J2EE期末复习知识点汇总

J2EE期末复习知识点汇总

- 第零章：J2EE简介
  - J2EE客户端
  - Web客户端
  - Web组件
    - Servlet
    - JSP
  - J2EE容器
    - 什么是容器?
    - 容器能干什么?
  - Java核心技术
    - Servlet
    - JSP
  - Web应用程序
    - 定义
    - 存储
    - 组成
- 第一章：HTTP协议
  - HTTP请求包
  - HTTP应答包
  - HTTP请求方法
  - URI
  - URL
- 第二章：Servlet技术
  - Servlet
  - Servlet容器
  - Servlet接口
    - 包
    - 父类
    - 方法
  - 请求处理
  - 生命周期
    - 加载和实例化

初始化
处理请求
服务结束
部署Servlet
过滤器
过滤器特点
过滤器作用
过滤器的编程步骤
Filter对请求的过滤
Filter对响应的过滤
过滤器使用的注意事项
会话
会话定义
会话追踪机制
session的常用方法
URL路径解析
匹配规则

第零章：J2EE简介

J2EE 提供了一套设计、开发、汇编和部署企业应用程序的规范; 提供了企业级应用程序的开发平台，提供了分布式、基于组件、松耦合、安全可靠、独立于平台的应用程序环境;提供了开发企业级应用程序的技术架构。

J2EE客户端

由三部分组成:

1. Web 客户端
2. Applets(客户端小应用程序)
3. Application 客户端

Web客户端

由两部分组成:

1. 运行在 Web 层的 Web 组件生成的包含*各种标记语言* (HTML 、XML 等等) 的*动态 Web 页面* 。
2. 接收从服务器传送来的页面并将它显示出来的 ***Web浏览器***。

Web组件

既可以是 servlet也可以是 JSP页面。静态的 HTML页面，Applets 服务器端的功能类并不被 J2EE 规范视为 Web 组件。

Servlet

是一个 Java类，它可以动态地处理请求并作出响应。

JSP

是一个基于文本的文档，它以 servlet 的方式执行，但是它可以更方便建立静态内容。

J2EE容器

什么是容器?

容器为 J2EE 应用程序组件提供了运行时支持;

容器充当组件与支持组件的底层特定平台的功能之间的接口;

J2EE 服务器以容器的形式为每一个组件类型提供底层服务(如事务处理、状态管理、多线程、资源池等）。

容器能干什么?

容器是一个组件和支持组件的底层平台特定功能之间的接口。在一个 Web 组件、enterprise bean或者是一个应用程序客户端组件可以被执行前，它们必须被装配到一个 J2EE 应用程序中，并且部署到它们的容器。

装配:

1. 为 J2EE 应用程序中的每一个组件以及 J2EE 应用程序本身指定容器的设置。
2. J2EE 体系结构提供了可配置的服务意味着在**相同的 J2EE 应用程序**中的应用程序组件根据其被部署在什么在地方在**实际运行时会有所不同**。
3. 容器还管理诸如一个 enterprise bean和 servlet 的生存周期、数据库连接资源池等不能配置的服务。

