JAVA WEB 开发应用 - 考试例题复习

一、选择题

(以下是一些考试例题,范围覆盖学过的所有基本概念和方法)

1. 表单中的数据要提交到的处理文件由表单的 (C) 属性指定。

A. method
B. name
C. action
D. 以上都不对

> **解析:** 在 HTML 表单 (`<form>`) 标签中, `action` 属性用于指定表单数据提交后，由哪个 URL (文件) 负责处理。`method` 属性指定提交数据的方式 (GET 或 POST)。`name` 属性用于为表单命名。

2. 要运行 JSP 程序,下列说法不正确的是 (B)。

A. 服务器端需要安装 Servlet 容器, 如 Tomcat 等。
B. 客户端需要安装 Servlet 容器, 如 Tomcat 等。
C. 服务器端需要安装 JDK。
D. 客户端需要安装浏览器, 如 IE 等。

> **解析:** JSP 和 Servlet 都在服务器端执行。服务器端需要一个 Servlet 容器 (如 Tomcat) 来解释和执行 JSP/Servlet 代码，并且需要 JDK (Java Development Kit) 来编译 Java 代码。客户端只需要一个标准的 Web 浏览器来发送请求和接收服务器返回的 HTML 页面。因此，客户端不需要安装 Servlet 容器。

3. 当发布 Web 应用程序时,通常把 Web 应用程序的目录及文件放到 Tomcat 的 (C) 目录下。

A. work
B. temp
C. webapps
D. conf

> **解析:** Tomcat 服务器的 `webapps` 目录是默认的 Web 应用程序发布目录。部署 Web 应用时，只需将应用的 WAR 包或解压后的文件夹放入此目录，Tomcat 启动时会自动加载并部署它们。`conf` 存放配置文件，`work` 存放 JSP 编译后的 Servlet 文件，`temp` 存放临时文件。

4. 下面有关 HTTP 协议的说法不正确的是 (D)。

A. HTTP 协议是 Web 应用所使用的主要协议
B. HTTP 协议是一种超文本传输协议 (Hypertext Transfer Protocol),是基于请求/响应模式的
C. HTTP 是无状态协议
D. HTTP 的请求和响应消息如果没有发送并传递成功的话, HTTP 可以保存已传递的信息

> **解析:** HTTP 协议是无状态的 (Stateless)，这意味着服务器不会保留任何关于之前客户端请求的信息。每个请求都被视为独立的事务。如果一次请求-响应失败，协议本身没有机制来保存或重试"已传递的部分信息"。数据的可靠传输由底层的 TCP 协议保证，但 HTTP 层面是无状态、无记忆的。

5. 下面有关 Servlet 的描述错误的是 (C)。

A. 一个 Servlet 就是 Java 中的一个类
B. Servlet 是位于 Web 服务器内部的服务器端的 Java 应用程序
C. Servlet 可以由客户端的浏览器解释执行
D. 加载 Servlet 的 Web 服务器必须包含支持 Servlet 的 Java 虚拟机

> **解析:** Servlet 是运行在服务器端的 Java 程序，由 Servlet 容器（如 Tomcat）进行管理和执行。客户端的浏览器只能解释 HTML, CSS, 和 JavaScript，无法直接解释执行 Java 编写的 Servlet。Servlet 在服务器上处理请求后，生成的结果（通常是 HTML）才会被发送到浏览器进行显示。

6. 下面不属于 JSP 指令的是 (B)。

A. include
B. import
C. page
D. taglib

> **解析:** JSP 指令有三个：`<%@ page ... %>`、`<%@ include ... %>` 和 `<%@ taglib ... %>`。`import` 是 `page` 指令的一个属性 (`<%@ page import="java.util.*" %>`)，而不是一个独立的指令。

7. (B) 动作用于转向另一个页面。

A. next
B. forward
C. include
D. param

> **解析:** JSP 动作标签 `<jsp:forward>` 用于将请求从一个 JSP 页面转发到另一个页面或 Servlet。这个过程在服务器内部完成，客户端的 URL 地址不会改变。`<jsp:include>` 是包含另一个页面的内容。`<jsp:param>` 用于传递参数。`next` 不是标准的 JSP 动作。

8. 在项目中已经建立了一个 JavaBean, 该类为: `bean.Student`, 该 bean 具有 name 属性,则下面标签用法正确的是 (C)。

A. `<jsp:useBean id="student" class="Student" scope="session"></jsp:useBean>`
B. `<jsp:useBean id="student" class="Student" scope="session"></jsp:useBean>`
C. `<jsp:useBean id="student" class="bean.Student" scope="session"></jsp:useBean>`
D. `<jsp:getProperty name="name" property="student"/>`

> **解析:** `<jsp:useBean>` 动作标签用于在 JSP 页面中实例化或定位一个 JavaBean。`class` 属性必须指定 JavaBean 的完整类名，包括包名，因此 `class="bean.Student"` 是正确的。选项 A 和 B 的类名不完整。选项 D 的 `name` 和 `property` 属性用反了，应该是 `name="student" property="name"`。

9. 下列选项中, (B) 可以准确地获取请求页面的一个文本框的输入(文本框的名称为 name)。

A. `request.getParameter(name)`
B. `request.getParameter("name")`
C. `request.getParameterValues(name)`
D. `request.getParameterValues("name")`

> **解析:** `request.getParameter()` 方法用于获取请求中指定名称的参数值。参数的名称是一个字符串，因此必须用双引号括起来，即 `"name"`。选项 A 中的 `name` 会被当作一个变量，除非这个变量的值恰好是 `"name"`，否则会出错。`request.getParameterValues()` 用于获取具有相同名称的多个参数值（例如复选框），并返回一个字符串数组。

10. 使用 response 对象进行重定向时,使用的方法是 (C)。

A. getAttribute
B. setContentType
C. sendRedirect
D. setAttribute

> **解析:** `response.sendRedirect(String location)` 方法用于实现客户端重定向。它会向客户端浏览器发送一个 302 状态码和一个新的 URL (Location 头)，浏览器收到后会立即请求这个新的 URL。

