填空10\*1；

选择10\*1；ppt内容

程序分析（20=5+10+5）

程序填空（20=10+10=5\*2）

程序设计（40=

实体创建10

表单创建5

完整的servlet获取数据17（数据库操作JDBC和DataSource）

JSP，EL，响应输出8）

protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

// TODO Auto-generated method stub

String username = req.getParameter("username");

String password = req.getParameter("password");

resp.setContentType("text/html;charset=UTF-8");

PrintWriter out = resp.getWriter();

out.println("<!DOCTYPE html>");

out.println("<html><body>");

if ("admin".equals(username) && "admin".equals(password)) {

req.setAttribute("username", username);

RequestDispatcher rd = req.getRequestDispatcher("/welcome.jsp");

rd.forward(req, resp);(分写；2分）

} else {

RequestDispatcher rd = req.getRequestDispatcher("/login.html");

rd.forward(req, resp);

}

out.println("</body></html>");

}

第一章

HTTP（Hypertext Transfer Protocol）称为超文本传输协议，它是Web使用的协议。

是一个基于请求-响应（request-response）模式的协议。

Web服务器处理静态请求和动态请求

URL（Uniform Resource Locator）称为统一资源定位器,指Internet上位于某个主机上的资源。

URI（Uniform Resource Identifier）称为统一资源标识符

http默认端口号：80

Tomcat：8080

MySQL：3306

XML（eXtensible Markup Language）称为可扩展标记语言

XML具有如下的一些特点：

1. 可扩展性。
2. 更多的结构和语义。
3. 自描述性。
4. 数据与显示分离。（初始化参数）

Tomcat安装目录中包含一系列的子目录，这些子目录分别用于存放不同功能的文件。

（1）bin：用于存放Tomcat的可执行文件和脚本文件（扩展名为bat的文件）。

（2）conf：用于存放Tomcat的各种配置文件，如web.xml、server.xml。

（3）lib：用于存放Tomcat服务器和所有Web应用程序需要访问的JAR文件。

（4）logs：用于存放Tomcat的日志文件。

（5）temp：用于存放Tomcat运行时产生的临时文件。

（6）webapps：Web应用程序的主要发布目录，通常将要发布的应用程序放到这个目录下。

（7）work：Tomcat的工作目录，JSP编译生成的Servlet源文件和字节码文件放到这个目录下。

[http://localhost:8080或者http://127.0.0.1:8080](http://localhost:8080或者http:/127.0.0.1:8080)

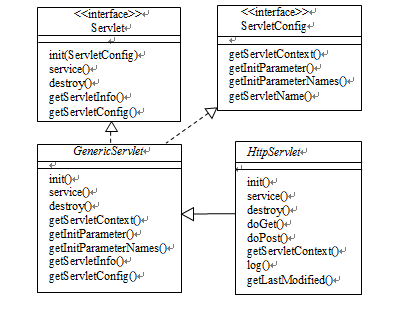
（localhost和127.0.0.1都表示本地计算机）访问Tomcat服务器。

第二章Servlet

Servlet是运行在Web服务器端的Java应用程序，它使用Java语言编写。

与Java程序的区别是，Servlet 对象主要封装了对HTTP请求的处理，并且它的运行需要Servlet容器的支持。Servlet容器指的就是Web服务器。

Servlet API：



1次

N次

1次

手机屏幕截图

描述已自动生成

|  |  |  |
| --- | --- | --- |
| 特征 | GET方法 | POST方法 |
| 资源类型 | 静态的或动态的 | 动态的 |
| 数据类型 | 文本 | 文本或二进制数据 |
| 数据量 | 一般不超过255个字符  (最多不能超过2KB。) | 没有限制 |
| 可见性 | 数据是URL的一部分，在浏览器的地址栏中对用户可见 | 数据不是URL的一部分而是作为请求的消息体发送，在浏览器的地址栏中对用户不可见  POST比GET请求方式更安全 |
| 数据缓存 | 数据可在浏览器的URL历史中缓存 | 数据不能在浏览器的URL历史中缓存 |

在客户端如果发生下面的事件，浏览器就向Web服务器发送一个HTTP请求。

1. 用户在浏览器的地址栏中输入URL并按回车键。
2. 用户点击了HTML页面中的超链接。
3. 用户在HTML页面中添写一个表单并提交。

前两种方法向Web服务器发送的都是 GET请求。如果使用HTML表单发送请求可以通过method属性指定使用GET请求或POST请求。’

有关属性存储的方法有4个，定义在HttpServletRequest接口中。格式如下。

public void setAttribute(String name,Object obj)

public Object getAttribute(String name) :要类型转换

public void removeAttribute(String name)

public Enumeration getAttributeNames()

处理中文乱码

方法一：将请求对象的字符编码和响应的内容类型都设置成UTF-8

request.setCharacterEncoding("UTF-8");

response.setContentType("text/html;charset=UTF-8");

方法二：客户端读取的中文使用String类的getBytes()方法

String username = request.getParameter("username");

username = new String(username.getBytes("ISO-8859-1"), "UTF-8");

输出流---PrintWriter

PrintWriter out = response.getWriter();

输出流---ServletOutputStream

ServletOutputStream sos= response.getOutputStream();

设置内容类型

response.setContentType("text/html;charset=UTF-8");

|  |  |  |
| --- | --- | --- |
| 状态码范围 | 含  义 | 示  例 |
| 100~199 | 表示信息 | 100表示服务器同意处理客户的请求 |
| 200~299 | 表示请求成功 | 200表示请求成功，204表示内容不存在 |
| 300~399 | 表示重定向 | 301表示页面移走了，304表示缓存的页面仍然有效 |
| 400~499 | 表示客户的错误 | 403表示禁止的页面，404表示页面没有找到 |
| 500~599 | 表示服务器的错误 | 500表示服务器内部错误，503表示以后再试 |

