名词解释：15% 填空题 25% 简答题：30% 问答题：30%

一、名词解释

企业应用：Java EE 是 sun 公司推出的企业级应用程序版本和标准，在 Java SE 基础上构建，是一个让企业开发大幅度缩短投放市场时间的体系结构。

能够帮助开发和部署可移植、健壮、可伸缩、安全、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 web2.0 应用程序。

例如：servlet，jdbc，jsp，ejb，邮件服务

java EE 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中运行。

容器：支持组件和底层平台功能的接口，通过配置一些文件就可以提供安全、事务管理、JNDI 查找和远程连接的服务，java EE 组件必须编译成模块发布到容器中才能运行。

web 应用：web 应用是对 web 或应用服务器的动态扩展

面向表示：包括用与交互的 web pages，用于响应用户的动态内容

面向服务：实现了 web 的端点服务

web 模块：web 资源的最小可发布且可用的单元

web 资源：静态 web 内容文件，比如图片

servlet：一种 java 编程语言类，是服务器端的小程序，用于扩展服务器的能力，用请求-响应的编程模型来管理访问

jsp 页面：一种文本，包括静态数据（html，xml，svg）和 jsp 元素（标准 jsp 语法，xml 语法，如 java 代码，指令标签）

EJB：ejb 是服务器端组件，封装业务逻辑。

简化了大型分布式企业级应用：

ejb 提供系统级别的服务，包括事务管理、安全认证等；

客户端不用实现逻辑代码，可以专注表现层，更轻便。

提供基于接口的组件，标准 api，便于调用。

Jpa：为 java 开发者提供对象关系映射功能，便于管理 java 应用中的数据

实体类：轻量持久数据组件，通常代表数据库中的一张表，每一个实例是表中的一行

二、概念

- **j2ee 采用的技术-分布式多层应用模型**

客户层：运行在客户端机器上的客户端组件

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

动态页面—web 客户端组件，利用浏览器展示界面

Applets—采用 java 创建的基于 html 的程序

web 层：运行在 J2EE 服务器上的 web 层组件，可以直接和数据层交互，也可以通过业务层和数据库交互

java servlet，jsf，jsp，可能包括 java bean 组件

业务层：运行在 J2EE 服务器上的业务逻辑层组件，包括 EJB，业务逻辑代码

ejb—企业 bean，完成业务逻辑处理，负责和数据库交互

逻辑代码—完成特点商业领域需求的逻辑代码

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

- **j2ee 组件和 java 类的区别**
  - 组件按照 java EE 的规范被编译成 java EE 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，数据库连接等功能
  - 普通 java 类安装 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
- **容器提供的服务，可配置和不可配置**

  容器类型：EJB 容器，Web 容器，应用客户端容器，Applet 容器

  可配置的服务：

  安全性-配置用户 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 之间的底层通信

  不可配置的服务（生命周期模型）：

  EJB 和 servlet 生命周期，数据库连接资源池，java ee 平台 api 的获取
- **web、ejb 两种类型的容器，关于这两种容器具体提供的服务**

  EJB 容器：管理 EJB 的执行，EJB 容器和组件可以运行在 java EE 服务器上

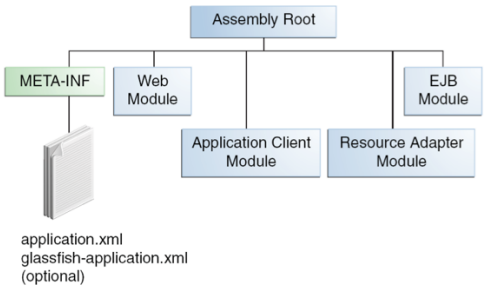
  Web 容器：管理 web pages，servlets 以及一些 EJB 组件的运行，web 容器和组件可以运行在 javaEE 服务器上
- **标准的 javaEE 单元的内容**

  组件：企业 bean，web page，servlet，applet

  可选部署描述文件：javaEE 部署描述符（由 javaEE 规范定义，配置部署设置）

  运行时部署描述符（配置 javaEE 实现时需要的特定参数）
- **部署打包文件的三种类型**

  web: .war ejb: .jar web+ejb: .ear
- **ear 文件的文件结构**



- **ear 文件的四类模块及其内容**

  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 应用部署描述文件

  ejb 应用模块：*.jar，包括企业 bean 类文件，EJB 部署描述文件

  应用客户端模块：*.jar，包括类文件，应用客户段部署描述文件

  资源配置模块：*.rar，包括所有的 java 接口、类、本地库、其他文件以及资源配置部署描述文件
- **java web 应用的请求处理过程**
  - 客户端向 web 服务器**发送 http 请求**
  - 采用 servlet 和 JSP 技术实现的 web 服务器将请求转化为**HTTPServletRequest object**