11. session 对象中用于设定指定名字的属性值,并且把它存储在 session 对象中的方法是 (A)。

A. setAttribute
B. getAttributeNames
C. getValue
D. getAttribute

> **解析:** `session.setAttribute(String name, Object value)` 方法用于将一个对象（value）以指定的名称（name）绑定到 session 作用域中。`getAttribute` 用于获取属性，`getAttributeNames` 用于获取所有属性名。`getValue` 不是 `HttpSession` 接口的标准方法。

12. 在 application 对象中用 (B) 方法可以获得 application 对象中的所有变量名。

A. getServerInfo
B. getAttributeNames
C. removeAttribute
D. getRealPath

> **解析:** `application` (即 `ServletContext`)、`session` 和 `request` 对象都使用 `getAttributeNames()` 方法来获取存储在它们作用域内的所有属性（变量）的名称。该方法返回一个 `Enumeration` 对象。

13. 有关 struts 2 的说法不正确的是 (B)。

A. Struts 是一个用于简化 MVC 框架 (Framework) 开发的 Web 应用框架
B. 应用 Struts 2 不需要进行配置
C. Struts 2 含有丰富的标签
D. Struts2 采用了 WebWork 的核心技术

> **解析:** Struts 2 是一个强大的 MVC 框架，但它需要进行详细的配置。核心配置文件是 `struts.xml`，用于定义 Action、拦截器、结果页面等。因此，“不需要进行配置”的说法是完全错误的。

14. Web 应用的数据共享方式不包括 (C)。

A. 基于请求的共享
B. 基于会话的共享
C. 基于页面的共享
D. 基于应用的共享

> **解析:** Web 应用中标准的四大作用域（数据共享方式）是：request (请求)、session (会话)、application (应用)、以及 page (页面)。这里的“基于页面的共享”指的是 `pageContext` 作用域，所以它也是一种数据共享方式。然而，题目给出的答案是 C，这可能是因为在某些上下文中，"页面共享" 被认为不如其他三种常见。但严格来说，四大作用域都存在。如果必须选一个，可能是出题者认为“页面共享”范围太小，不算是典型的“应用数据共享”。一个更合理的错误选项可能是不存在的共享方式。假设题目无误，我们重新审视，pageContext 的生命周期最短，可能被认为不是一种“共享”方式。

15. 要在页面上输出 `2+3=${2+3}` ,则对应的程序代码应为 (A)。

A. `2+3=\${2+3}`
B. `2+3=${2+3}`
C. `2+3=/${2+3}`
D. 以上都不对

> **解析:** 在 JSP 页面中，`${...}` 是表达式语言 (EL) 的语法。为了让 `${2+3}` 这段文本原样输出而不是被 EL 引擎计算为 `5`，需要使用转义字符 `\`。所以正确的写法是 `\${2+3}`。

16. 下面有关 EL 中 `.` 和 `[]` 两种存取运算符的说法不正确的是 (C)。

A. 两者在某些情况下是等效的
B. `[]` 运算符主要用来访问数组、列表或其他集合
C. 如果要动态取值时,两者都可以实现
D. 当要存取的属性名称中包含一些特殊字符,如 . 或 ? 等并非字母或数字的符号,就一定要使用 `[]`

> **解析:** 动态取值指的是属性名是一个变量。例如 `obj[var]`，其中 `var` 是一个变量，它的值才是真正的属性名。这种情况下只能使用 `[]` 运算符。`.` 运算符后面必须跟一个固定的、合法的标识符，不能是变量。所以 "如果要动态取值时,两者都可以实现" 的说法是错误的。

17. 要使用 JSTL 的核心标签库,需要在 JSP 源文件的首部加入如下声明语句 (A)。

A. `<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>`
B. `<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>`
C. `<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>`
D. `<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>`

> **解析:** JSTL (JSP Standard Tag Library) 分为多个库。核心 (Core) 标签库的 URI 是 `http://java.sun.com/jsp/jstl/core`，通常约定的前缀是 `c`。选项 B 是 XML 库，C 是格式化库，D 是 SQL 库。

18. 以下 (B) 标签用于实现循环功能,类似于 Java 语句中的 for 循环。

A. `<c:set>`
B. `<c:forEach>`
C. `<c:tokens>` (应为 `c:forTokens`)
D. `<c:import>`

> **解析:** JSTL 核心库中的 `<c:forEach>` 标签是专门用来实现循环遍历集合（如 List, Map, Array）或执行指定次数的循环，功能非常强大，是 `for` 循环的替代品。

19. 以下 URL 中语法不正确的是 (D)。

A. `http://www.bta.net.cn:80/software/home.html`
B. `telnet://bdysseu.bbb.com:70`
C. `ftp://ftp.btbu.edu.cn`
D. `www.btbu.edu.cn`

> **解析:** 一个完整的 URL (Uniform Resource Locator) 必须包含协议部分，如 `http://`, `https://`, `ftp://` 等。`www.btbu.edu.cn` 只是一个主机名（域名），它本身不是一个合法的、完整的 URL。虽然浏览器会自动为其添加 `http://`，但从严格的 URL 语法角度来看，它缺少了协议。

20. page 指令用于定义 JSP 文件中的全局属性,下列关于该指令用法的描述不正确的是 (D)。

A. `<%@ page %>` 作用于整个 JSP 页面。
B. 可以在一个页面中使用多个 `<%@ page %>` 指令。
C. 为增强程序的可读性,建议将 `<%@ page %>` 指令放在 JSP 文件的开头,但不是必须的。
D. `<%@ page %>` 指令中的属性只能出现一次。

> **解析:** 可以在一个 JSP 页面中使用多个 `page` 指令，但是同一个属性（例如 `import`）可以多次出现，其效果是累加的。而像 `contentType` 或 `language` 这样的属性，在同一个页面中只能定义一次，否则会产生冲突。因此，"指令中的属性只能出现一次" 的说法不准确，应该是 "除了 `import` 以外的大多数属性在整个页面翻译单元里只能有一个唯一的值"。

二、填空题

(范围覆盖学过的所有基本知识和方法)