Java核心技术

本节介绍Java Servlet技术和JSP技术。

Servlet

Servlet 是驻留在服务器上 Java 类，用于响应通过 HTTP 传入的请求。

JSP

Java 服务器页面允许程序员将 Servlet 代码写入基于文本的文档中。这些页面与 HTML 页面类似，只是它们还含有 Java 代码。

Web应用程序

本节介绍Web应用程序的定义、存储和组成。

定义

Web应用程序是 servlet、jsp页面、HTML页面、类和其他资源等的集合。

存储

1. web 归档文件， 以.war 扩展名结尾；
2. web 归档文件展开后的目录结构

组成

1. 协议(或称为服务方式)
2. 存有该资源的主机 IP 地址(有时也包括端口号)
3. 主机资源的具体地址，如目录和文件名等

第一章：HTTP协议

本章介绍HTTP协议的请求包、应答包、请求方法、URI和URL。

HTTP请求包

HTTP请求包由三个部分组成：

1. 方法-URI-协议/版本
2. 请求头
3. 请求正文

HTTP应答包

HTTP应答包由三个部分组成：

1. 协议 -状态代码 -描述
2. 应答头
3. 应答正文

HTTP请求方法

HTTP有七种请求方法：

1. GET
2. POST
3. HEAD
4. OPTIONS
5. PUT
6. DELETE
7. TRACE

URI

URI由三部分组成：

1. 访问资源的命名机制
2. 存放资源的主机名
3. 资源自身的名称，由路径表示

URL

URL由三部分组成：

1. 协议 + "//"
2. 存有该资源的主机 IP 地址(有时也包括端口号) + "/"
3. 主机资源的具体地址，如目录和文件名等

第二章：Servlet技术

Java Servlets是基于 Java技术的 Web 组件，用来扩展以请求 /响应为模型的服务器的能力，提供动态内容。

本章主要介绍Servlet、Servlet容器、Servlet接口、请求处理、生命周期、过滤器、会话追踪机制、URL路径解析。

Servlet

使用 Java Servlet 应用程序设计接口(API)及相关类和方法的 Java 程序。由容器或引擎来管理，通过请求 /响应模型与 Web客户进行交互。

Servlet容器

Servlet容器是Servlet的运行环境。

Servlet容器是Web服务器的一部分，管理和维护Servlet的整个生命周期。

Servlet容器必须支持HTTP协议，负责处理客户请求，把请求传送给适当的Servlet并把结果返回给客户。

Servlet接口

本节介绍Servlet接口所在的包、父类和实现方法。

包

javax.servlet.Servlet

父类

javax.servlet.http.HttpServlet

方法

```
public void init(ServletConfig config) throws ServletException
/* 一旦对 servlet 实例化后，容器就调用此方法。容器把一个 ServletConfig 对象传统给此方法， 这样 servlet 的实例就可以把与容器相关的配置数据保存起来供以后使用 */

public void service(ServletRequest req,ServletResponse res)throws ServletException,IOException
```

```
/* 成功初始化后此方法才能被调用处理用户请求。 前一个参数提供访问初始请求数据的方法
和字段， 后一个提供 servlet 构造响应的方法 */

public void destroy()
/* 容器可以在任何时候终止 servlet 服务 */

public ServletConfig getServletConfig()
/* 在 servlet 初始化时，容器传递进来一个 ServletConfig 对象并保存在 servlet
实例中，该对象允许访问两项内容：初始化参数和 ServletContext 对象 */

public String getServletInfo()
/* 此方法返回一个 String 对象，该对象包含 servlet 的信息 */
```

请求处理

1. Serlvet 接口只定义了一个服务方法就是 service，而 HttpServlet 类实现了该方法并且要求调用对应的 doXXX()

   方法。

2. 通常情况下，在开发基于 HTTP 的 servlet 时只需要关心 doGet 和 doPost 方法，其它的方法需要开发者非常的

   熟悉 HTTP 编程。

3. 通常情况下，实现的 servlet 都是从 HttpServlet 扩展而来。 doPut 和 doDelete 方法允许开发者支持 HTTP/1.1 的对应特性; doHead 是一个已经实现的方法，它将执行 doGet 但是仅仅向客户端返回 doGet 应该向客户端返回的头部的内容; doOptions 方法自动的返回 servlet 所直接支持的 HTTP 方法信息; doTrace方法返回 TRACE 请求中 的所有头部信息。

生命周期

加载和实例化

1. 容器负责加载和实例化一个 servlet。实例化和加载可以发生在引擎启动的时候，也可以推迟到容器需要该 servlet 为客户请求服务的时候(加载的时机) 。
2. 容器必须先定位 servlet 类，在必要的情况下，容器使用通常的 Java 类加载工具加

载该 servlet，可能是从本机文件系统，也可以是从远程文件系统甚至其它的网络服务。
3. 容器加载 servlet 类以后，它会实例化该类的一个实例。 需要注意的是可能会实例化多个实例， 例如一个 servlet 类因为有不同的初始参数而有多个定义，或者servlet实现 SingleThreadModel而导致容器为之生成一个实例池。