• 这个 object 会**被发送给 web 组件**，web 组件与 java bean 组件或数据库交互，产生动态内容

• web 组件可以生成一个 **HTTPServletResponse** 或者将请求传递给另一个 web 组件

• web 组件最终生成一个 **HTTPServletResponse object**

• web 服务器将这个 object 转化为**一个 HTTP response**，并将它返回给客户端

- **web 容器提供了哪些服务**

  请求分派，安全，并发，生命周期管理，提供 API 给 web 组件

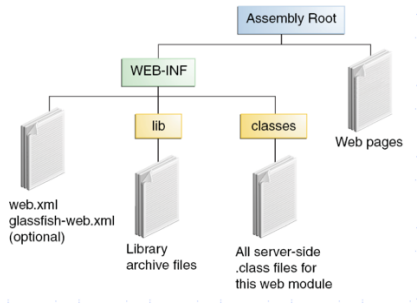
- **web 模块包括哪些内容：**

  开发的 web 组件，web 资源（静态页面、图片），服务器端实用类（数据库 bean，shopping carts 等），客户端类（applets，实用类）

- **Web 模块的文件结构**

  最顶层是应用的 document root：存放 XHTML 页面、客户端类、静态 web 资源

  document root 包括一个叫做 WEB-INF 的子文件夹



- **Web-INF 的子目录**

  Tags：标签文件

  Classes：服务器端 classes，servlets、企业 bean classes、实用类、java bean

  Lib：企业 bean 的 jar 文件，服务器端会用到的 jar 库文件

  部署描述文件：web 应用部署描述文件 web.xml，ejb 部署描述文件 ejb-jar.xml

- **Tomcat 的组成部分**

  Tomcat 是一个基于组件的服务器，构成组件可在 server.xml 中配置。

  Server：代表某个服务器

  Service：代表服务，包含一个 engine，多个 connector

  Connector：在某个指定端口侦听客户请求，将获得的请求交给 Engine 处理

  Java HTTP Connector 在端口 8080 侦听来自客户浏览器的 http 请求

  Engine：将获得的请求匹配到某个虚拟主机上，并把请求交给该 Host 来处理

  Host：虚拟主机，每个和某个网络域名相配，可部署多个 web 应用，对应不同的 context

  Context：对应一个 web 应用。由 servlet、html 页面、java 类、jsp 页面等资源构成

- **Tomcat 处理 HTTP 请求的过程**

  url: http://localhost:8080/HelloWorld/

  • 请求被发送到本机端口 8080，被 java HTTP Connector 获得

  • Connector 将该请求交给它所在的 Service 的 Engine 处理，并等待 Engine 的回应

  • Engine 获得请求，匹配所有虚拟主机

  • Engine 匹配到名为 localhost 的主机

  • localhost 主机获得请求，匹配所有的 Context

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

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

• Context 匹配到 url pattern 为/的 Servlet
• 构造 HttpServletRequest 对象和 HttpServletResponse，作为参数调用该 Servlet 的 service 方法
• Context 把执行完之后的 HttpServletResponse 对象返回给 localhost 主机
• Host 吧 HttpServletResponse 对象返回给 Engine
• Engine 把 HttpServletResponse 对象返回给 Connector
• Connector 把 HttpServletResponse 对象返回给客户浏览器

- **servlet 生命周期**

servlet 的生命周期由 servlet 所部署的容器控制，当一个客户端请求发送到服务器时，容器执行以下步骤：
• 若 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 方法，传递 request 和 response 对象
• 若容器需要移除这个 servlet，它会通过调用 servlet 的 destroy 方法释放它

- **url 所包含的元素**

http://[*host*]:[*port*][*request-path*]?[*query-string*]
host：主机地址
port：端口
request-path：由以下三部分组成
context 路径—斜线/和 servlet 的 web 应用的 context root 的拼接
servlet 路径—与激活该请求的组件别名相应的路径部分，由斜线/开始
路径信息—请求路径部分
query-string：传递的参数和值

