1. 填空题 25%
2. 名词解释 15% 尽可能描述清楚
3. 简答题
4. 问答题

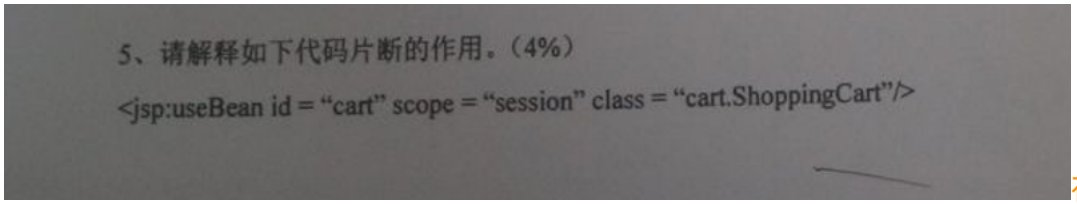
（以下只是对老师所划重点的概述，标准答案需要ppt里找）

其他题目：

名词解释：

RMI：远程方法调用，是对象中间件技术，允许JVM远程调用另外一台JVM的方法

简答：



在Session作用域内创建了一个cart.ShoppingCart类的实例对象，叫cart

一、概览

1. JavaEE能干嘛?

目的：开发企业应用。Java平台企业版（Java EE）平台的目标是为开发人员提供一组功能强大的API，同时缩短开发时间，降低应用程序复杂度并提高应用程序性能。

2. 企业应用的定义

企业应用程序为企业提供了业务逻辑。

① 企业应用通常是集中管理的，经常与其他企业软件交互。

② 在信息技术领域的应用，企业应用必须以更低的成本，更快的速度，和更少的资源来进行设计，构造，和开发。

JavaEE（Java企业应用）不只是一个web应用，还按照分布式多层应用模型开发。

3. 分布式多层应用模型

3.1. 每一层的工作，每一层上是什么组件，做什么?

一般，J2EE应用程序被认为是三层体系结构的，因为它们主要分布在三个地方：客户端，服务器端，数据库或者是后台的遗留层。【Client层、Web层、Business层、EIS层 （JavaEE平台四层应用模型）】

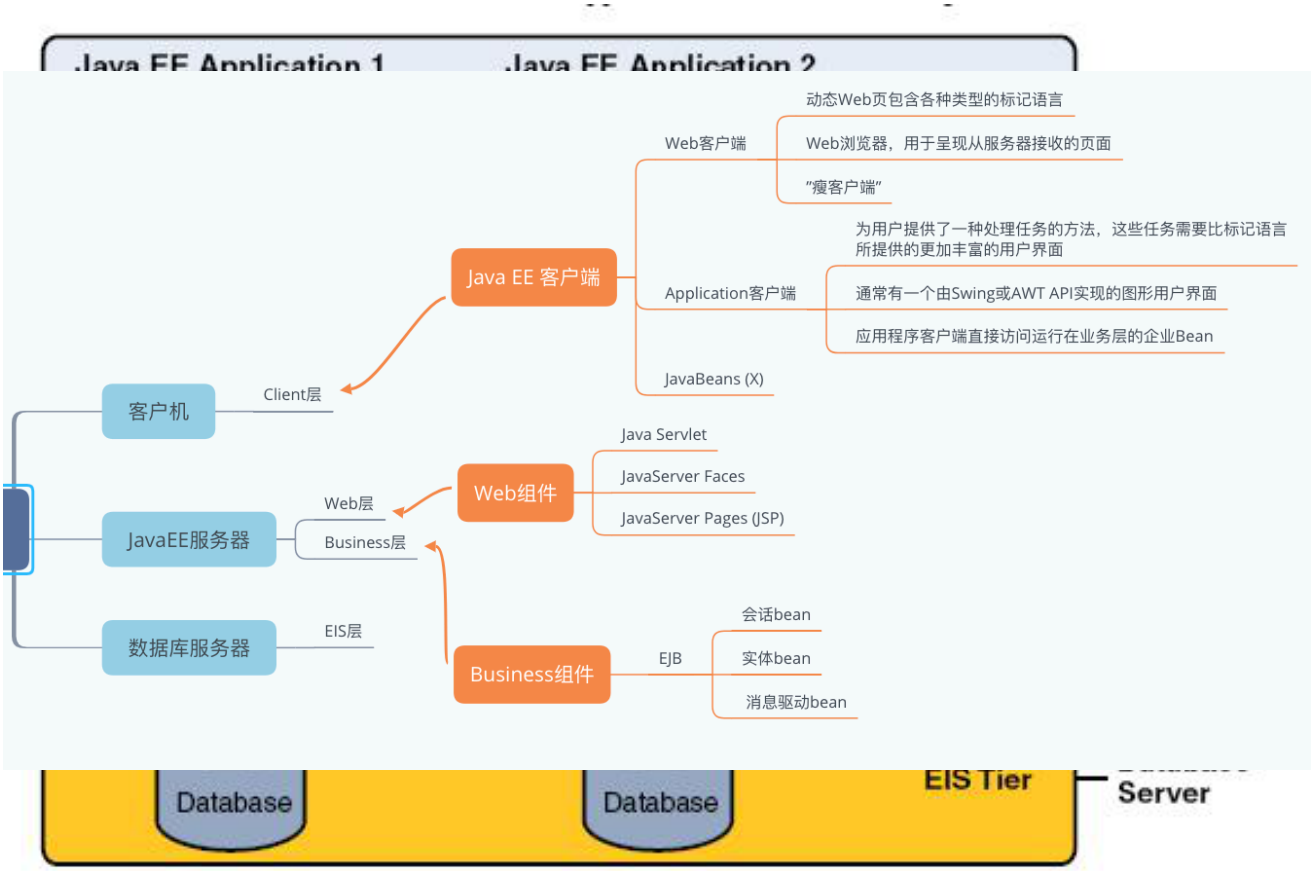


FIGURE 1–1 Multitiered Applications

(1) 运行在客户端机器上的客户层组件（**Client层——Java EE 客户端**）

J2EE两类客户端是?