部署描述文件(.xml)

servlet-name：定义servlet名称；唯一；可通过ServletConfig.getServerName()获取

servlet-class：指定Servlet类的完整名称（带包名），容器使用类创建实例，可使用同一个 类创建多个实例。

init-param：定义向Servlet传递的参数。<param-name><param-value>

load-on-startup：指定是否在Web应用程序启动时加载该Servlet

映射：

< servlet-mapping >元素的作用：用在Servlet和URL样式之间定义一个映射。

它包含了两个子元素< servlet- name >和< url-pattern >

< servlet-name >元素给出的Servlet名字必须是在< servlet >元素中声明过的Servlet的名字

< url-pattern>元素指定对应于Servlet的URL路径，该路径是相对于Web应用程序上下文根 的路径。

扩展名匹配

以“\*”开头，后面接一个扩展名（如\*.do或者\*.pdf）

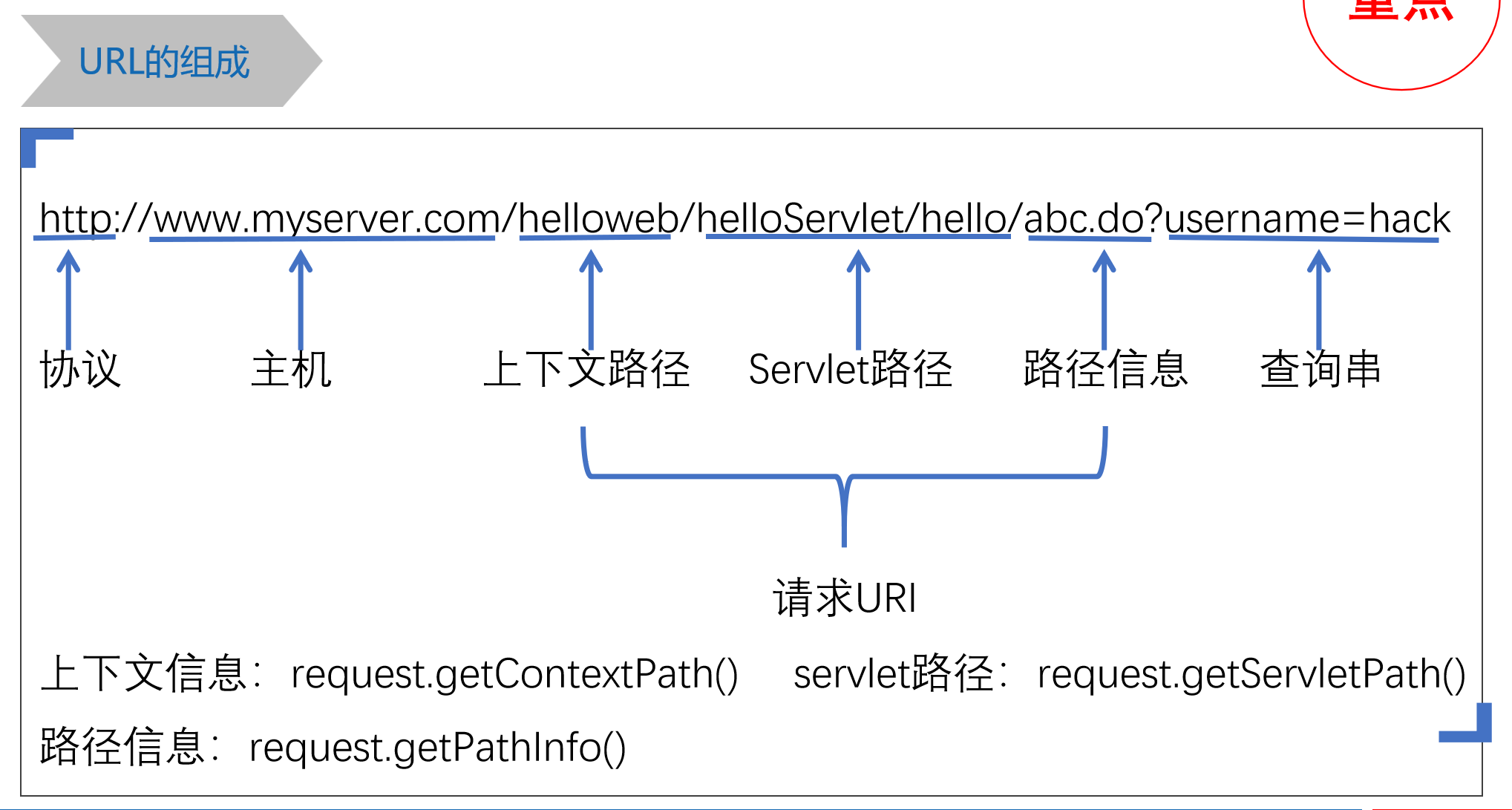
<servlet-mapping>

<servlet-name>PDFGenServlet</servlet-name>

<url-pattern>\*.pdf</url-pattern>

</servlet-mapping>

将把任何在servlet路径中以扩展名为pdf的请求都发送给PDFGenServlet



注解：

为HelloServlet添加的注解。

@WebServlet(name="helloServlet",urlPatterns={"/hello-servlet"})