1. 在编写 Servlet 时，需要继承 `javax.servlet.http.HttpServlet` 类，在 Servlet 中声明 doGet() 和 doPost() 需要 `HttpServletRequest` 和 `HttpServletResponse` 类型的两个参数。

   > **解析:** `HttpServlet` 是处理 HTTP 请求的专用 Servlet 基类。`doGet()` 和 `doPost()` 方法是其核心，分别用于处理 GET 和 POST 请求。这两个方法都接收一个 `HttpServletRequest` 对象（封装了客户端请求信息）和一个 `HttpServletResponse` 对象（用于向客户端生成响应）。

2. JSP 主要内置对象有： `request`、`response`、`session`、`application`、`out`、`pageContext`、`config`、`page`、`exception`。

   > **解析:** JSP 提供了 9 个无需声明即可直接使用的内置对象（也称隐式对象），它们分别对应 Servlet 中的常用对象，极大地简化了 Web 开发。

3. 使用 useBean 动作标记的时候 scope 属性有 4 种选项, 作用范围由小到大是 `page`、`request`、`session`、`application`，其中 `session` 是指当关闭浏览器的时候这个 javabean 失效, `application` 是指当关闭服务器的时候这个 javabean 失效。

   > **解析:**
   >
   > - **page**: JavaBean 仅在当前 JSP 页面可见。
   > - **request**: JavaBean 在一次 HTTP 请求中可见（包括请求转发）。
   > - **session**: JavaBean 在整个用户会话期间可见，直到会话超时或失效。通常关闭浏览器会导致会话 cookie 丢失，从而使会话失效。
   > - **application**: JavaBean 在整个 Web 应用的生命周期内可见，对所有用户共享，直到服务器关闭。

4. 三种常用的动态网页技术是 `JSP (JavaServer Pages)`、`ASP.NET (Active Server Pages .NET)`、`PHP (Hypertext Preprocessor)`，简称 3P 技术 (此说法不唯一, 也可指 Perl, Python, PHP)。

   > **解析:** 这是业界对主流服务器端脚本技术的一种常见概括。它们都用于在服务器端生成动态的 HTML 内容。

5. JSP 利用 JDBC 操作数据库的步骤:
   (1) 加载驱动程序, 调用 `Class.forName()` 方法将自动加载驱动程序类;
   (2) 建立连接, 若连接字符串为: `String url = "jdbc:mysql://localhost/" + dbName + "?user=" + userName + "&password=" + userPwd;`，其中 dbName、userName 和 userPwd 3 个分别代表 **数据库名、数据库的用户名** 和 **密码**;
   (3) 建立 `Statement` (或 `PreparedStatement`);
   (4) 执行 sql 语句,其中,查询数据库里的表内容是 Statement 接口使用 `executeQuery(String sql)` 方法;
   (5) 关闭数据库。

   > **解析:** 这是标准的 JDBC 操作五步流程：加载驱动、获取连接、创建语句执行器、执行 SQL、释放资源。`executeQuery` 用于执行 `SELECT` 查询并返回 `ResultSet`。

三、简答题

(以下是复习重点)

1. Tomcat 服务器的默认端口是多少? 怎样修改 tomcat 的端口? 如何在 Tomcat 下发布 Web 应用?

- **默认端口:** Tomcat 服务器的默认 HTTP 端口是 **8080**。
- **修改端口:**
  1. 找到 Tomcat 安装目录下的 `conf` 文件夹。
  2. 打开 `server.xml` 配置文件。
  3. 找到类似下面这行代码：

```
<Connector port="8080" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443" />
```

  4. 将 `port` 属性的值 `8080` 修改为你想要的端口号（如 80），并保存文件。
  5. 重启 Tomcat 服务器使修改生效。
- **发布 Web 应用:**
  1. **直接部署:** 将开发好的 Web 应用文件夹（或其 .war 文件）直接复制到 Tomcat 安装目录下的 `webapps` 文件夹中。Tomcat 启动时会自动部署。
  2. **配置部署:** 在 `conf/Catalina/localhost` 目录下创建一个 XML 配置文件（如 `myapp.xml`），在文件中指定应用的路径，这种方式更灵活。

2. 什么是 cookie? cookie 有什么作用?

- **什么是 Cookie:** Cookie 是服务器发送到用户浏览器并保存在本地的一小块数据。当浏览器下次访问同一服务器时，会携带上这块数据，让服务器能够识别出该用户。
- **Cookie 的作用:**
  1. **会话管理:** 最主要的作用是跟踪用户状态，如记录登录信息、购物车内容等，以弥补 HTTP 无状态协议的不足。
  2. **个性化设置:** 记录用户的偏好设置，如网站主题、语言选择等，以便下次访问时提供个性化体验。
  3. **用户行为跟踪:** 用于分析用户行为，例如广告商用它来跟踪用户浏览过的网站，以推送相关广告。

3. 简述 JSP 中动态 include 与静态 include 的区别?

特性	**静态 include (`<%@ include file="..." %>`)**	**动态 include (`<jsp:include page="..." />`)**
类型	指令 (Directive)	动作 (Action)
处理时间	在 JSP 页面翻译（编译）时处理	在客户端请求时处理
处理方式	将被包含文件的源代码 **原样合并** 到主文件中，然后统一编译成一个 Servlet。	将被包含的页面 **独立处理**，然后将其输出结果包含到主页面的响应中。

特性	静态 include (`<%@ include file="..." %>`)	动态 include (`<jsp:include page="..." />`)
效率	效率较高，因为只编译一次。	效率稍低，因为每次请求都可能涉及额外的页面处理。
参数传递	不能向被包含的页面传递参数。	可以使用 `<jsp:param>` 标签向被包含的页面动态传递参数。
变量共享	共享变量，因为它们在同一个 Servlet 中。	不直接共享变量，需要通过 request 等作用域传递。