①Web 客户端

- 动态Web页包含各种类型的标记语言（HTML，XML等）
- Web浏览器，用于呈现从服务器接收的页面
- Web客户端有时称为"瘦客户端"，当您使用瘦客户机时，这些重量级操作将卸载到在Java EE服务器上执行的企业bean（Enterprise beans）

②Application 客户端

- 为用户提供了一种处理任务的方法，这些任务需要比标记语言所提供的更加丰富的用户界面
- 通常有一个由Swing或AWT API实现的图形用户界面
- 应用程序客户端直接访问运行在业务层的企业Bean

③JavaBeans（Java EE规范不将JavaBeans组件视为Java EE组件）

(2) 运行在Java EE服务器上的Web应用层组件（**Web层——Web 组件**）

① Java Servlet
② JavaServer Faces

③ JavaServer Pages (JSP)

(3) 运行在Java EE服务器上的业务逻辑层组件（**Business层——Business 组件**）

① Enterprise JavaBeans (EJB)(会话bean 实体bean 消息驱动bean)

(4) 运行在EIS服务器上的企业信息系统(EIS)层组件.（处理EIS软件并包括企业基础设施系统）

3.2. 分布式：EJB 部署在不同物理机并可以远程通信

3.3. 多层：客户端 web层 Business层 企业信息系统层（不只是数据库）

4. 组件

4.1. 什么是组件？（为什么叫组件？）

- Java EE组件是一个单独的功能软件单元，它可以组装成Java EE应用程序，并与其他组件通过相关的类和文件进行通信。

4.2. Java EE 的组件与其他标准Java类的区别（Java类不能部署到Tomcat上）

- Java EE组件被组装到Java EE应用程序中，经过验证可以很好地构建并符合Java EE规范，并且可以部署到生产中，在那里运行并由Java EE服务器管理。

4.3. 在多层上（客户端 web层 EJB层）分别都有哪些组件？

(1) 客户端：

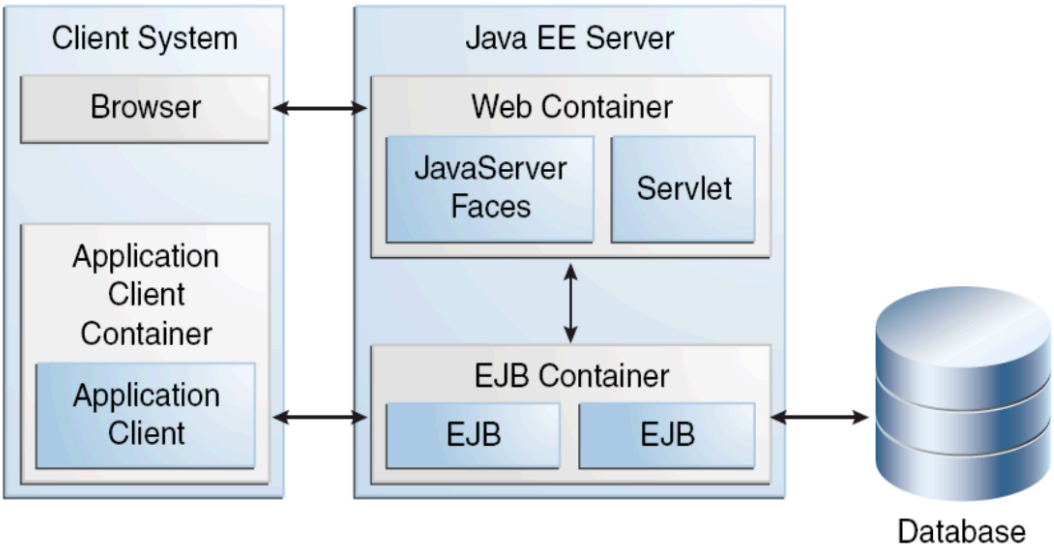
①Web客户端：Applets

②应用客户端：JavaBean组件

(2) Web层：Servlet; Web page(如JSP)

(3) EJB层：EJB（会话Bean，实体Bean，消息驱动Bean）

Figure 1–6 Java EE Containers



5. 容器

5.1. 什么是JavaEE的容器？（运行时环境）

容器是组件与底层平台功能间的接口。在Web组件、企业bean或应用程序客户端运行之前，必须将其装配至Java EE模块，并部署至其容器中。

5.2. Java EE应用服务器是以容器的形式提供服务，容器分哪些类型?

(1) EJB 容器 ： 管理J2EE程序的执行（Java EE Server提供）
(2) Web 容器：管理网页，servlet和部分EJB程序的执行（Java EE Server提供）
(3) Application Client 容器：管理客户端组件应用的执行
(4) Applets 容器：管理applets的执行

5.3. 这些容器提供了哪些服务？（可配置的、不可配置的）

(1) 可配置的服务

1. Security安全：允许您配置Web组件或企业bean，以便只有授权用户才能访问系统资源
2. Trasaction事务：允许您指定构成单个事务的方法之间的关系，以便将一个事务中的所有方法视为一个单元
3. JNDI查找：为EJB中的多个命名和目录服务提供统一的接口，以便应用程序组件可以访问这些服务
4. Remote Connectivity远程连接：管理客户端和企业bean之间的底层通信。 创建企业bean后，客户端调用它的方法就像在本地虚拟机中的调用一样

(2) 不可配置的服务

1. enterprise bean与servlet的生命周期
2. 数据库链接资源池
3. 数据持久化
4. Java EE平台API的连接

6. 打包、部署和维护

6.1. 打包之后归档文件的类型 三种

- web -> war(Web Archive)： 包括servlet类文件，JSP页面文件，支持类文件，GIF和HTML文件，XML配置文件
- ejb -> jar(Java Archive)：包括ejb文件，ejb配置文件
- web+ejb -> ear(Enterprise Archive)

7. 有哪些开发角色?