@WebServlet注解包含多个属性，它们与web.xml中的对应元素等价。

name属性指定Servlet名称，等价于web.xml中的<servlet-name>元素。

urlPatterns属性指定一组URL映射模式，该元素等价于web.xml文件中的<url-pattern>元素

loadOnStartup属性指定该Servlet的加载顺序，等价于web.xml文件中的<load-on-startup>元素

initParams属性，它的类型是WebInitParam[]，指定Servlet的一组初始化参数，等价于<init-param>元素

@WebInitParam注解主要作用是为Servlet或Filter指定初始化参数，它等价于web.xml文件中<servlet>和<filter>元素的<init-param>子元素。

通常配合@WebServlet和@WebFilter使用。

name属性，指定初始化参数名，等价于<param-name>元素。

value属性，指定初始化参数值，等价于<param-value>元素。

ServletConfig

要得到ServletConfig接口对象有两种方法：

覆盖Servlet的init(ServletConfig config)，然后把容器创建的ServletConfig对象保存到一个成员变量中。

ServletConfig config = null;

public void init(ServletConfig config){ //重写init方法

super.init(config); // 调用超（父）类的init()

this.config = config;

}

另一种方法是在Servlet中直接使用getServletConfig()获得ServletConfig对象，如下所示。

ServletConfig config = getServletConfig();

检索Servlet上下文初始化参数的方法：

String getInitParameter(String name)

Enumeration getInitParameterNames()

获得Web应用程序的初始化参数：

ServletContext context = getServletContext();

String email = context.getInitParameter("adminEmail");

重点：ServletContext初始化参数 --->归属整个Web应用程序

Servlet初始化参数 ---> 归属于定义他们的Servlet

ServletContext对象是应用级对象（application级），而ServletRequest是request级对象。

1.如果使用绝对路径（"/index.jsp"），那他两毫无区别。

2.如果使用相对路径，则只有ServletRequest.getRequestDispatcher可用，因为request对象本身包含当前请求的path

第三章JSP

|  |  |  |
| --- | --- | --- |
| JSP页面元素 | 简要说明 | 标签语法 |
| 声明 | 声明变量与定义方法 | <%! Java 声明     %> |
| 小脚本 | 执行业务逻辑的Java代码 | <% Java 代码      %> |
| 表达式 | 用于在JSP页面输出表达式的值 | <%= 表达式       %> |
| 指令 | 指定转换时向容器发出的指令 | <%@  指令       %> |
| 动作（标签） | 向容器提供请求时的指令 | <jsp:标签名     /> |
| EL表达式 | JSP 2.0引进的表达式语言 | ${applicationScope.email} |
| 注释 | 用于文档注释 | <%-- 任何文本    --%> |