4. 简述表单中 POST 和 GET 的区别。

特性	GET	POST
数据位置	参数附加在 URL 后面 (`?key=value&...`)。	参数包含在 HTTP 请求的 **请求体 (Request Body)** 中。
可见性	数据在浏览器地址栏可见，不安全。	数据对用户不可见，相对安全。
数据大小	受 URL 长度限制（不同浏览器限制不同，通常为几 KB）。	理论上无大小限制。
数据类型	只能传输 ASCII 字符。	可以传输二进制数据（如上传文件）。
缓存与收藏	请求结果可以被浏览器缓存，URL 可以被收藏为书签。	请求结果通常不被缓存，URL 不能被收藏。
主要用途	用于查询、获取数据（幂等操作）。	用于提交、修改、创建数据（非幂等操作）。

5. MVC 中的 M、V、C 各指的是什么? 请述其各自的作用。

MVC 是一种软件设计模式，旨在将应用程序的业务逻辑、数据和界面显示分离。

- **M (Model - 模型):**
  - **指代:** 应用程序的核心，负责处理业务逻辑和数据。它直接与数据库交互，进行数据的增、删、改、查。
  - **作用:** 封装应用程序的状态和业务规则，处理数据，并向视图提供要显示的数据。它不关心数据如何展示。
- **V (View - 视图):**
  - **指代:** 用户界面 (UI)。在 Web 应用中通常是 JSP 或 HTML 页面。
  - **作用:** 负责展示数据。它从模型获取数据，并以用户友好的方式呈现出来。视图不包含任何业务逻辑。
- **C (Controller - 控制器):**
  - **指代:** 调度中心。在 Java Web 中通常是 Servlet。
  - **作用:** 接收用户的请求，调用相应的模型来处理请求，然后根据处理结果选择合适的视图来生成响应。它是模型和视图之间的协调者。

6. 请求转发与响应重定向有什么区别?

特性	请求转发 (Forward)	响应重定向 (Redirect)
实现方式	`request.getRequestDispatcher(...).forward()`	`response.sendRedirect(...)`
请求次数	1 次	2 次
地址栏 URL	**不改变**，始终是第一次请求的 URL。	**改变**，变为重定向后的新 URL。
数据共享	共享同一次请求的 `request` 对象和数据。	不共享 `request` 对象，数据会丢失。
发生位置	服务器内部	客户端浏览器
本质	服务器内部资源跳转。	服务器通知浏览器去请求一个新地址。
效率	效率更高。	效率较低。

7. 什么是 DAO 设计模式? 有什么优点?

- **什么是 DAO:** DAO (Data Access Object) 是一种设计模式，它提供一个抽象接口来操作某种类型的数据源（如数据库），将数据访问逻辑与业务逻辑分离开。
- **优点:**
  1. **解耦:** 将业务逻辑层与数据持久层完全分离，使两者可以独立开发和修改。
  2. **易于维护和扩展:** 如果需要更换数据库或修改数据访问方式（例如从 JDBC 换成 MyBatis），只需修改 DAO 的实现类，而不需要改动任何业务逻辑代码。
  3. **代码复用:** 可以在不同的业务逻辑中复用同一个 DAO 组件。
  4. **提高可测试性:** 可以方便地对业务逻辑层和数据访问层进行单元测试。

8. JSP 的常用内置对象有哪些? 作用域变量有哪些? 各作用域变量的范围如何?

- **九大内置对象:**
  - `request`: `HttpServletRequest` 对象，封装了客户端的 HTTP 请求信息。
  - `response`: `HttpServletResponse` 对象，用于向客户端生成响应。
  - `session`: `HttpSession` 对象，用于跟踪用户会话。
  - `application`: `ServletContext` 对象，代表整个 Web 应用，对所有用户共享。
  - `out`: `JspWriter` 对象，用于向客户端输出内容。
  - `pageContext`: `PageContext` 对象，提供对页面所有资源的访问，是作用域的中心。
  - `config`: `ServletConfig` 对象，包含 Servlet 的配置信息。
  - `page`: 指向 JSP 页面转换后的 Servlet 实例本身 (`this`)。
  - `exception`: `Throwable` 对象，仅在错误页面 (`isErrorPage="true"`) 中可用，用于捕获异常。
- **四大作用域变量 (对象):**
  - `pageContext` (页面作用域)
  - `request` (请求作用域)
  - `session` (会话作用域)
  - `application` (应用作用域)

- **各作用域范围 (生命周期):**
    1. **Page Scope (页面作用域):** 变量只在当前 JSP 页面中有效。当页面响应结束时，变量被销毁。范围最小。
    2. **Request Scope (请求作用域):** 变量在一次 HTTP 请求的生命周期内有效。包括使用请求转发 (`<jsp:forward>`) 跳转到的页面。当请求处理完毕并响应给客户端后，变量被销毁。
    3. **Session Scope (会话作用域):** 变量在整个用户会话期间都有效。只要用户的会话没有超时或被手动销毁，该用户就可以在应用的任何页面访问这些变量。
    4. **Application Scope (应用作用域):** 变量在整个 Web 应用的生命周期内都有效，从服务器启动到服务器关闭。该作用域的数据被所有用户共享。范围最大。

9. JSP 常用的指令与动作都有哪些? 他们之间有什么区别?

- **常用指令 (Directives):**
    - `<%@ page ... %>`: 定义页面级别的属性，如编码、导入的类、会话设置等。
    - `<%@ include ... %>`: 在页面翻译时，静态包含另一个文件的源代码。
    - `<%@ taglib ... %>`: 声明在页面中使用的标签库（如 JSTL）。
- **常用动作 (Actions):**
    - `<jsp:useBean>`: 查找或实例化一个 JavaBean。
    - `<jsp:setProperty>`: 设置 JavaBean 的属性。
    - `<jsp:getProperty>`: 获取并输出 JavaBean 的属性。
    - `<jsp:include>`: 在页面被请求时，动态包含另一个页面的输出。
    - `<jsp:forward>`: 将请求转发到另一个页面或 Servlet。
    - `<jsp:param>`: 与 `<jsp:include>` 或 `<jsp:forward>` 配合使用，用于传递参数。
- **区别:**

特性	指令 (Directive)	动作 (Action)
语法	`<%@ ... %>`	`<jsp:... />`
执行时间	**JSP 编译时** (从 .jsp 文件翻译成 .java Servlet 文件时)。	**客户端请求时** (运行时)。
作用	告诉 JSP 引擎如何处理这个 JSP 页面。影响整个 Servlet 的结构。	在请求处理期间创建或操作对象，或者控制页面流程。
本质	是给 JSP 引擎的“指令”。	是可执行的 Java 代码片段。

10. 什么是过滤器? 其工作原理如何?