- Application Component Provider 应用组件提供商
- Applications Assembler 应用程序组装者
- Application Deployer and Administrator 应用程序部署者和管理员
- The Java EE product provider　JavaEE产品提供商
- The tool provider　开发工具提供者

二、web应用

1. web应用

1.1. 什么是web应用？有两类web应用(面向表示，面向服务也就是web service)

(1) Web应用是Web/Application服务器的动态拓展

(2) 有面向表示、面向服务两种类型

1.2. Web应用的结构规范（目录结构吧）

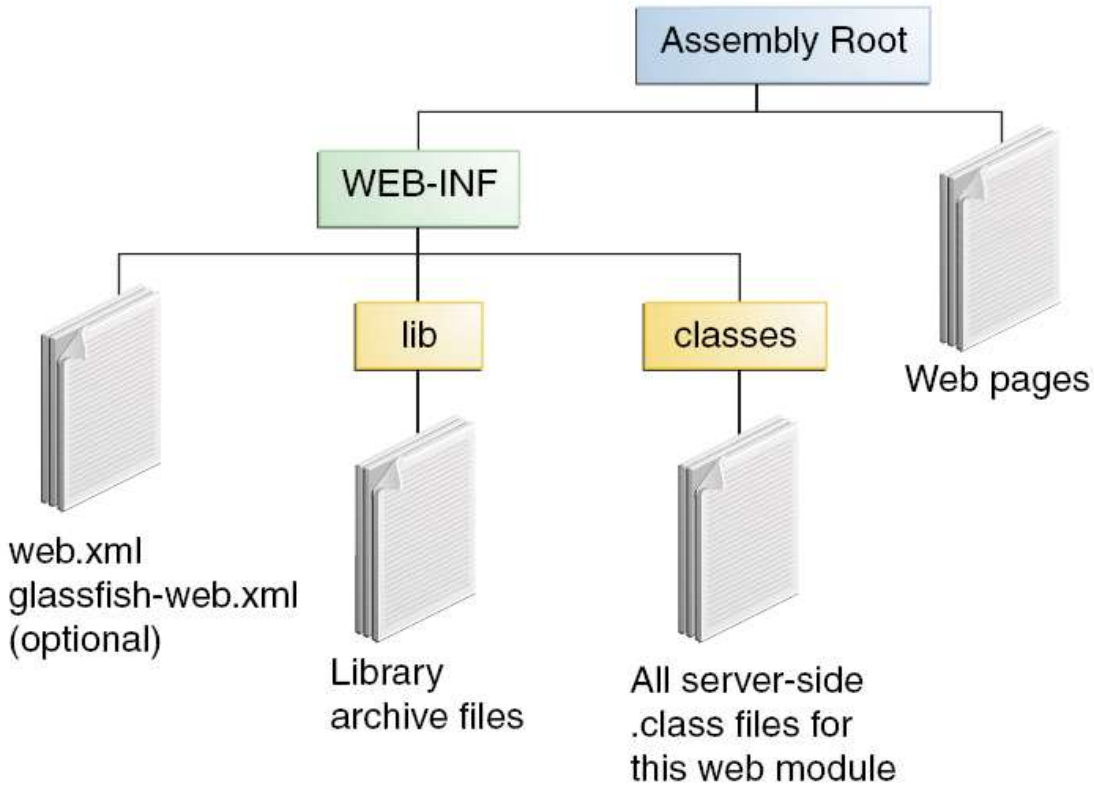
(1) Web模块的顶级目录是应用程序的文档**根目录**——根目录：存放html，XHTML页面，客户端类和文档，静态资源（例如图片等）的目录；

(2) 文档根目录包含名为**WEB-INF**的子目录。WEB-INF包含以下文件和目录：

1. - **classes**：包含服务器端类的文件夹（servlets, enterprise bean class files, utility classes, and JavaBeans components）
2. - **tags** 标签文件夹，包含标签库的实现
3. - **lib** 引用类库文件夹，包含JAR文件（内含企业Bean），和服务端的类调用的类库的JAR包
4. - **配置类文件**，例如web.xml（Web应用程序部署描述符）和ejb-jar.xml（EJB部署描述符）

(3) 您还可以在文档根目录或WEB-INF / classes /目录中创建特定于应用程序的子目录（即包目录）

Figure 5–3 Web Module Structure



2. web.xml配置文件 包含哪些信息 含义?

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
```

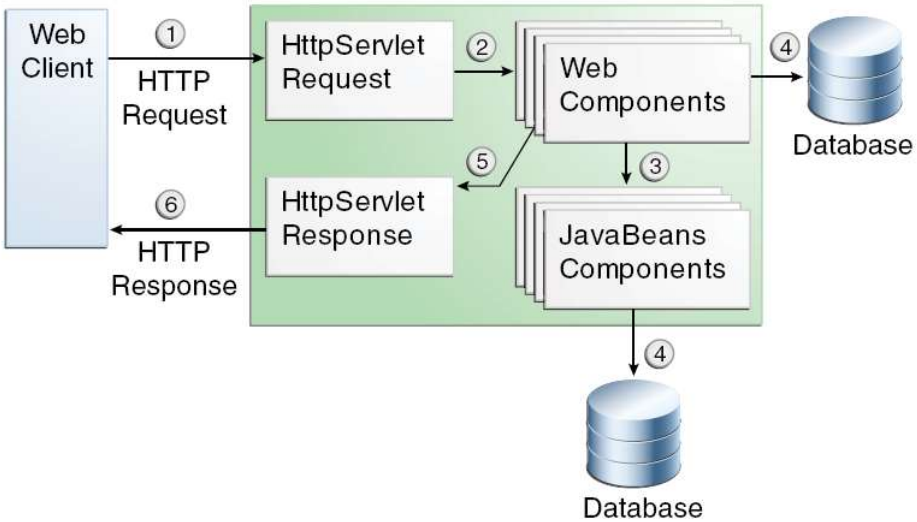
标题:DOCTYPE声明，告诉服务器适用的servlet规范的版本，指定DTD(Document Type Definition)