指令（在页面转换期间完成，是静态）

page指令通知容器关于JSP页面的总体特性。

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

include指令实现把另一个文件（HTML、JSP等）的内容包含到当前页面中。

<%@ include file=“copyright.html” %>

taglib指令用来指定在JSP页面中使用标准标签或者自定义标签的前缀和标签库的URI。

<%@ taglib prefix=”demo” uri=”/WEB-INF/mytaglib.tld” %>

（1）标签名、属性名以及属性值都是大小写敏感的。

（2）属性值必须使用一对单引号或者双引号括起来。

动作（actions）是页面发给容器的命令，它指示容器在页面执行期间完成某种任务。动作的一般语法为：

<prefix:actionName attribute-list />

在JSP页面中可以使用三种动作：JSP标准动作，标准标签库（JSTL）中的动作和用户自定义动作。

JSP标准include动作：<jsp:include page="copyright.jsp" />

常用的JSP动作

（1）jsp:include，在当前页面包含另一个页面的输出

（2）jsp:forward，将请求转发到指定的页面

（3）jsp:useBean，查找或者创建一个JavaBeans对象

（4）jsp:setProperty，设置JavaBeans对象的属性值

（5）jsp:getProperty，获取JavaBeans对象的属性值

（6）jsp:plugin，在JSP页面嵌入一个插件

JSP页面执行过程

客户首次访问页面时，Web容器执行该JSP页面要经过6个阶段。

1、转换阶段

2、编译阶段

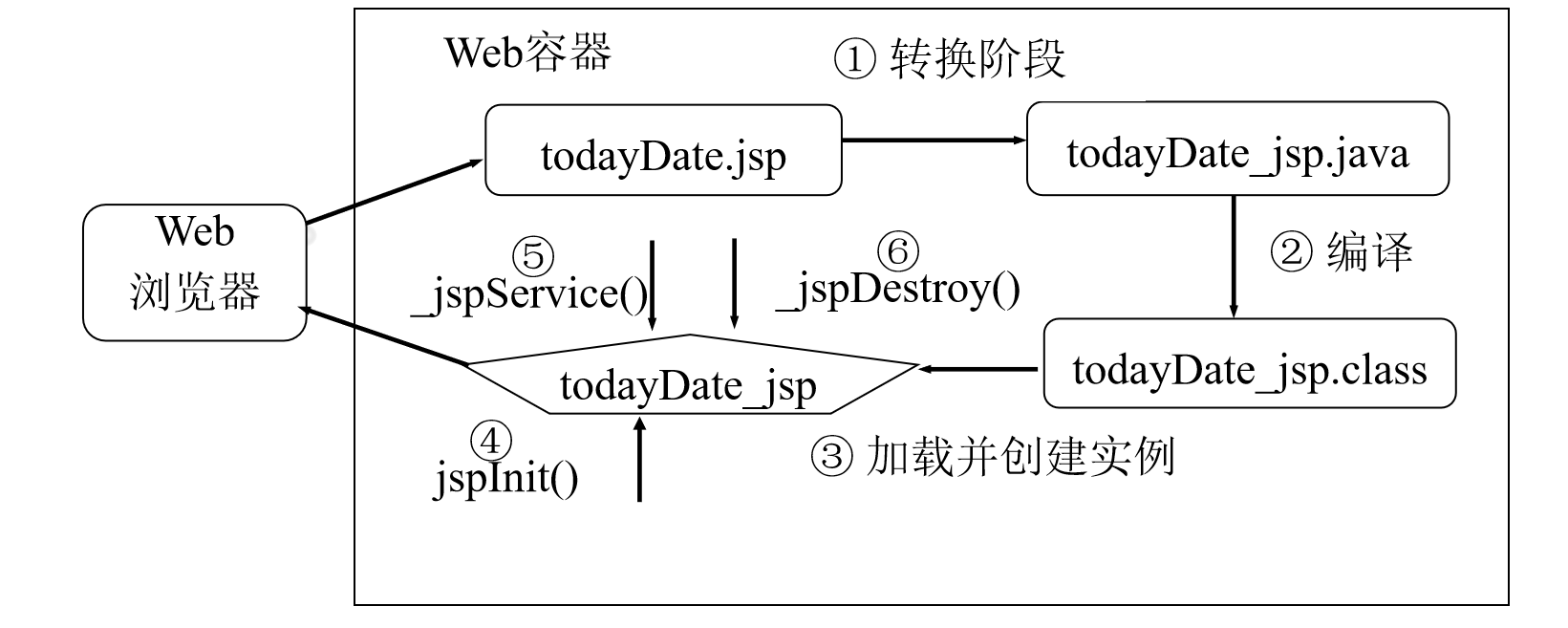
3、类的加载和实例化阶段

4、调用\_jspInit()

5、调用\_jspService()

6、调用\_jspDestroy()

前三个阶段将JSP页面转换成一个Servlet类并装载和创建该类实例，后三个阶段是初始化、提供服务和销毁阶段。



小脚本被转换成页面实现类的\_jspService()方法的一部分，因此小脚本中声明的变量成为该方法的局部变量，故它们出现的顺序很重要。

<% String s = s1 + s2; %>

<%! String s1 = "hello"; %>

<% String s2 = "world"; %>

<% out.print(s); %>

只输出”hello”

实例变量被自动初始化为默认值，而局部变量使用之前必须明确赋值。

<%! int a; %> // a是实例变量，默认值0（只初始化一次）

<% int b; %> // b是局部变量，使用前需赋值

a = <%= ++a %><br>

b = <%= ++b %><br> // 该行代码不能被编译

如果修改为<% int b = 5; %>，多次访问上面这个页面，a的值怎么变化？增加

b的值怎么变化？不变

JSP表达式并不总是输出到页面中，它们的也可以传递给JSP动作的属性。

<%! String pageURL = "copyright.jsp"; %>

<jsp:include page="<%= pageURL %>" />

向动作传递一个一个属性值使用的表达式称为请求时属性表达式

请求时属性表达式不能使用在指令的属性中，因为指令具有转换时的语义，容器仅在页面转换期间使用指令。

~~<%@ include file="<%= pageURL %>" %>~~

JSP隐含变量

session是javax.servlet.http.HttpSession类型的隐含变量，它在JSP页面中表示会话对象。

要使用会话对象，必须要求JSP页面参加HTTP会话，即要求将JSP页面的page指令的session属性值设置为true（默认值）。

<%@ page session = “true" %>（如果为false，则报错500）

<html><body>

会话ID = <%=session.getId()%>

</body></html>

config是javax.servlet.ServletConfig类型的隐含变量。

在讲DD部署文件为Servlet传递初始化参数，然后在Servlet找那个使用ServletConfig对象检索这些参数。

同理，JSP也可以做到。使用<jsp-file>元素使Servlet与JSP文件关联。

pageContext是javax.servlet.jsp.PageContext类型的隐含变量，它是一个页面上下文对象。

有下面三个作用：

1. 存储隐含对象的引用。
2. 在不同作用域内返回或设置属性的方便的方法。（3.7节详细说明）
3. 提供了forward()和include()。

例如，在Servlet中将请求转发到另一个资源，需要写下面两行代码。

RequestDispatcher rd = request.getRequestDispatcher("other.jsp");

rd.forward(request, response);

在JSP页面中，通过使用pageContext变量仅需一行就可以完成上述功能。

pageContext.forward("other.jsp");

page指令属性

使用page指令的errorPage属性将异常代理给另一个包含错误处理代码的JSP页面。使用isErrorPage属性指定当前页面是错误处理页面。

程序3.7 helloUser.jsp <%@ page errorPage="errorHandler.jsp/html" %>

程序3.8 errorHandler.jsp <%@ page isErrorPage="true" %>

静态包含和动态包含布局页面

静态包含是将另一个文件的内容包含到当前JSP页面中。用include指令完成，语法为：

<%@ include file="relativeURL" %>

file是include指令唯一的属性，它是指被包含的文件。文件路径以斜杠（/）开头，是相对于Web应用程序文档根目录的路径，路径不以斜杠开头，相对于当前JSP文件的路径。

从被包含页面中访问变量

由于被包含JSP页面的代码成为主页面代码的一部分，因此每个页面都可以访问另一个页面中定义的变量，包括共享所有的隐含变量。

静态包含的限制

（1）在转换阶段不进行任何处理，所以file属性值不能是请求时表达式。

<%!String URLPage="copyright.html"%>

<%@include file= "<%= URLPage %>" %>

（2）不能通过file属性值向被包含的页面传递任何参数，因为请求参数是请求的一个属性，它在转换阶段没有任何意义。

（3）被包含的页面可能不能单独编译。最好避免这种依赖性，使用隐含变量pageContext共享对象，通过使用pageContext对象的setAttribute方法。