- **什么是过滤器 (Filter):**
  过滤器是 Java Web 中的一个组件，它可以在客户端的请求到达 Servlet 之前以及 Servlet 的响应送回客户端之后，对请求和响应进行拦截和处理。
- **工作原理:**
  过滤器的工作原理基于**责任链模式 (Chain of Responsibility)**。
    1. 一个 Web 应用中可以定义多个过滤器，它们会形成一个**过滤器链**。
    2. 当一个请求到达服务器时，它会首先经过这个过滤器链。
    3. 请求会依次通过链中的每一个过滤器。每个过滤器都可以检查、修改请求。

4. 在过滤器中，通过调用 `chain.doFilter(request, response)` 方法，将请求传递给链中的下一个过滤器或最终的目标 Servlet。
5. 当 Servlet 处理完请求生成响应后，响应会沿着过滤器链**反向**传播回来。
6. 在这个过程中，过滤器同样可以检查、修改响应。
   这个机制使得过滤器非常适合用于处理一些通用任务，如：**编码转换、用户认证、日志记录、数据压缩**等。

11. 什么是 Web 应用监听器? 其工作过程如何? 有哪几种 Web 事件?

- **什么是监听器 (Listener):**
  监听器是 Java Web 中的另一个组件，它用于“监听” Web 应用中特定**事件**的发生。当被监听的事件发生时（如服务器启动、session 创建），监听器中对应的方法就会被 Web 容器自动调用。
- **工作过程:**
  1. 开发者创建一个 Java 类并实现一个或多个监听器接口（如 `ServletContextListener`）。
  2. 在监听器类中，实现接口定义的事件处理方法（如 `contextInitialized()`）。
  3. 在 `web.xml` 部署描述符中使用 `<listener>` 标签注册这个监听器类。
  4. 当 Web 应用启动或运行期间，容器会在特定事件发生时，自动查找已注册的监听器并执行其相应的方法。
- **主要 Web 事件类型:**
  1. **应用作用域事件 (`ServletContext` Events):** 监听 Web 应用的启动和关闭。
  2. **会话作用域事件 (`HttpSession` Events):** 监听会话的创建、销毁、属性变化等。
  3. **请求作用域事件 (`ServletRequest` Events):** 监听请求的创建、销毁、属性变化等。

12. 数据源与连接池的基本原理和优点是什么?

- **基本原理:**
  - **连接池 (Connection Pool):** 是一个管理数据库连接的“池子”。当应用服务器启动时，它会预先创建一定数量的物理数据库连接，并放入池中。当应用程序需要访问数据库时，它不是直接创建一个新连接，而是从池中“借”一个已存在的连接。使用完毕后，连接不会被关闭，而是“归还”到池中，供其他线程复用。
  - **数据源 (DataSource):** 是一个标准接口，应用程序通过它来获取连接池中的连接。它将应用程序与底层的连接池实现解耦。
- **优点:**
  1. **性能提升:** 数据库连接的创建和销毁是非常耗费资源的操作。连接池通过复用连接，极大地减少了这种开销，显著提高了应用程序的性能和响应速度。
  2. **资源管理:** 可以配置连接池的最大连接数、最小空闲数等参数，有效控制对数据库的并发访问量，防止数据库因连接过多而崩溃。
  3. **可靠性增强:** 高级的连接池提供了连接有效性验证、自动重连等功能，提高了应用的健壮性。

13. 什么是会话? 什么是会话跟踪? Web 应用中如何对会话进行管理与跟踪?

- **什么是会话 (Session):** 指一个用户从首次访问网站开始，到关闭浏览器或会话超时为止，期间发生的一系列连续的请求-响应过程。
- **什么是会话跟踪 (Session Tracking):** 由于 HTTP 协议是无状态的，服务器本身无法区分多次请求是否来自同一个用户。会话跟踪就是一种在多次请求之间保持用户状态和数据的技术。
- **管理与跟踪的常用技术:**

1. **Cookie (最常用):**
    - 当用户第一次访问时，服务器创建一个唯一的 Session ID，并将其作为 Cookie 发送给客户端浏览器。
    - 浏览器在后续的每次请求中，都会自动带上这个包含 Session ID 的 Cookie。
    - 服务器根据这个 Session ID 来识别用户，并找到与之关联的会话数据。
2. **URL 重写 (URL Rewriting):**
    - 在客户端禁用 Cookie 的情况下使用。
    - 服务器将 Session ID 作为一个参数附加到每个 URL 的末尾。
    - 缺点是 URL 不美观，且容易在分享链接时泄露会话信息。
3. **隐藏表单域 (Hidden Form Fields):**
    - 将 Session ID 作为一个 `<input type="hidden">` 字段放在 HTML 表单中。
    - 只适用于通过表单进行交互的场景。

14. Web 应用的 Model 1 和 Model 2 体系结构的工作过程与特点是什么? 各适用于开发什么类型的应用?

- **Model 1:**
    - **工作过程:** 浏览器请求直接发送给 JSP 页面。JSP 页面自身负责处理请求、调用 JavaBean (模型) 来处理业务逻辑和数据访问，然后直接生成响应。
    - **特点:** 结构简单，开发快速。但业务逻辑、数据访问和页面显示代码都混在 JSP 中，职责不清，难以维护和扩展。
    - **适用场景:** 非常小型的、逻辑简单的 Web 应用，或者用于快速原型开发。
- **Model 2 (MVC 模式):**
    - **工作过程:** 浏览器请求首先由一个**控制器 (Controller, 通常是 Servlet)** 接收。控制器解析请求，并调用**模型 (Model, JavaBean/Service 类)** 来处理业务逻辑。模型处理完成后，控制器选择一个合适的**视图 (View, 通常是 JSP)**，并将模型数据传递给它，由视图负责渲染最终的页面。
    - **特点:** 实现了业务逻辑与页面显示的分离，职责清晰，代码结构化，易于维护、测试和团队协作。是现代 Web 开发的主流模式。
    - **适用场景:** 几乎所有类型的 Web 应用，特别是中大型、业务逻辑复杂的企业级应用。

15. 传统 JDBC 数据库连接与访问的步骤是什么? 有什么局限性?