```
<web-app>          主正文：根元素web-app
    ......
    <servlet>    应包括的Servlet类
        <servlet-name>**</servlet-name>  注册名
        <servlet-class>**</servlet-class>    类名
    </servlet>
    <servlet-mapping>   指定Servlet可以映射到哪种URL模式
        <servlet-name>**</servlet-name>  注册名
        <url-pattern>**</url-pattern>          URL模式
    </servlet-mapping>
    ......
</web-app>
```

3. Servlet（不考察详细API）
   3.1. 面向表示里适合做什么？ 适合作为controller
   3.2. 面向服务里适合做什么

   3.3. 一个Web请求的处理模型（request，response）

Figure 6–1 Java Web Application Request Handling



(1) 客户端向Web服务器发送HTTP请求1。
(2) 实现Java Servlet和JavaServer Pages技术的Web服务器将请求转换为HttpServletRequest对象。
(3) HttpServletRequest对象被传递给Web组件2，该组件可与JavaBeans组件3或数据库交互以生成动态内容4。
(4) 然后，Web组件可以生成HttpServletResponse，或者可以将请求传递给另一个Web组件5。
(5) Web组件最终生成HttpServletResponse对象5。
(6) Web服务器将此对象转换为HTTP响应并将其返回给客户端6。

3.4. Servlet的生命周期

(1) servlet的生命周期其容器控制。
(2) 将请求映射到servlet时，容器将执行以下步骤。
    1. 如果servlet的实例不存在，则Web容器
        1.1. 载入servlet类
        1.2. 创建servlet类的**实例**
        1.3. 调用**init**方法初始化servlet实例
    2. 然后调用**service方法**，传递request和response对象。
(3) 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

3.5. Servlet的线程安全（什么时候遇到，如何处理）

(1) 处理：
    1. 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，避免使用可修改的类变量和实例变量。（实例变量就是静态变量）
    2. 修改servlet类，实现 SingleThreadModel 接口——单线程 ： 该接口指定了系统如何处理对同一个Servlet的调用。如果一个Servlet被这个接口指定,那么在这个Servlet中的service方法将不会有两个线程被同时执行，当然也就不存在线程安全的问题。（但是将引起大量的系统开销）
(2) 什么时候遇到：多线程同时使用同一类变量或实例变量。

3.6. 请求url的组成部分

protocol://[host]:[port][request path]?[query string]

protocol：协议，常用的是http协议
request path 由以下元素组成：
① Context path上下文路径 ： 正斜杠（/）与servlet的Web应用的上下文根的拼接
② Servlet path：与激活该请求的组件别名相应的路径部分，由正斜杠（/）开始。

③路径信息：剩下的部分，包含参数等

3.7. forward vs. sendRedirect （服务器转发，客户端重新提交）

(1) HTTP重定向（response.sendRedirect(NewURL)）：发送的请求信息又回送给客户机，让客户机重新提交请求，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，允许任意URL。

(2) forward标准动作：使用RequestDispatcher，服务器请求资源，直接访问目标地址的URL，把那个URL的响应内容读取过来，然后再将这些内容返回给浏览器，浏览器根本不知道服务器发送的这些内容是从哪来的，所以地址栏还是原来的地址。

4. Tomcat

4.1. Tomcat由哪些组件组成（.net责任链模式）

(1) Server：代表一个服务器

(2) Connector：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

(3) Engine：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

(4) Host：代表虚拟主机，每一个都和某个网络域名想匹配，每一个都可部署多个web应用

(5) 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）