动态包含是通过JSP标准动作<jsp:include>实现的。动态包含是在请求时将另一个页面的输出包含到主页面的输出中。该动作的格式如下：

<jsp:include page="relativeURL" flush="true | false" />

这里page属性是必须的，其值必须是相对URL，并指向任何静态或动态Web组件，包括JSP页面、Servlet等。

下面是include动作的使用：

<jsp:include page=“copyright.jsp" />

page属性的值可以是请求时表达式，例如：

<%! String pageURL = "copyright.jsp"; %>

<jsp:include page="<%= pageURL %>" />

作用域对象

|  |  |  |
| --- | --- | --- |
| 作用域名 | 对应的对象 | 存在性和可访问性 |
| 应用作用域 | application | 在整个Web应用程序有效 |
| 会话作用域 | session | 在一个用户会话范围内有效 |
| 请求作用域 | request | 在用户的请求和转发的请求内有效 |
| 页面作用域 | pageContext | 只在当前的页面（转换单元）内有效 |

应用作用域

在Servlet中可以使用ServletContext接口的setAttribute()方法将对象存储在应用作用域中。

String username = request.getParameter("username");

ServletContext context = getServletContext();

context.setAttribute("name", username);

在JSP页面中就可使用下面代码访问context中数据：

<%=application.getAttribute("name") %>

<jsp:useBean>动作示例：

<jsp:useBean id="customer" class="com.demo.Customer" />

等价代码：

Customer customer = (Customer)pageContext.getAttribute("customer");

if (customer == null){

customer = new Customer();

pageContext.setAttribute("customer", customer);

}

MVC：

模型2结构又称为MVC（Model-View-Controller）设计模式。在这种结构中，将Web组件分为模型（Model）、视图（View）和控制器（Controller），每种组件完成各自的任务。

该模型的最大优点是将业务逻辑和数据访问从表示层分离出来。

实现MVC模式的一般步骤

1. 定义JavaBeans存储数据

2. 编写Servlet处理请求

3. 数据结果存储在作用域中

4. 转发请求到JSP页面

5. 从JavaBeans中提取数据

错误处理

有多种错误处理方法：声明式错误处理和编程式错误处理。

使用page指令的errorPage属性指定一个错误处理页面，通过page指令的isErrorPage属性指定页面是错误处理页面。。

还可以在web.xml文件中为整个Web应用配置错误处理页面。使用这种方法还可以根据异常类型的不同或HTTP错误码的不同配置错误处理页面。

处理Servlet中产生的异常最简单的方法是将代码包含在try-catch块中，在异常发生时通过catch块将错误消息发送给浏览器。

try{

….

}catch(SQLException e){

response.sendError(

HttpServletResponse.SC\_INTERNAL\_SERVER\_ERROR,

"产生数据库连接错误，请联系管理员！"

);

}

HttpServletResponse接口定义了两个重载的sendError()方法，如下所示。

public void sendError (int sc)

public void sendError (int sc, String msg)

第四章Session

Web服务器跟踪客户的状态通常有4种方法：

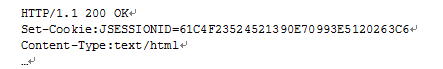
（1）使用Servlet API的Session机制；（最常用）

（2）使用持久的Cookie对象；

（3）使用URL重写机制；

（4）使用隐藏的表单域。

当服务器向客户端发送响应时，服务器将该会话ID与响应数据一起发送给客户，这是通过Set-Cookie响应头实现的，响应信息如：



这里的JSESSIONID的值即为会话ID，它是32位的十六进制数。

使用HttpSession对象：

通常需要三步：

1）创建或返回与客户请求关联的会话对象。

2）在会话对象中添加或删除“名/值”对属性。

3）如果需要可使会话失效。

创建或返回会话使用request对象的getSession()方法。

HttpSession getSession(boolean create)

HttpSession getSession()

超时：<session-timeout>元素中指定的以分钟为单位的超时期限。0或者小于0的值表示会话永不过期。如果没有通过上述方法设置会话的超时时限，默认情况下是30分钟。如果用户在指定期间内没有执行任何动作，服务器就认为用户处于不活动状态并使会话对象无效。

注销session：session.invalidate()

Cookie：  
对Cookie的管理需要使用javax.servlet.http.Cookie类，构造方法如下：

public Cookie(String name, String value)

Cookie类的常用方法如下：

public String getName()：返回Cookie名称，名称一旦创建不能改变

public String getValue()：返回Cookie的值

public void setValue(String newValue)：在Cookie创建后为它指定一个新值

public void setMaxAge(int expiry)：设置Cookie在浏览器中的最长存活时间，单位为秒。如果参数为负数，表示Cookie并不永久保存，如果是0表示删除该Cookie。

public int getMaxAge()：返回Cookie在浏览器上的最大存活时间

Cookie的管理包含两方面：将Cookie对象发送给客户端和从客户端读取Cookie

创建Cookie对象:

调用Cookie类的构造方法可以创建Cookie对象。下面语句创建了一个Cookie对象：

Cookie userCookie = new Cookie("username", "hacker");

设置Cookie的最大存活时间:  
在默认情况下发送到客户端的Cookie对象只是一个会话级别的Cookie，它存储在浏览器的内存中，用户关闭浏览器后Cookie对象将被删除。如果希望浏览器将Cookie对象存储到磁盘上，需要使用Cookie类的setMaxAge()设置Cookie的最大存活时间。 userCookie.setMaxAge(60\*60\*24\*7);

向客户发送Cookie对象:

要将Cookie对象发送到浏览端，需要调用响应对象的addCookie()将Cookie添加到Set-Cookie中。

response.addCookie(userCookie);