初始化

1. 容器必须在 servlet能够处理客户端请求前初始化它。
2. 初始化的目的是读取永久的配置信息。
3. 通过调用它的 init 方法并给它传递唯一的一个(每个 servlet 定义一个) ServletConfig 对象完成这个过程。
4. 该配置对象允许 servlet 访问容器的配置信息中的名称-值对(name-value)初始化参数。同时给 servlet 提供了访问实现了 ServletContext 接口的具体对象的方法，该对象描述了 servlet 的运行环境。

处理请求

1. 在 servlet 被适当地初始化后，容器就可以使用它去处理请求了。
2. 每一个请求由 ServletRequest类型的对象代表，而 servlet 使用 ServletResponse回应该请求。这些对象被作为 service 方法的参数传递给 servlet。
3. 在 HTTP 请求的情况下，容器必须提供代表请求和回应的 HttpServletRequest和HttpServletResponse的具体实现。 需要注意的是容器可能会创建一个 servlet 实例并将之放入等待服务的状态， 但是这个实例在它的生存期中可能根本没有处理过任何请求。

服务结束

1. 容器在能够调用 destroy 方法前，它必须允许那些正在 service 方法中执行的线程执行完或者在服务器定义的一段 时间内执行(这个时间段在容器调用 destroy 之前)。
2. 一旦 destroy 方法被调用，容器就不会再向该实例发送任何请求。如果容器需要再使用该 servlet，它必须创建新 的实例。 destroy 方法完成后，容器必须释放 servlet 实例以便它能够被垃圾回收。

部署Servlet

元素

1. servlet元素必须含有 servlet-name元素和 servlet-class元素，或者 servlet-name元素和 jsp-file 元素。
2. servlet-name 元素用来定义 servlet 的名称，该名称在整个应用中必须是惟一的。
3. servlet-class元素用来指定 servlet 的完全限定的类名称。
4. jsp-file 元素用来指定应用中 JSP文件的完整路径，这个完整路径必须由 /开始。
5. init-param 元素是可选元素，有 param-name, param-value两个子元素。

元素

1. 元素为一个 servlet 实例提供一个 URL pattern;
2. 必须包含 元素和 元素;
3. 必须和在 web.xml 文件某处 元素定义的 元素一致。

过滤器

1. 它能够对 Servlet 容器的请求和响应对象进行检查和修改。
2. 过滤器本身并不产生请求和响应对象，它只能提供过滤作用。
3. 过滤器在Servlet被调用之前检查并系应该Request对象，Request Header和Request Context；在Servlet被调用之后检查并修改Response对象。
4. 过滤器负责过滤的Web组件可以是Servlet、JSP或HTML文件，即所有静态的和动态的web资源。

过滤器特点

1. 检查和修改 ServletRequest和 ServletResponse对象;
2. 可以被指定和特定的 URL 关联，只有当客户请求访问该 URL 时，才会触发过滤器;
3. 可以被串联在一起， 形成管道效应， 协同修改请求和响应对象 (过滤器链)。

过滤器作用

1. 查询请求并作出相应的行动;
2. 修改请求的头部和数据;
3. 用户可以提供自定义的请求;
4. 修改响应的头部和数据;
5. 与外部资源进行交互。

过滤器的编程步骤

1. 建立一个实现Filter接口的类：所有的 Servlet 过滤器类都必须实现 javax.servlet.Filter 接口。这个接口含有 3个过滤器类必须实现的方法: init(,)、doFilter(,)、destroy（）。
2. 在doFilter方法中实现过滤：doFilter 方法为大多数过滤器的关键部分。每当调用一个过滤器时， 都要执行 doFilter。对于大多数过滤器来说， doFilter 执行的步骤是基于传入信息的。因此，可能要利用作为 doFilter 的第一个参数提供的 ServletRequest。 这个对象常常构造为 HttpServletRequest类型，以提供对该类的更特殊方法的访问。
3. 调用 FilterChain 对象的 doFilter 方法：Filter 接口的 doFilter 方法以一个 FilterChain 对象作为它的第三个参数。在调用该对象的 doFilter 方法时，激活下一个相关的过滤器。这个过程一般持续到链中最后一个过滤器为止。在最后一个过滤器调用其 FilterChain 对象的 doFilter 方法时，激活 servlet 或页面自身。但是，链中的任意过滤器都可以通过不调用其 FilterChain 的 doFilter 方法中断这个过程。 在这样的情况下， 不再调用 JSP页面的 serlvet，并且中断此调用过程的过滤器负责将输出提供给客户机。
4. 对相应的 servlet和 JSP页面注册过滤器：部署描述符文件的 2.3 版本引入了两个用于过滤器的元素，分别是:filter 和 filter-mapping 。 filter 元素向系统注册一个过滤对象，filter-mapping 元素指定该过滤对象所应用的 URL 。
5. 禁用激活器 servlet。