1. 在创建时根据在获得<Tomcat_home>\conf\web.xml和<Webapp_home>/WEB-INF/web.xml载入Servlet类。
2. 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）

4.2. Tomcat处理一个请求的过程，各个组件在其中起到的作用（例如http://localhost:8080/HelloWorld/ ）

(1) 请求被发送到本机端口8080，被JavaHTTPConnector 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URLPATTERN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。

5. Session（两种跟踪机制（cookie, url重写），如何实现
   5.1. Cookie机制
      (1) 当用户第一次访问站点时，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)；
         1. 把sessionID放到HttpSession对象中
      (2) Server创建一个暂时的HTTP cookie
         1. cookie存储这个sessionID
         2. Server将cookie添加到Http响应中
         3. 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

   5.2. URL重写
      (1) 当用户第一次访问站点时，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)；
      (2) Server将sessionID放在返回给客户端的URL中 // response.encodeURL(URL);
      (3) 客户浏览器发送的请求将包含sessionID； // http://…;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
      (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. Cookie(不使用Session单独使用Cookie时如何使用，机制是什么?
   6.1. 使用场景
      (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”（eg：记录用户id,预填充）
      (3) 存储在本地计算机硬盘上
   6.2. Session使用场景（eg：跟踪登录信息）
      (1) 保存在服务器端内存中
      (2) 使用机制不同
   6.3. 机制
      (1) Cookie类也是Servlet包的一部分 // Cookie myCookie=new Cookie(name, value);
      (2) 将cookie加在Http响应信息中 // response.addCookie(myCookie);

(3) cookie被放置到客户机浏览器中，当用户退出Browser时，cookie会被删除掉，不会被存储在客户端硬盘上。
(4) 如果要存储到客户机硬盘上 //myCookie.setMaxAge(int expiry);
(5) 当Server调用Servlet时，浏览器发送的Cookie将包含在HttpServletRequest对象中 //Cookie[] cookies=request.getCookies();

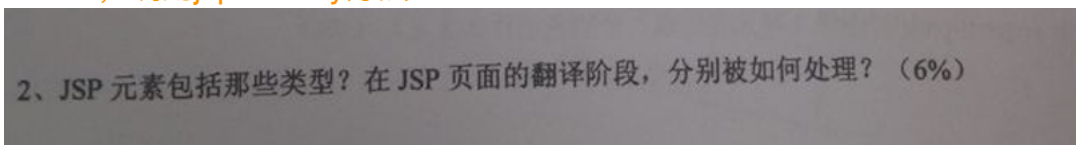
7. 共享信息的四种作用域对象（分别是什么，分别的使用场景，作用域的开始和结束）
   (1) Web Context：作用域为**应用程序运行期**，工程启动后存在，当容器关闭时被销毁；
   (2) Session：作用域为**会话期**，从打开一个浏览器窗口开始，关闭窗口，会话关闭，当会话超时，被销毁；
   (3) Request：作用域为**用户请求期**，只要Server向客户端输出内容，就被销毁；
   (4) Page：作用域为**页面执行期**。它的有效范围只在当前jsp页面里。 从把变量放到pageContext开始，到jsp页面结束。

8. 什么是过滤器，过滤器与其他web组件(如servlet)的区别，用在什么地方
   8.1. 定义：过滤器是一个可以改变请求或响应的标题或内容（或两者）的对象。
   8.2. 过滤器与Web组件的不同之处在于：过滤器通常不会自行创建响应。 相反，过滤器提供可以“附加”到任何类型的Web资源的功能。
   8.3. 主要任务
      (1) 查询request并作出相应的行动
      (2) 阻塞request-response对的传递
      (3) 修改request的headers或data
      (4) 修改response的headers或data
      (5) 与外部资源交互
   8.4. 作用