- **url 和 uri 的区别**

uri：统一资源标识符，用来唯一的标识一个资源
url：统一资源定位器，是一种具体的 uri
区别在于，
• url 可以用来标识一个资源，并且指明了如何定位这个资源。
• uri 是一种语义上的抽象概念，可以是绝对的，也可以是相对的；url 必须提供足够的信息来定位

- **如何处理 servlet 线程安全的问题**

servlet 默认是多线程的，实例变量是在堆中分配的，不是线程安全的。request、response、局部变量是线程安全的。
处理 servlet 线程安全主要有三种方法：
• 编写线程安全的类，避免使用可用修改的类变量和实例变量
• 同步对共享数据的操作，使用 synchronized 关键字
• 修改 servlet 类，使其实现 SingleThreadModel 接口（单线程）

- **Web 应用中的 Session 的两种跟踪机制**

cookie 机制，url 重写。一般使用 url 重写机制，因为 cookie 在客户端可能不启用。

- **Cookie 机制的实现原理**

• 用户第一次访问站点时，会创建一个新的会话对象（HttpSession），server 为会话对象分配一个唯一的会话标识号（SessionID）
• server 创建一个暂时的存储了该会话标识号的 HTTP cookie，server 将 cookie 添加到 http 响应中，cookie 被放置到客户端浏览器中，存储在客户端硬盘

• 客户端浏览器发送包含 cookie 的请求

•根据浏览器发送的 sessionID 信息（cookie），server 找到对应的 HttpSession 对象，跟踪会话

• 在会话超时间隔期间，如果没有收到新的请求，server 将删除此会话对象

- **url 重写的实现原理**

  • cookie 被客户禁用时，采用 url 重写机制

  • 用户第一次访问站点时，会创建一个新的会话对象（HttpSession），server 为会话对象分配一个唯一的会话标识号（SessionID）

  • server 将 sessionID 放在返回给客户端的 url 中国年

  • 客户浏览器发送的请求将包含 sessionID

  •根据请求包含的 sessionID 信息（url），server 找到相应的 HttpSession 对象，跟踪会话

- **cookie 和 session 的区别和使用场景**

  Cookie：

  • 跟踪会话，也可以独立于 http 会话使用

  • 长期记住用户信息

  • 存储在客户端本地硬盘上

  •可记录用户登录 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

  Session：

  • 保存在服务器眼内存中

  • 使用机制不同

  • 跟踪用户的购物车、导航信息、登录状态

- **web 应用中共享信息的四种作用域对象**

  Web Context：作用域为应用程序运行期，工程启动后存在，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，会话超时后被销毁

  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。页面创建时开始。

- **过滤器和其他 web 组件的区别**

  过滤器是一个 object，可以修改 request 或者 response 的 header 和 context。

  与 web 组件的区别在于，过滤器不是自己创建一个响应，而是提供可以被任何类型的 web 资源获取的功能，它不应该对 web 资源有任何依赖。

- **过滤器的作用和使用场景**

  作用：

  • 改善代码重用，在不修改 servlet 代码的情况下向 servlet 添加功能—身份验证

  • 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

  使用场景：代码重用、应用安全策略、日志、图像转换、加密、动态压缩输出、为特定目标浏览器传输 xml 输出

- **监听器监听的生命周期事件**

  监听器是一个对象，可以监听 Servlet 的生命周期并做一些操作。

  可以监听 web context、session、request 三种生命周期。

  监听系统关闭或开启、监听用户访问次数、监听用户是否登录、监听 session 是否改变。

- **注解和 xml 文件的区别是什么**
  - 注解往往是类别级的，xml 配置则可以表现的更加灵活
  - 对于类别级且不会发生变动的配置可以优先考虑注解配置
  - 对于第三方类以及容易发生调整的配置优先考虑 xml 配置
- **jsp 元素的内容和翻译处理过程**
  - **指令元素**：控制 web 容器转换并执行 jsp 页面
    page 指令—用于一个或者多个属性
    include 指令—将 jsp 页面转换为 Servlet 时引入其他文件
    taglib 指令—转化为调用标记处理程序，该程序实现自定义标签调用
  - **脚本元素**：插入到 jsp 页面的 servlet 类中
    表达式—作为传递的参数来调用 jsp 表达式解释器
    声明，代码段（scriptlet）
  - **行为元素**：
    允许用户在页面中使用 java bean 组件和有条件的把页面控制权转移到其他页面上
    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
  - 注释