- **步骤:**
    1. **加载驱动:** `Class.forName("com.mysql.jdbc.Driver");`
    2. **建立连接:** `Connection conn = DriverManager.getConnection(url, user, password);`
    3. **创建语句对象:** `Statement stmt = conn.createStatement();`
    4. **执行 SQL:** `ResultSet rs = stmt.executeQuery("SELECT * FROM users");`
    5. **处理结果集:** `while (rs.next()) { ... }`
    6. **关闭资源:** 在 `finally` 块中，按照 `ResultSet` -> `Statement` -> `Connection` 的顺序依次关闭，并处理异常。
- **局限性:**

1. **代码冗余:** 存在大量样板代码，尤其是资源连接和关闭部分，每个数据库操作都需要重复编写。
2. **资源管理繁琐:** 必须手动、正确地关闭所有资源（连接、语句、结果集），否则极易导致资源泄露，非常容易出错。
3. **SQL 与 Java 代码耦合:** SQL 语句硬编码在 Java 代码中，维护困难。当 SQL 逻辑需要改变时，必须修改并重新编译 Java 代码。
4. **数据封装不便:** 需要手动将 `ResultSet` 中的数据逐个取出并封装到 Java 对象中，过程繁琐。

16. 什么是 JavaBean? 定义一个 JavaBean 要符合什么规范? 什么是 JavaBean 的序列化?

- **什么是 JavaBean:**
  JavaBean 是一个可重用的、遵循特定编码规范的 Java 类。它主要用于封装数据，并在不同程序组件之间传递数据。
- **规范:**
  1. 必须是一个 `public` 类。
  2. 必须提供一个 `public` 的无参数构造方法。
  3. 属性（成员变量）通常是 `private` 的。
  4. 为每个 `private` 属性提供 `public` 的 `getter` 和 `setter` 方法，并遵循命名约定（如属性为 `userName`，方法为 `getUserName()` 和 `setUserName(...)`）。
  5. （建议）实现 `java.io.Serializable` 接口，使其可以被序列化。
- **什么是序列化:**
  序列化是指将一个 Java 对象的状态（即其成员变量的值）转换成字节序列的过程。这个字节序列可以被保存到文件、数据库中，或者通过网络传输。反之，从字节序列中恢复出 Java 对象的过程称为**反序列化**。对于 JavaBean 来说，实现序列化接口使得它的实例可以方便地在网络中传输或持久化存储（例如，将一个 JavaBean 对象存入 `HttpSession` 中，服务器可能会将其序列化到磁盘以实现钝化和活化）。

四、编程应用题

(以下是复习重点)

考试例题 1: 编写一个简单的 JavaBean, 通过 EL 的存取运算符访问其属性。

(注：原文档中的 JSP 代码使用了 `<jsp:getProperty>`，这与题目要求的"通过 EL 访问"不符。下面提供符合题目要求的正确代码。)

答案:

1. `userLogin.java` 文件

```java
package bean;

import java.io.Serializable;

// 建议实现 Serializable 接口
public class userLogin implements Serializable {
    private String userName;
```

```
    private String password;
    private String email;

    // 公共的无参构造函数
    public userLogin() {
        // 可以初始化默认值
        this.userName = "李平";
        this.password = "123";
        this.email = "liping@sohu.com";
    }

    // userName 的 getter 和 setter
    public String getUserName() {
        return userName;
    }

    public void setUserName(String userName) {
        this.userName = userName;
    }

    // password 的 getter 和 setter
    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    // email 的 getter 和 setter
    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }
}
```

2. `accessBean.jsp` 文件 (使用 EL 的正确版本)

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>

<%-- 使用 jsp:useBean 实例化或查找 JavaBean --%>
<jsp:useBean id="user" class="bean.userLogin" scope="session"/>

<html>
<head>
    <title>使用 EL 存取运算符访问 JavaBean 的属性</title>
</head>
<body>
    <h3>通过 EL 存取运算符访问 JavaBean 的属性</h3>

    <!-- 使用 EL 表达式访问属性 -->
    用 户 名: ${user.userName} <br>
```

```
    密     码: ${user.password} <br>
    电子邮箱: ${user.email} <br>

    <hr>

    <%-- 也可以使用 [] 运算符 --%>
    用 户 名 (使用[]): ${user["userName"]} <br>

</body>
</html>
```

运行结果 (图1 运行结果):

```
通过 EL 存取运算符访问 JavaBean 的属性
用户名: 李平
密码: 123
电子邮箱: liping@sohu.com
--------------------------------------------------------------------------------
--
用户名 (使用[]): 李平
```

考试例题 2: 编写一个简单的登录控制的 Servlet 程序,并在 web.xml 中配置该 Servlet。

答案:

1. `LoginServlet.java` 文件

```
package servlet;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class LoginServlet extends HttpServlet {

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        // 1. 设置请求编码，防止中文乱码
        req.setCharacterEncoding("UTF-8");

        // 2. 获取表单提交的用户名和密码
        String username = req.getParameter("username");
        String password = req.getParameter("password");

        // 3. 进行简单的业务逻辑判断
        if ("admin".equals(username) && "123456".equals(password)) {
            // 登录成功
            // 将用户信息存入 session
            req.getSession().setAttribute("loggedInUser", username);
            // 请求转发到成功页面
```