要从客户端读入Cookie，Servlet应该调用请求对象的getCookies()，该方法返回一个Cookie对象的数组。大多数情况下，只需要循环访问该数组的各个元素寻找指定名字的Cookie，然后对Cookie调用getValue()取得与指定名字关联的值。

调用请求对象的getCookies方法

Cookie[] cookies=request.getCookies();

对Cookie数组循环

for(int i = 0;i<cookies.length;i++){

Cookie cookie = cookies[i];

if(cookie.getName().equals(cookieName))

cookieValue = cookie.getValue();

}

文件上传：

上传文件表单的<form>标签中应该指定enctype属性，它的值应该为“multipart/form-data”，<form>标签的method属性应该指定为“post”，同时表单应该提供一个<input type="file">的输入域用于指定上传的文件。

第五章JDBC

JDBC是Java程序访问数据库的标准，它是由一组Java语言编写的类和接口组成，这些类和接口称为JDBC API。

JDBC的基本功能包括：

（1）建立与数据库的连接。

（2）发送SQL语句。

（3）处理数据库操作结果。

加载驱动程序最常用的方法是使用Class类的forName()静态方法

public static Class<?> forName(String className) throws ClassNotFoundException

eg：Class.forName("com.mysql.cj.jdbc.Driver");

参数className为字符串表示的完整的驱动程序类名。该方法返回一个Class类的对象。

驱动程序加载成功之后应使用DriverManager类的getConnection()建立数据库连接对象。

static Connection getConnection(String dburl,String user,String password)

数据库URL与一般的URL不同，它用来标识数据源。

String dburl =

"jdbc:mysql://127.0.0.1:3306/webstore?useSSL=true";

Connection conn = DriverManager.getConnection(dburl, "root", "123456");

通过Connection对象，可以创建语句（Statement）对象。对于不同的语句对象，可以使用Connection接口的不同方法创建。

Statement对象--->createStatement()

PreparedStatement对象--->preparedStatement()

Statement stmt = conn.createStatement(); 语句对象通过连接对象创建

创建一个预编译的PreparedStatement对象。

String sql="SELECT \* FROM products";

PreparedStatement pstmt = dbconn.prepareStatement(sql);

对于查询语句，调用executeQuery(String sql)返回ResultSet。

对于DDL语句如CREATE、ALTER、DROP和DML语句如INSERT、UPDATE、DELETE等须使用语句对象的executeUpdate(String sql)。该方法返回值为整数，用来指示被影响的行数。

String sql = "SELECT \* FROM products" ;

Statement stmt = conn.createStatement();

ResultSet rst = stmt.executeQuery(sql) ;

while(rst.next()){

out.print(rst.getString(1)+"\t") ;

}

PreparedStatement对象通常用来执行动态SQL语句，此时需要在SQL语句通过问号指定参数，每个问号为一个参数。

设置占位符

public void setString(int parameterIndex, String x)

parameterIndex为参数的序号，x为一个字符串。

例如：String sql = "INSERT INTO products VALUES(?, ?, ?, ?) ";

pstmt.setInt(1,105);

pstmt.setString(2,“手机”);

pstmt.setDouble(3,1490.0);

pstmt.setInt(4,5);

数据源:

在Tomcat中可以配置两种数据源：

局部数据源：只能被定义数据源的应用程序使用。

全局数据源：可被所有的应用程序使用。

局部数据源：

建立局部数据源非常简单，在Web应用程序的META-INF目录中建立一个context.xml文件

<?xml version="1.0" encoding="UTF-8"?>

<Context reloadable = "true">

<Resource

name="jdbc/webstoreDS" ---数据源名

type="javax.sql.DataSource" ---资源类型

maxTotal="4" ---数据源最大连接数

maxIdle="2" ---连接池中可空闲连接数

driverClassName="com.mysql.jdbc.Driver" ---使用驱动程序的完整类

url="jdbc:mysql://127.0.0.1:3306/webstore?useSSL=true" 数据库URL

username="root"

password="123456"

maxWaitMillis="5000"/> ---在没有可用连接等待时间

</Context>

全局数据源：

全局数据源可被所有应用程序使用，它是通过<tomcat-install>/conf/server.xml文件的<GlobalNamingResources>元素定义的，定义后就可以在任何的应用程序中使用。假设要配置一个铭文jdbc/webstoreDS的数据源。步骤如下：

（1）在server.xml文件中<GlobalNamingResources>元素内增加Resource相关代码

<Resource

name="jdbc/webstoreDS" ---数据源名

type="javax.sql.DataSource" ---资源类型

maxTotal="4" ---数据源最大连接数

maxIdle="2" ---连接池中可空闲连接数

driverClassName="com.mysql.jdbc.Driver" ---使用驱动程序的完整类

url="jdbc:mysql://127.0.0.1:3306/webstore?useSSL=true" 数据库URL

username="root"

password="123456"

maxWaitMillis="5000"/>

（2）在Web应用程序中建立一个META-INF目录，在其中建立一个context.xml文件。配置完毕后需重新启动Tomcat。

<Context reloadable="true">

<ResourceLink

global="jdbc/webstroeDS" ---全局资源名

name="jdbc/sampleDS" ---数据源明

type="javax.sql.DataSource"/> ---资源类型完整类名

<WatchedResource>WEB-INF/web.xml</WatchedResource>

</Context>

配置了数据源，使用javax.naming.Context接口的lookup()查找JNDI （Java Naming and Directory Interface，JNDI）数据源，下面代码可以获得jdbc/webstoreDS数据源的引用。

Context context = new InitialContext();