  动态 jsp 元素：指令、脚本、行为
- **Jsp 生命周期**

  jsp 页面的请求是请求一个 servlet，因此，其生命周期取决于 java servlet 技术，优势在于 jsp 是容器自动创建线程，响应速度快于 servlet 的访问。
  - 请求被**映射**到 jsp 页面
  - **检查并翻译**：web 容器检查这个 jsp 页面的 servlet 是否比 jsp 页面旧。如果 servlet 更旧，web 容器将 jsp 页面翻译成 servlet 类并执行。
  - **实例并初始化**：如果不存在该 jsp 页面的 servlet 实例，容器加载 jsp 页面的 servlet 类并实例化，同时通过调用 jspInit 方法初始化 servlet 实例
  - **处理请求**：容器调用_jspService 方法，传递 request 和 response 对象
  - **终止**：如果需要销毁 jsp 页面的 servlet，容器调用 jspDestroy 方法
- **page 指令常见属性**

  属性描述了对 jsp 容器有意义的设置。
  session—指定当前的 jsp 页面访问请求是否要包括在一个 http 会话中
  import—指定要导入的包
  extends—指定要扩展的其他类
  contentType—设置页面的文本类型和字符编码
  buffer—设置 buffer 的大小
  ThreadSafe—false 意味着一次只能有一个线程执行 jsp 中的代码
  errorPage：指定异常发生时要跳转到的错误页面
- **Include 指令**

  将其他文件中包含的文本插入到该 jsp 页面中，实现 jsp 页面的模块化，使 jsp 的开发和维护更简单
- **taglib 指令**

  指明标签

- **脚本元素创建的对象**

  脚本元素：声明、代码段、表达式

  declarations：声明变量或方法，只在当前页面可用，被翻译成 jsp 页面 servlet 类中的声明

  scriptlets：包括任何需要在脚本中使用的代码片段，被翻译成 java 语句片段并且插入 jsp 页面 servlet 的 service 方法

  expressions：将转化为字符串的表达式值插入数据流，返回给客户端。被翻译成将表达式值转化为 string 对象并放入隐式 put 对象的语句。

- **常见隐式对象**

  out 对象：做 PrintWriter 对象能做的一切事情

  request 对象：得到请求信息中的参数

  response 对象：发送重定向，修改 http 头，指定 url 重写

  session 对象：HTTPSession 的实例

  application 对象：从 web.xml 获取初始化参数、访问 RequestDispatcher

  pageContext 对象：对页面作用域属性的访问

- **include 动作和 include 指令的区别**

  • include 指令是编译时包含，是静态的；

  include 动作是运行时包含，使用 RequestDispatcher

  • include 指令比 include 动作快

  • include 指令不能有重复内容；include 指令在被包含 jsp 执行完毕后，再包含到本页面，可与本页有重复内容，

- **forward 动作和 http 重定向 redirect 的区别**

  http 重定向：response. sendRedirect（newURL）发送的请求消息又回送给客户端，让客户端再转发到另一资源上，新的 url 出现在 web 浏览器中，需要在服务器和客户端之间增加一次通信

  forward 动作：使用 RequestDispatcher，jsp 的转发功能是在服务器自身实现的

- **java bean 的设计规范**

  • java bean 组件的属性包括单值或数组，read / write、read-only、write-only。

  • 必须能够简单的被公有方法获取：每一个可读的属性，必须有 getProperty 的方法；每一个可写的属性，必须有 setProperty 方法

  • 必须定义一个无参构造函数

- **java bean 的四种作用域**

  application，session，request，page

- **企业 bean 使用场景**

  大规模分布式应用，需要事务支持保证数据的安全性，需要不同的客户端

- **会话 bean 和消息驱动 bean，这两类的分别的作用？区别？**

  会话 bean：封装业务逻辑，可以被本地或远程客户端、web 服务调用，非持久化。访问分布式的 server 端应用，客户端会调用会话 bean 的方法。

  消息驱动 bean：允许 javaEE 应用异步处理消息，监听并接收 JMS 消息，可以处理 JMS 消息和其他类型的消息

  区别：

  •会话 bean 允许同步的发送或接收 JMS 消息，不支持异步；如果要异步接收消息，使用消息驱动 bean

- **会话 bean：三类，有状态、无状态、单例。分别的含义**

  有状态会话 bean

  定义：实例变量维持一个唯一的客户端或 bean 会话的状态。不共享，可以只有一个客户端，当客户端终止时，会话 bean 也终止。如果客户端移除了这个 bean，会话终止，状态消失。