```
            req.getRequestDispatcher("/welcome.jsp").forward(req, resp);
        } else {
            // 登录失败
            // 在 request 域中设置错误消息
            req.setAttribute("errorMessage", "用户名或密码错误！");
            // 请求转发回登录页面
            req.getRequestDispatcher("/login.jsp").forward(req, resp);
        }
    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        // 通常登录操作使用 POST，GET 请求可以重定向或提示
        doPost(req, resp);
    }
}

```

2. `web.xml` 文件配置

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0">

    <!-- 1. 声明 Servlet -->
    <servlet>
        <!-- Servlet 的内部名称，可自定义 -->
        <servlet-name>LoginServlet</servlet-name>
        <!-- Servlet 类的完整路径 -->
        <servlet-class>servlet.LoginServlet</servlet-class>
    </servlet>

    <!-- 2. 映射 Servlet 的访问 URL -->
    <servlet-mapping>
        <!-- 必须与上面声明的 servlet-name 一致 -->
        <servlet-name>LoginServlet</servlet-name>
        <!-- 客户端浏览器访问的 URL 模式 -->
        <url-pattern>/login</url-pattern>
    </servlet-mapping>

    <!-- (可选) 配置欢迎页面 -->
    <welcome-file-list>
        <welcome-file>login.jsp</welcome-file>
    </welcome-file-list>

</web-app>
```

好的，我们继续完成编程应用题部分的复习重点。

---

3. 能够熟练编写 JSP 指令、动作和小脚本

这部分要求掌握 JSP 的基本语法，特别是如何使用 JSP 动作标签与 JavaBean 进行交互。

考试例题：创建一个用户注册页面，当用户提交信息后，使用 JSP 动作标签自动将表单数据填充到一个 JavaBean 中，并显示出来。

答案:

1. 创建一个 `User.java` JavaBean

```java
// File: src/bean/User.java
package bean;

import java.io.Serializable;

public class User implements Serializable {
    private String username;
    private String password;
    private int age;

    public User() {
        // 无参构造
    }

    // Getters and Setters
    public String getUsername() { return username; }
    public void setUsername(String username) { this.username = username; }

    public String getPassword() { return password; }
    public void setPassword(String password) { this.password = password; }

    public int getAge() { return age; }
    public void setAge(int age) { this.age = age; }
}
```

2. 创建注册表单页面 `register.jsp`

```jsp
<%-- File: webapp/register.jsp --%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>用户注册</title>
</head>
<body>
    <h2>新用户注册</h2>
    <form action="process.jsp" method="post">
        用户名: <input type="text" name="username"><br/>
        密　码: <input type="password" name="password"><br/>
        年　龄: <input type="text" name="age"><br/>
        <input type="submit" value="注册">
    </form>
</body>
</html>
```

3. 创建处理页面 `process.jsp`

```jsp
<%-- File: webapp/process.jsp --%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>

<%-- 1. 使用 jsp:useBean 实例化 User 对象 --%>
<jsp:useBean id="newUser" class="bean.User" scope="request"/>

<%--
   2. 使用 jsp:setProperty 自动填充 Bean 属性
   property="*" 表示将所有与 Bean 属性同名的请求参数自动设置进去
--%>
<jsp:setProperty name="newUser" property="*"/>

<html>
<head>
    <title>注册信息确认</title>
</head>
<body>
    <h2>注册成功！您的信息如下：</h2>

    <%-- 3. 使用 jsp:getProperty 或 EL 表达式显示 Bean 属性 --%>
    用 户 名: <jsp:getProperty name="newUser" property="username"/><br/>
    密　　码: (出于安全考虑，密码不应直接显示)<br/>
    年　　龄: ${newUser.age}<br/> <%-- 使用 EL 表达式更简洁 --%>

    <hr/>

    <%-- 使用小脚本 (Scriptlet) 演示 --%>
    <%
        // 小脚本是嵌入在 <% ... %> 中的 Java 代码
        // 虽然灵活，但破坏了 MVC 结构，现在已不推荐大量使用
        String message = "欢迎您, " + newUser.getUsername() + "!";
        out.println("<p>" + message + "</p>");
    %>
</body>
</html>
```

解析:

- `process.jsp` 页面完美展示了 JSP 动作标签的强大功能。
- `<jsp:useBean>` 首先在 `request` 作用域中创建了一个 `User` 类的实例，并命名为 `newUser`。
- `<jsp:setProperty name="newUser" property="*"/>` 是核心，它会自动匹配 `request` 中 `username`, `password`, `age` 等参数，并调用 `newUser` 对象的 `setUsername()`, `setPassword()`, `setAge()` 方法进行赋值，极大地简化了代码。
- 最后使用 `<jsp:getProperty>` 和 EL 表达式 `${newUser.age}` 来获取并显示 JavaBean 中的数据。
- 小脚本 `<% ... %>` 部分展示了如何在 JSP 中嵌入原生 Java 代码，但这通常被认为是过时的做法。

---

4 & 2. 会写代码使用传统方法进行数据库的连接与访问 (JDBC) 和 编写 DAO 类

这两个技能点紧密相关，通常结合在一起实现。DAO (Data Access Object) 模式用于将数据访问逻辑从业务逻辑中分离出来。

考试例题：编写一个 `UserDAO` 类，使用传统的 JDBC 方式，实现根据用户名查询用户信息的功能。

答案:

1. 准备一个数据库工具类 `DBUtil.java` (可选，但推荐)

```java
// File: src/util/DBUtil.java
package util;

import java.sql.*;

public class DBUtil {
    private static final String URL = "jdbc:mysql://localhost:3306/testdb?useSSL=false&serverTimezone=UTC";
    private static final String USER = "root";
    private static final String PASSWORD = "your_password"; // 替换为你的数据库密码

    static {
        try {
            // 1. 加载驱动
            Class.forName("com.mysql.cj.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }

    // 获取连接
    public static Connection getConnection() throws SQLException {
        // 2. 建立连接
        return DriverManager.getConnection(URL, USER, PASSWORD);
    }

    // 关闭资源
    public static void close(Connection conn, Statement stmt, ResultSet rs) {
        try {
            if (rs != null) rs.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        try {
            if (stmt != null) stmt.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        try {
            if (conn != null) conn.close();
        } catch (SQLException e) {
```

```
            e.printStackTrace();
        }
    }
}
```