DataSource ds =(DataSource)context.lookup("java:comp/env/jdbc/webstoreDS");

得到了DataSource对象的引用后，就可以通过它的getConnection()获得数据库连接对象Connection。

dbconn = ds.getConnection();

这里的连接对象是通过连接池管理的。

DAO:

DAO（Data Access Object）称为数据访问对象。

DAO设计模式可以在使用数据库的应用程序中实现业务逻辑和数据访问逻辑分离，使应用的维护变得简单。

它将数据访问实现（通常使用JDBC技术）封装在DAO类中，提高应用程序的灵活性。

首先定义一个DAO接口，它负责建立数据库连接。

然后为每种实体的持久化操作定义一个接口，如ProductDao接口负责Product对象的持久化，CustomerDao接口负责Customer对象的持久化，最后定义这些接口的实现类。

第六章 表达式语言

可以使用JSP标准动作访问JavaBeans的属性：

<jsp:useBean id="employee" class="com.demo.Employee“ scope="session" />

<jsp:getProperty name="employee" property="empName" />

使用EL，通过点号表示法即可访问JavaBeans的属性：

${employee.empName}

使用表达式语言还允许访问嵌套属性。设Employee有一个address属性，它的类型为Address，它有city、street和zipCode属性。

<li>城市:${employee.address.city}

<li>街道:${employee.address.street}

<li>邮编:${employee.address.zipCode}

在EL中可以访问各种集合对象的元素，集合可以是数组、List对象或Map对象。

一般格式 ${attributeName[entryName]}

（1）如果attributeName对象是数组，则entryName为下标。上述表达式返回指定下标的元素值。

String[] fruits= {"apple","orange","banana"};

request.setAttributes("myFruits",fruits);

${myFruits[2]} 或者 ${myFruits[“2”]}

（2）如果attributeName对象是实现了List接口的对象，则entryName为索引。假设在Servlet中包含下列代码：

ArrayList<String> fruit = new ArrayList<>();

fruit.add("apple");

fruit.add("orange");

fruit.add("banana");

request.setAttribute("myFruit", fruit);

在JSP页面中就可以使用下面EL访问下标是2的列表元素。

我最喜欢的水果是:${myFruit[2]}

（3）如果attributeName对象是实现了Map接口的对象，则entryName为键值。假设在Servlet中包含下列代码：

Map<String,String> capital = new HashMap<String,String>();

capital.put("England","伦敦");

capital.put ("China","北京");

capital.put ("Russia","莫斯科");

request.setAttribute("capital", capital);

在JSP页面中使用下面EL访问指定键的值。

中国的首都是:${capital["China"]}<br> 程序6.8 CollectServlet.java

俄罗斯的首都是:${capital.Russia}



pageContext变量包含request、response、session、out和servletContext属性，使用pageContext变量可以访问这些属性的属性。

${pageContext.request.method}

${pageContext.request.requestURL}

EL是可以通过成员访问运算符访问对象的属性的，但不允许调用对象的方法。

下面的使用是错误的。

${pageContext.request.getMethod()}

然而，仍然可以使用下面的脚本表达式。

<%= request.getMethod() %>

第七章 JSTL

|  |  |  |
| --- | --- | --- |
| 库名称 | 使用的URI | 前缀 |
| **核心标签库** | **http://java.sun.com/jsp/jstl/core** | **c** |
| XML标签库 | http://java.sun.com/jsp/jstl/xml | x |
| 国际化和格式化库 | http://java.sun.com/jsp/jstl/fmt | fmt |
| SQL标签库 | http://java.sun.com/jsp/jstl/sql | sql |
| 函数库 | http://java.sun.com/jsp/jstl/functions | fn |

|  |  |
| --- | --- |
| JSTL标签类别 | JSTL标签 |
| 通用目的 | <c:out>、<c:set>、<c:remove>、<c:catch> |
| 条件控制 | <c:if>、<c:choose>、<c:when>、<c:otherwise> |
| 循环控制 | <c:forEach>、<c:forTokens> |
| URL处理 | <c:url>、<c:import>、<c:redirect>、<c:param> |

在JSP页面中使用JSTL，必须使用taglib指令来引用标签库，例如，要使用核心标签库，必须在JSP页面中使用下面的taglib指令。

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

c:out

【格式1】不带标签体的情况

<c:out value = "value" [escapeXml="{true|false}"] default = “默认值" />

【格式2】带标签体的情况

<c:out value = "value" [escapeXml="{true|false}"] default value />

标签需要一个value属性，它的值是向JSP页面中输出的值。default表示如果value的值为null或者不存在，就输出默认值。在格式2中的默认值在标签体中给出。

如果escapeXml为true（默认值），表示将value属性值中包含的<、>、’、“或者&等特殊符号转换为相应的实体引用（或字符编码）。比如小于号(<)将转换为&lt;，大于号(>)将转换为&gt;。如果escapeXml为false将不转换。

c:set

使用<c:set>标签的主要作用如下：

（1）定义一个字符串类型的作用域变量，并通过变量名引用它。

（2）通过变量名引用一个现有的作用域变量。

（3）设置作用域变量的属性值。

【格式1】不带标签体的情况

<c:set var = "varName" value= "value"

[scope = "{page| request| session| application}"] />

【格式2】带标签体的情况

<c:set var = "varName" [scope = "{page| request| session| application}"]

bodyContent

</c:set>

这里的var的属性值指定作用域变量的名称，value属性指定变量的值，scope指定变量的作用域，缺省为page作用域。这两种格式的区别是格式1使用value属性指定变量值，而格式2实在标签体中指定变量值。

c:remove

<c:remove>标签用来从作用域中删除变量，语法格式为：

<c:remove var="varName" [scope = "{page|request|session|application}"] />

var属性指定要删除的变量名，可选的scope属性指定作用域。如果没有指定的scope属性，容器将先在page作用域查找变量，然后是request，接下来是session，最后是application作用域，找到后将变量清除。