      (1) 改善代码重用，在不修改servlet代码的情况下想servlet添加功能（如身份验证）
      (2) 用于跨多个servlet执行一些功能，创建可重复使用的功能
      (3) 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）
   8.5. 应用：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、图像转换和加密、动态压缩输出
9. 为什么有些配置不能用注解完成，而一定要写在配置文件中，二者的区别是什么?
   9.1. 注解往往是类级别的，因此，XML 配置则可以表现得更加灵活
   9.2. 在应用中，往往需要同时使用注解配置和 XML 配置
      (1) 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
      (2) 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
10. 监听器（如监听servlet的生命周期，包括创建与销毁等）

10.1. 监视并响应servlet生命周期时间，当事件发生时，listener的相应方法被调用。
10.2. 作用域：作用于整个context
10.3. 生命周期：？？？ ？？？
10.4. 监听器应用场景：监听器用来监听request的生命周期或者是session的变化。可以用来统计登陆次数和登录状态，统计在线人数，监听session是都改变等。（这个答案我有点怀疑

三、**JSP**（了解一下JSP的理念）

生命周期：当客户端请求jsp页面的时候，web容器检查这个jsp页面的servlet是否比页面旧，如果servlet更旧，web容器就把jsp页面翻译成servlet类并执行。如果不存在该jsp页面的servlet实例，容器加载jsp页面的servlet类并且实例化，同时通过调用jspInit方法初始化servlet实例。然后容器调用jspService方法处理请求，传递request和response对象。如果有要销毁页面的servlet，调用jspDestroy方法



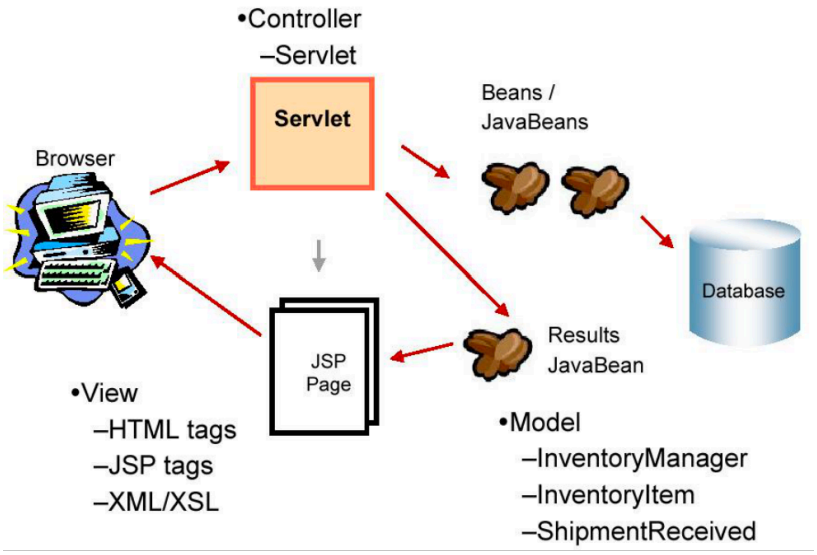
包括指令元素，行为元素，脚本元素。

在翻译阶段，指令元素控制web容器转换并执行jsp页面，例如page指令控制属性，include指令表示将jsp页面转换为servlet时引入其他文件，taglib指令咋混华为调用标签处理程序。行为元素会被转换成方法来调用java bean，比如set/get。或是转换成Java Servlet API的调用，比如forward/include，或是转换成浏览器的特定标记来激活一个applet，比如jsp:plugin。脚本元素会插入到jsp页面的servlet类中，调用jsp的表达式解释器。

1. 什么是JSP

在传统的网页HTML中插入Java程序段和JSP标记，从而形成JSP文件，后缀名为(*.jsp)。可以轻松创建含有静态与动态web组件的web内容。

2. 包含两类文本（静态、动态jsp element）
   2.1. JSP页面是一个文本文档，包含两种类型的文本：
      (1) 静态数据，可以用任何基于文本的格式（如HTML，SVG，WML和XML）表示
      (2) 动态内容，由JSP元素（jsp element）构成。
3. 隐式对象
   3.1. 隐式对象被Web容器创建，包含关于某一个特定请求、页面、会话或应用的信息
   3.2. Out对象：
      (1) 做PrintWriter对象能做的一切事情；
      (2) 调用print()/println()方法，把信息回送给客户端浏览器；



(3) 作用域是当前页面（page）；
(4) 每个JSP页面有一个out对象的实例；
(5) 缺省采用缓存，可以使用page指令调整其大小；

3.3. Request对象：使用request对象得到请求信息中的参数
3.4. Response对象：使用response对象发送重定向、修改HTTP头、指定URL重写
3.5. Session对象：javax.servlet.http.HttpSession的实例
3.6. Application对象：从web.xml获取初始化参数、访问RequestDispatcher
3.7. PageContext对象：对页面作用域的属性的访问

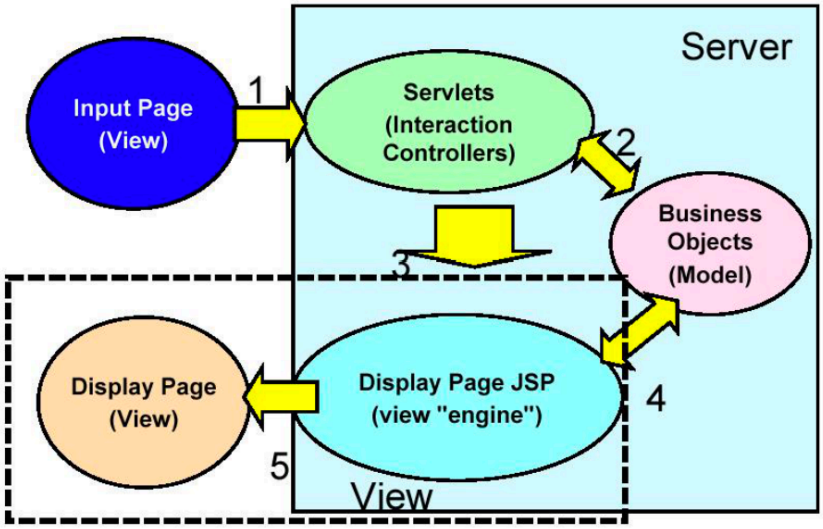
4. JavaBean（jsp中用的 狭义的）它的设计规范是怎样的?
   4.1. 一个JavaBean组件属性分如下几种：Read/write, read-only, or write-only
   4.2. 简单，也就是说它只包含一个数值或一个索引（即指代表一个数组）
   4.3. 访问方法简单，运用遵循如下约束的public方法：
      (1) 对每一个可读的属性，具有方法：PropertyClass getProperty() { ... }
      (2) 对每一个可写的属性，具有方法：setProperty(PropertyClass pc) { ... }
   4.4. JavaBean还需要有一个无参构造函数