  使用场景：1）client 跨越多个方法调用，需要维持相关信息 2）client 需要和其他应用组件进行交互 3）该 session bean 需要多个 ejb 交互才能实现，需要维护一个工作流，如购物车

  无状态会话 bean

  不维持会话状态，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 可从容器的无状态会话 bean 实例池中获取实例

  可以实现 web 服务，但有状态会话 bean 不可以

  使用场景：1）对特定的客户端，bean 没有状态信息 2）在单次方法驱动时，bean 对所有客户端进行同样的操作，如发送 e-mail 3）实现 web 服务

  单例会话 bean

  在应用的生命周期中，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 服务端点

  使用场景：1）应用的初始化工作 2）关闭时的清理工作

- **JNDI 查找和依赖注入的区别**

  依赖注入：1）clien 有 ejb 的实例引用，采用 java 注解的方式 2）ejb 和 client 在同一个 jvm 中，是最简单的获取 ejb 引用的方式 3）jsp、web 应用、其他 ejb、java ee 应用客户端都支持依赖注入

  JNDI 查找：1）ejb 和 client 在不同的 jvm 或者同一个 jvm 中，通过 JNDI 来查找实例 2）client 端可以是简单的 j2se 应用 3）JNDI 支持识别 java ee 组件的全局语法，简化了显式查找的语法

- **客户端的类型**

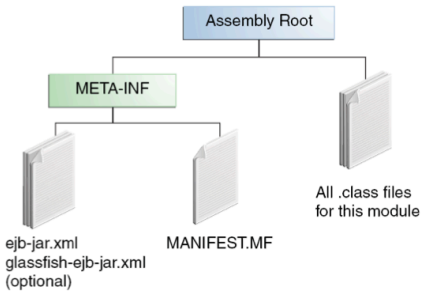
  远程客户端：1）client 和 ejb 在不同的 jvm 上，client 可以是 web 组件、应用客户端、其他 ejb 2）对于远程 client，ejb 的位置透明 3）必须通过业务接口访问 ejb，不能无接口访问 4）采用 JNDI 查找 5）客户端通过远程接口调用 ejb，效率不高

  本地客户端：1）client 和 ejb 在同一个 jvm 上，client 可以是 web 组件或其他 ejb 2）ejb 位置不透明 3）可以采用无接口方式、依赖注入、JNDI 查找访问 ejb，不用 new 操作符创建 ejb 实例 4）访问快速高效

  web 服务

- **ejb 模块的文件结构**

  包括企业 bean 类，业务接口，帮助类



jpa生命周期

- **这些实体实例一共有四种状态，哪四种？分别有什么含义**

new：1）获得了一块内存空间，但并没有存入数据库，没有纳入 JPA EntityManager 的管理中。2）在数据库中不存在一条与它对应的记录 3）new：new 生成

managed：1）持久化对象，纳入 JPA EntityManager 管理。2）数据库中存在一条与之对应的数据，对其操作会影响数据库。3）new→managed:persist() managed:find()

detached：1）游离态对象，已经脱离 JPA EntityManager 的管理 2）数据库中可能还存在一条与它对应的记录，但对其操作不影响数据库 3）managed→detached: em.clear() detached→managed:merge()

removed:1）删除数据库中的记录，在适当时候被垃圾回收 2）remove()方法

- **安全的实现机制**

两种方式：声明式，编程技术

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

- **声明式的安全中，需要声明哪些与安全配置相关的信息**

1） **角色(role)**：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

2） **访问控制（security-constraint）**：规定特定角色能够访问什么样的资源，包括 web 资源、角色约束、用户数据约束

3） **认证机制（Authentication-Mechanisms)**：规定使用的认证方式

- **声明式安全的认证方式**

基于 HTTP 的认证：服务器向客户端请求用户名和密码，并验证是否为授权用户，是默认的认证方式，不关闭浏览器就不会清除 session。

基于表单的认证：访问需要认证的页面时返回登录表单，当用户通过认证后，才可以访问后续页面

基于 SSL 证书的认证：在客户端和服务器之间加密传输

- **基于 mvc 的 web 应用构架流程**

1） 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

- **web 层应该用什么样的技术，返回的模型可以用什么**

view 层:用户界面，html tags、jsp tags、XML/XSL

controller 层：接收用户动作，对数据做适当处理，servlet，

model 层：封装应用数据，处理商业逻辑，bean / java bean、InventorManager、InventorItem、ShipmentReceived