之中。持久的实例在数据库中有对应的记录，并拥有一个持久化标识 (identifier)。对该实体对象的任何修改，都会在清理缓存时同步到数据库中。持久对象与 Session 和 Transaction 相关联，在一个 Session 中，对持久对象的改变不会马上对数据库进行变更，而必须在 Transaction 终止，也就是执行 commit() 之后，才在数据库中真正运行 SQL 进行变更，持久对象的状态才会与数据库进行同步。在同步之前的持久对象称为脏 (dirty) 对象。
- 脱管/游离状态 (Detached)：游离对象在数据库中可能还存在一条与它对应的记录，只是现在这个游离对象脱离了 Hibernate 的缓存管理。自由对象不会在数据库中出现与它对应的数据记录

三种状态主要取决于对象是否在 session 缓存中
三种状态转化的方法都是通过 session 方法来调用



JPA 是四种状态：前三种一样。多了一个 remove 删除（是什么状态）

•New：没有持久化标识，且还未与持久化上下文关联。

•Managed：有持久化标识，且已与持久化上下文关联。

•Detached：有持久化标识，且当前没有与持久化上下文关联。

•Removed :有持久化标识 ,已与持久化上下文关联 ,但计划将从数据库中删除。The entity’s data will be removed from the data store when the transaction is completed, or as a result of the flush operation.

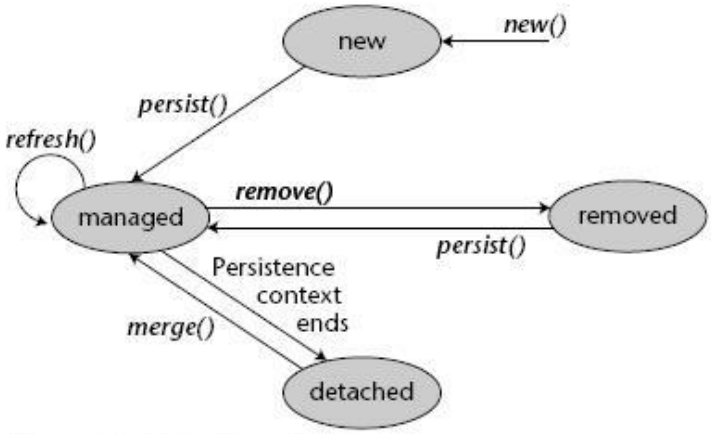


Figure 6.3 Entity life cycle.