Filter对请求的过滤

1. Servlet容器创建一个Filter实例;
2. 过滤器实例调用 init 方法，读取过滤器的初始化参数;
3. 过滤器实例调用 doFilter 方法，根据初始化参数的值判断该请求是否合法;
4. 如果该请求不合法则阻塞该请求

5. 如果该请求合法则调用 chain.doFilter 方法将该请求向后续传递

Filter对响应的过滤

1. 过滤器截获客户端的请求
2. 重新封装 ServletResponse，在封装后的 ServletResponse中提供用户自定义的输出流
3. 将请求向后续传递
4. 从封装后的 ServletResponse中获取用户自定义的输出流
5. 将响应内容通过用户自定义的输出流写入到缓冲流
6. 在缓冲流中修改响应的内容后清空缓冲流，输出响应内容

过滤器使用的注意事项

1. 由于 Filter、 FilterConfig 、FilterChain 都是位于 javax.servlet 包下，并非 HTTP 包所特有的，所以其中所用到 的请求、响应对象ServletRequest、ServletResponse在使用前都必须先转换成 HttpServletRequest、HttpServletResponse再进行下一步操作。
2. 在web.xml中配置Servlet和Servlet过滤器，应先声明过滤元素，再声明Servlet元素。

会话

本章从会话定义、会话追踪机制、Session常用方法、URL路径解析来介绍Servlet会话。

会话定义

会话是指一个用户在客户端登录，为达到某个目的与服务器端进行多次交互，最后退出应用系统的全过程。

会话追踪机制

Servlet 的会话追踪机制是基于 Cookie 或 URL 重写技术，融合了这两种技术的优点。当客户端允许使用 Cookie 时，内建 session对象使用 Cookie 进行会话追踪; 如果客户端禁用 Cookie，则选择使 用 URL 重写。

1. 创建会话：HttpSession session=request.getSession(true)
2. 在会话中保存数据的方法： setAttribute(String s, Object o)
3. 从会话提取原来所保存对象的方法： getAttribute(String s)
4. 关闭会话 HttpSession.invalidate()

session的常用方法

1. getAttribute():从 session中获取以前存储的值
2. getAttributeNames():返回 session中所有属性的名称
3. setAttribute():将键与值关联起来，存储进 session
4. removeAttribute():删除 session中存储的对应键的值
5. invalidate():删除整个 session及其存储的键值
6. logout():注销当前用户
7. getId():获取每个 session对应的唯一 ID
8. getCreationTime():获取 session创建的时间
9. getLastAccessedTime(:)获取session最后被访问的时间

URL路径解析

通常由 servlet 容器来解析一个指向 servlet 的 URL 。解析分两步， 1. 标识网络应用; 2. 定位具体的 servlet。上述两个步骤都会对 URL 中的 URI(除去主机名以外的) 部分进行分段处理， 形成三个部分， Context Path、 Servlet Path、Path Info。HttpServletRequest提供了三个方法- getContextPath(),getServletPath(和) getPathInfo()分别提取不同段内容。

匹配规则

1. Servlet容器先将整个 URI(除去 context path之后的)和 servlet mapping进行匹配。
2. 如果匹配成功， 则除 context path以外的剩余部分都是 servlet path。因此， path info 部分为空。
3. 以/为分界符倒着往前去和 servlet mapping匹配。如匹配成功，匹配的部分就是

servlet path，剩余部分是 path info。

4. 如果 URI 最后是某种文件扩展名， 则 servlet 容器去和 servlet mapping 匹配。如成功，则将整个 UR(I 除去 context path 之后的)视为 servlet path，而 path info 为空。
5. 如果始终没找到相匹配的 servlet mapping，则将请求发往默认 servlet。如果不存在默认 servlet，则 servlet 容器发送错误消息，指示 servlet 没找到 .