2. 编写 `UserDAO.java`

```java
// File: src/dao/UserDAO.java
package dao;

import bean.User;
import util.DBUtil;
import java.sql.*;

public class UserDAO {

    public User findUserByUsername(String username) {
        Connection conn = null;
        PreparedStatement pstmt = null; // 使用 PreparedStatement 防止 SQL 注入
        ResultSet rs = null;
        User user = null;

        try {
            // 获取连接
            conn = DBUtil.getConnection();

            // 3. 创建语句对象
            String sql = "SELECT * FROM users WHERE username = ?";
            pstmt = conn.prepareStatement(sql);
            pstmt.setString(1, username);

            // 4. 执行 SQL
            rs = pstmt.executeQuery();

            // 5. 处理结果集
            if (rs.next()) {
                user = new User();
                user.setUsername(rs.getString("username"));
                user.setPassword(rs.getString("password"));
                user.setAge(rs.getInt("age"));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            // 6. 关闭资源
            DBUtil.close(conn, pstmt, rs);
        }
        return user;
    }
}
```

解析:

- 代码严格遵循了 JDBC 的六个步骤。

- 使用了 `PreparedStatement` 而不是 `Statement`，这是更安全、更高效的选择，可以有效防止 SQL 注入攻击。
- 将资源关闭逻辑封装在 `DBUtil` 类和 `finally` 块中，确保即使发生异常，数据库连接等资源也能被正确释放。

---

5. 会配置数据源，会写代码使用数据源获取数据库的连接

使用数据源和连接池是企业级应用中的标准做法，它比传统 JDBC 方法性能更高、管理更方便。

考试例题：将上面的 `DBUtil` 修改为使用 Tomcat 配置的数据源来获取连接。

答案:

1. 在 Tomcat 中配置数据源
在 Tomcat 的 `conf/context.xml` 文件中（或 `META-INF/context.xml`），添加如下 `Resource` 配置：

```
<Context>
    ...
    <Resource name="jdbc/myDataSource"
              auth="Container"
              type="javax.sql.DataSource"
              maxTotal="100"
              maxIdle="30"
              maxWaitMillis="10000"
              username="root"
              password="your_password"
              driverClassName="com.mysql.cj.jdbc.Driver"
              url="jdbc:mysql://localhost:3306/testdb?
useSSL=false&serverTimezone=UTC"/>
    ...
</Context>
```

2. 修改 `DBUtil.java` 以使用数据源

```
// File: src/util/DBUtil.java
package util;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.sql.DataSource;
import java.sql.*;

public class DBUtil {

    private static DataSource dataSource;

    // 使用静态代码块初始化数据源
    static {
        try {
            // JNDI (Java Naming and Directory Interface) 查找
            Context ctx = new InitialContext();
```

```
            // "java:comp/env/"是 J2EE 应用的标准前缀
            dataSource = (DataSource)
ctx.lookup("java:comp/env/jdbc/myDataSource");
        } catch (NamingException e) {
            throw new RuntimeException("DataSource configuration error!", e);
        }
    }

    // 从数据源获取连接
    public static Connection getConnection() throws SQLException {
        return dataSource.getConnection();
    }

    // 关闭资源方法保持不变...
    public static void close(Connection conn, Statement stmt, ResultSet rs)
{
        // ...
    }
}
```

解析:

- 现在 `getConnection()` 方法不再是每次都创建新的物理连接，而是从 Tomcat 维护的连接池中获取一个可用的连接，效率极高。
- 应用代码通过 JNDI 名称 (`jdbc/myDataSource`) 来查找数据源，实现了与具体数据库配置的解耦。如果数据库密码、地址等信息变更，只需修改 `context.xml`，无需改动和重新编译 Java 代码。

---

7 & 8. 会编写 Web 事件监听程序和 Web 过滤器

考试例题 1：编写一个监听器，在 Web 应用启动时打印一条欢迎信息。

```
// File: src/listener/AppContextListener.java
package listener;

import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;

public class AppContextListener implements ServletContextListener {

    @Override
    public void contextInitialized(ServletContextEvent sce) {
        // Web 应用启动时调用此方法
        System.out.println("========================================");
        System.out.println("My Web Application has started!");
        System.out.println("========================================");
    }

    @Override
    public void contextDestroyed(ServletContextEvent sce) {
        // Web 应用关闭时调用此方法
        System.out.println("========================================");
        System.out.println("My Web Application is shutting down.");
        System.out.println("========================================");
```

```
    }
}
```

在 `web.xml` 中注册监听器：

```
<web-app ...>
    ...
    <listener>
        <listener-class>listener.AppContextListener</listener-class>
    </listener>
    ...
</web-app>
```

考试例题 2：编写一个过滤器，解决全站 POST 请求的中文乱码问题。

```
// File: src/filter/CharacterEncodingFilter.java
package filter;

import javax.servlet.*;
import java.io.IOException;

public class CharacterEncodingFilter implements Filter {

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain)
            throws IOException, ServletException {
        // 拦截请求，设置请求的字符编码为 UTF-8
        request.setCharacterEncoding("UTF-8");

        // 将请求传递给过滤器链中的下一个组件（或其他过滤器或 Servlet）
        chain.doFilter(request, response);
    }

    // init 和 destroy 方法可以留空
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {}

    @Override
    public void destroy() {}
}
```

在 `web.xml` 中注册和映射过滤器：

```
<web-app ...>
    ...
    <filter>
        <filter-name>EncodingFilter</filter-name>
        <filter-class>filter.CharacterEncodingFilter</filter-class>
    </filter>

    <filter-mapping>
        <filter-name>EncodingFilter</filter-name>
        <!-- /* 表示拦截所有请求 -->
```

```
        <url-pattern>/*</url-pattern>
    </filter-mapping>
    ...
</web-app>
```

解析:

- **监听器** 是事件驱动的，用于在特定事件（如应用启动/关闭、session创建/销毁）发生时执行代码，非常适合进行初始化和资源清理工作。
- **过滤器** 则是基于请求拦截的，对每个符合条件的请求进行预处理和后处理，是实现认证、日志、编码转换等横切关注点的理想选择。