四、MVC

1. M、V、C都是由什么组成的
   1.1. Model：Bean/JavaBeans(InventorManager,InventorItem,ShipmentReceived)
   1.2. View：Browser（HTML tags;JSP tags XML/XSL）
   1.3. Controller: Servlet
2. 各种不同的情况下，用什么技术实现 ？？？？？？
   2.1. 如V(view)在Web中可分动态页面和静态页面
3. 整个的控制流程是怎样的?
   ① 客户端发出请求（Web浏览器）。

② Servlet获取客户端的请求。
③ Servlet确定执行指定请求所需的程序元素（JavaBeans，EJB或其他对象）。
④ JavaBeans或EJB为servlet执行业务逻辑操作，并封装结果。
⑤ Servlet选择一个表示模板（JSP），用于将内容传送回客户端。
⑥ JSP通过访问JavaBeans提供的结果内容来生成特定响应。

五、EJB



1. 什么是“企业Bean”（EJB与分布式有关，而狭义的JavaBean是指JSP中那个用于传递的Bean）

定义：企业bean是用Java编程语言编写的，是一个封装应用程序业务逻辑的服务器端组件。

两类企业bean的作用

会话Bean（三种分别是什么）（前两种可以对照Spring中的Bean，相同/区别？？？）：为客户端执行某个任务，也有可能实现一个Web服务，**封装业务逻辑**，可以被本地的或者远程的客户端以及网络服务客户端的页面（web service client views）

有状态Bean stateful：

有状态的会话 bean在方法调用时可保持对话状态。

譬如客户的网上购物车。客户开始网上购物时，可以从数据库中检索客户的详细信息。客户往购物车里面添加商品或者从里面删除商品、下订单等时调用的其他方法也可以使用这些详细信息。

不过，有状态的会话bean是暂时性的，因为出现会话终止、系统崩溃或者网络故障后，状态不复存在。

客户端请求有状态的会话bean实例时，就为该客户端分配一个有状态的实例，并为该客户端保持该组件的状态。

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。

无状态Bean stateless：

无状态的会话 bean没有内部状态。

它们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。因此，每次调用无状态的业务方法都独立于前一次调用，譬如计算税款或者运费。用某个应税值调用计算税款的方法时，对税款值进行计算并返回给调用方法，而不必保存调用者的内部状态供以后调用。因为这些bean并不保持状态，所以容器对它们进行管理就很简单。

客户端请求无状态的bean实例时，可以从容器保持的无状态的会话bean 实例池当中接收一个实例。

另外，因为无状态的会话 bean可以共享，所以容器可保持数量较少的实例为许多客户端提供服务。

想指定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。

单例 singleton

消息驱动Bean：为一个特定的事件充当监听（如Java Message Service API）消费方式：异步

六、对象关系映射（ORM）

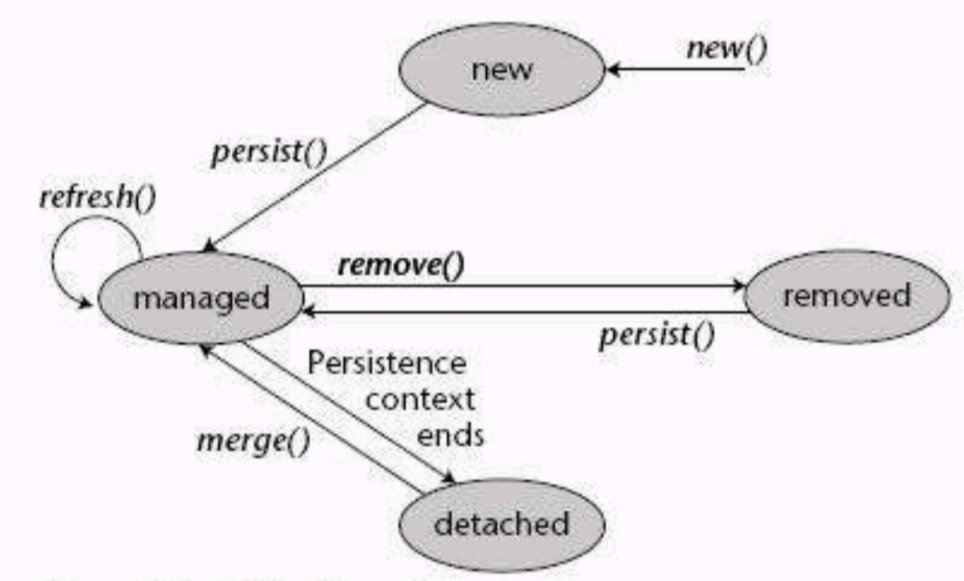
JDBC API包括两部分 application-level interface和service provider interface （不知道该放在哪里 就在这里吧）

1. JPA
   1.1. 什么是JPA

   JPA(Java Persistence API)提供关系型数据的Object-Relation映射工具，完成数据持久化
2. 实体类（生命周期，哪几种状态）
   2.1. 什么是实体类
   (1) 一个轻量的持久化领域对象
   (2) 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，通常代表关系型数据库中的一张表，一个实例对应表中一行。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。
   (3) 一个实体类有以下特点
   1. 类必须用 javax.persistence.Entity 注解
   2. 必须有一个public/protected，无参构造函数。除此之外可以有其它构造函数

3. 类不能被声明为final，而且任何方法或持久实例变量也都不能声明为final。
4. 如果一个实体实例作为游离态对象（例如通过会话bean的远程业务接口）按值传递，则该类必须实现可序列化接口。
5. 实体可以继承实体类或非实体类，非实体类也可以继承实体类
6. 持久化实例变量必须被声明为private/protected/package-private，而且只能被这个实体类的方法直接访问。Clients必须通过访问器或业务方法来访问实体的状态。

2.2. 实体类的生命周期————实体类的4种状态，含义，及转换（临时、持久、游离（、删除状态不知道ppt里有没有））



(1) new
1. 通过new生成一个实体对象