例如：<c:remove var="number" scope = "session" />

注意：<c:remove>标签不能用于删除JavaBeans或Map对象。

c:if

用来进行条件判断，它有下面两种语法格式。

【格式1】不带标签体的情况

<c:if test=“condition" var="varName"

[scope = "{page| request| session| application}"] />

【格式2】带标签体的情况

<c:if test=“condition" var="varName"

[scope = "{page| request| session| application}"] >

test为true执行标签体

</c:if>

c:choose

<c:choose>标签类似于Java语言的switch-case语句，它本身不带任何属性，包含多个<c:when>标签和一个<c:otherwise>标签，完成多分支结构。

<c:set var="color" value="white" scope="session" />

<c:choose>

<c:when test="${color == 'white'}">

白色!

</c:when>

<c:when test="${color == 'black'}">

黑色!

</c:when>

<c:otherwise>

其他颜色

</c:otherwise>

</c:choose>

第八章 监听器过滤器

监听器

处理ServletContextEvent事件

使用ServletContextListener接口

ServletContextListener接口定义的方法

public void contextInitialized (ServletContextEvent sce)：当ServletContext对象初始化调用。

public void contextDestroyed (ServletContextEvent sce)：当ServletContext对象销毁时调用。

处理ServletContextAttributeEvent事件

使用ServletContextAttributeListener接口

ServletContextAttributeListener接口定义的方法

void attributeAdded(ServletContextAttributeEvent sre)：当ServletContext对象中添加属性时调用该方法。

void attributeRemoved(ServletContextAttributeEvent sre)：当ServletContext对象中删除属性时调用该方法。

void attributeReplaced(ServletContextAttributeEvent sre)：当ServletContext对象中替换属性时调用该方法。

ServletRequestEvent事件是请求对象生命周期事件。

当一个请求对象初始化或者销毁时将发生该事件，处理该类事件需要使用ServletRequestListener接口。

public void requestInitialized(ServletRequestEvent sce)：当请求对象初始化时调用。

public void requestDestroyed(ServletRequestEvent sce)：当请求对象销毁时调用。

参数说明：ServletRequestEvent，该类定义了下面两个方法

public ServletContext getServletContext()：返回发生该事件的ServletContext对象

public ServletRequest getServletRequest()：返回发生该事件的ServletRequest对象

当在该对象上添加属性、删除属性或替换属性时会发生ServletRequestAttributeEvent事件，处理该类事件需要使用ServletRequestAttributeListener接口。

void attributeAdded(ServletRequestAttributeEvent sre)：当ServletRequest对象中添加属性时调用该方法。

void attributeRemoved(ServletRequestAttributeEvent sre)：当ServletRequest对象中删除属性时调用该方法。

void attributeReplaced(ServletRequestAttributeEvent sre)：当ServletRequest对象中替换属性时调用该方法。

HttpSessionEvent事件是会话对象生命周期事件。

当一个会话对象初始化或者销毁时将发生该事件，处理该类事件需要使用HttpSessionListener接口。

public void sessionCreated(HttpSessionEvent se)：当会话创建时调用。

public void sessionDestroyed(HttpSessionEvent se)：当会话销毁时调用。

参数说明：HttpSessionEvent，该类定义了下面方法

public HttpSession getSession()：返回状态发生改变的会话对象

当在该对象上添加属性、删除属性或替换属性时会发生HttpSessionBindingEvent事件，处理该类事件需要使用HttpSessionAttributeListener接口。

void attributeAdded(HttpSessionBindingEvent sre)：当在会话对象中添加属性时调用该方法。

void attributeRemoved(HttpSessionBindingEvent sre)：当从会话对象中删除属性时调用该方法。

void attributeReplaced(HttpSessionBindingEvent sre)：当会话对象中替换属性时调用该方法。

当一个对象绑定到会话对象或从会话对象中解除绑定时会发生HttpSessionBindingEvent事件，处理该类事件需要使用HttpSessionBindingListener接口。

void valueBound(HttpSessionBindingEvent event)：当对象绑定到一个会话时调用该方法。

void valueUnBound(HttpSessionBindingEvent event)：当对象从一个会话上解除绑定时调用该方法。

过滤器

Filter接口是过滤器API的核心，所有的过滤器都必须实现该接口。

public void init(FilterConfig filterConfig)

public void doFilter( ServletRequest request, ServletResponse response,

FilterChain chain)

public void destroy();

在web.xml中配置过滤器

<filter>

<filter-name>loginFilter</filter-name> 过滤器名称

<filter-class>com.ygj.control.loginFilter</filter-class> 过滤器类的包路径

<init—param> //可选

<param-name>参数名</param-name> 过滤器初始化参数

<param-value>参数值</param-value>

</init-pamm>

</filter>

<filter-mapping> 过滤器映射

<filter-name>loginFilter</filter-name>

<url-pattern>指定过滤器应用的对象</url-pattern>

</filter-mapping>

异步处理的操作步骤

（1）如果是使用注解方式配置的Servlet，那么设置@WebServlet注解的asyncSupport属性设置为true。如果是用xml配置方式配置的Servlet我们使用如下代码配置（<async-supported>true</async-supported>）：

<servlet>

<servlet-name>RequestInfoServlet</servlet-name>

<servlet-class>com.vgixt.helloworld.servlet.RequestInfoServlet</servlet-class>

<async-supported>true</async-supported>

</servlet>