2. 通过JVM获得了一块内存空间，但是并没有保存进数据库，还没有纳入JPA EntityManager管理中
3. 在数据库中不存在一条与之对应的记录

(2) managed
1. 纳入JPA EntityManager管理中的对象
2. new状态，可通过persist()方法将其与数据库相关联，成为**持久化对象**
3. 或使用find()方法，得到持久化对象
4. 在数据库中存在一条与之对应的记录，并拥有一个持久化标识（identifier）
5. 对持久化对象的操作，**影响数据库**

(3) detached
1. 游离对象
   - 例如：find()方法调用后，可关闭EntityManager，成为游离对象
     - 如：em.clear()
   - 对游离对象的操作，不影响数据库
   - 和new状态的区别

- 在数据库中可能还存在一条对应的记录，只是游离对象脱离了JPA EntityManager的管理

2. 游离态转化为持久对象：调用merge()方法

(4) removed

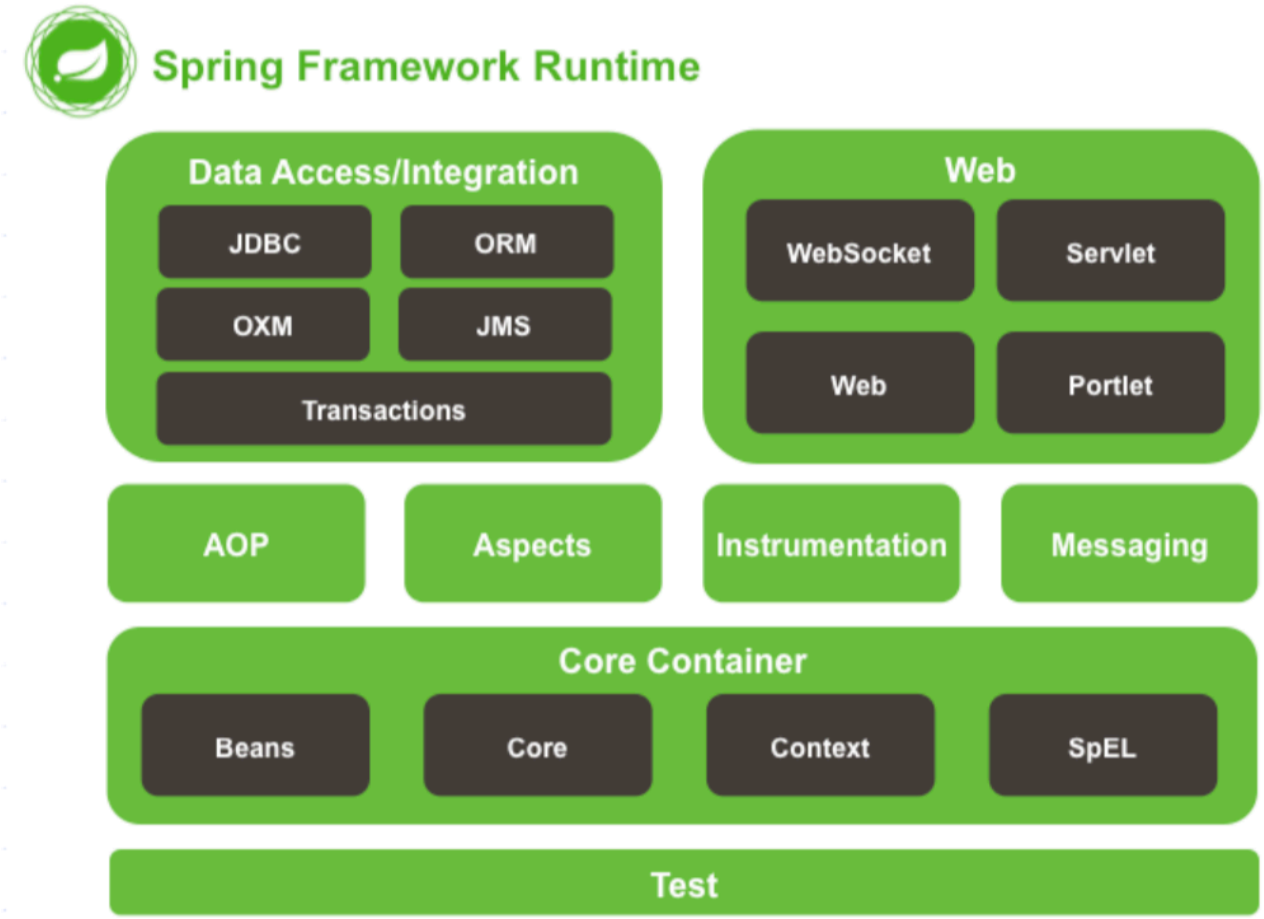
- remove()方法
- 删除数据库中的记录
- 在适当的时候被垃圾回收

七、Spring

Client引用EnterpriseBean的两种方式：依赖注入 JNDI查找

JNDI：Java命名目录接口，为开发人员提供了查找和访问各种命名和目录服务的统一，通用的方式，可以按照名称查找远程对象，中央注册中心存储了各种对象

1. 有哪些主要模块



1.1. 核心模块

(1) Core和Beans模块提供基本部分的框架， 包括IoC和Dependency Injection功能

(2) ApplicationContext上下文模块：建立在Core和Bean模块的基础上,使 Spring成为框架

(3) 数据访问/集成模块——包括JDBC、ORM、OXM、JMS和 Transaction模块

1. JDBC模块
2. ORM、事务模块
   2.1. ORM模块提供了对象-关系映射API集成层，包含JPA，JDO，Hibernate
   2.2. 事务模块提供了编程和声明式的事务管理的支持，利用了AOP模块
3. JMS两种消息消费方式：同步 异步

(4) AOP

(5) Web模块——建立在应用上下文模块基础上，提供适 合Web系统的上下文

1.2. 使用MVC的话使用哪些模块

1.3. 结合JPA/Hibernate的话使用哪些模块

1.4. …

2. Bean

2.1. 什么是Spring的Bean（所有能被其容器管理的类都叫Bean？）

八、安全 两种实现方式

1. 声明式安全（Declarative security）：角色 用户 策略

1.1. 通过使用开发描述（deployment descriptors）或者注释（annotations）来实现一个应用组件的安全要求；Deployment descriptors对于一个应用来说是外在的，包括了具体描述安全角色和访问要求向具体的环境安全角色、用户和策略的映射关系

(1) 使用表单/浏览器/客户证书

(2) 把安全配置情况存储在配置描述文件中

1. 角色、访问控制、身份认证要求
2. Web.xml

(3) 优先选择声明式安全

2. 编程式安全（Programmatic security）：嵌入在应用中被用来作出安全决策；Programmatic security 在只使用声明式安全不足以表现安全模型的时候使用。

- 细粒度的编程安全性
- HttpServletRequest接口
  - getAuthType();
  - getRemoteUser();
  - getUserPrincipal();
  - isuserInRole();

以上这些方法允许组件根据调用者或远程用户的安全角色来作出业务判断。

3. 安全性包括

3.1. 用户和组

(1) 用户：应用程序终端用户的账户名（ID）

(2) 组：命名的用户集合，可以包含0-n个用户；通常用于表示具有类似系统资源访问权限的用户

3.2. 认证和授权

(1) 认证（Authentication）：用户向系统证明“我是谁”

(2) 授权（Authorization）：引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”

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

审计和日志记录（防火墙）

审计和日志记录

- 收集、存储和分发整个系统中安全事件信息
- 查看执行的活动
- 帮助检测和调查应用程序环境中的潜在弱点

防火墙

- 禁止任何不需要的协议或客户类型访问应用程序

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

认证方式：

HTTP基本验证（HTTP Basic Authentication）

基于表单的验证（Form-Based Authentication）

基于客户端证书的验证（Client-Certificate Authentication）即通过HTTPS（HTTP over SSL）来保证验证的安全性