考试大纲

JSP隐含对象 7个隐含对象

九大内置对象： applicatin、config 、exception、out、page、pageContext、request、response、session

- request:请求对象，　　类型：httpServletRequest
- response:响应对象　　类型：httpServletResponse
- session：表示一次会话，在服务器端记录用户状信息的技术
- application：标识web应用上下文，类型：ServletContext，详情就看Servlet中的ServletContext的使用
- exception 表示发生异常对象，类型 Throwable，在上面我们介绍page指令中的一个errorPage属性时就有说到他
- page：page对象代表当前JSP页面，是当前JSP编译后的Servlet类的对象。相当于this。
- config：标识Servlet配置，类型：ServletConfig，api跟Servlet中的ServletConfig对象是一样的，能获取该servlet的一些配置信息，能够获取ServletContext
- out：输出响应体 类型：JspWriter
- pageContext：表示 jsp页面上下文（jsp管理者） 类型：PageContext

注意：标记了颜色的对象就是JSP独有的，其他的都是Servlet中的老东西。

JSP原理(JSP->Java)

EL表达式语言

PageContext类

属性相关函数：getAtrribute() setAttribute()

获取隐含对象：`getOut()` `getResponse()` `getRequest()` `getSession()` `getServletContext()` `getServletConfig()`

异常：`getException()`

`getPage()` `getELcontext()`

Servlet接口，Servlet原理，web.xml，web.xml文件中 <context-param>

Servlet接口例子 包含三个：init doGet destroy

```
public class MyServlet extends HttpServlet {
```

```
    // 初始化方法，仅调用一次
    public void init(ServletConfig config) throws ServletException {
        super.init(config);
        System.out.println("Servlet is being initialized");
    }

    // 处理 GET 请求
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html><body>");
        out.println("<h2>Hello, Servlet!</h2>");
        out.println("</body></html>");
    }

    // 销毁方法，释放资源
    public void destroy() {
        System.out.println("Servlet is being destroyed");
    }
}
```

```
<context-param>
    <param-name>databaseURL</param-name>
    <param-value>jdbc:mysql://localhost:3306/mydb</param-value>
</context-param>
<context-param>
    <param-name>maxConnections</param-name>
    <param-value>10</param-value>
</context-param>
```

```
ServletContext context = getServletContext();//servlet继承了HttpServlet 而
getServletContext()是在HttpServlet中实现的
String dbURL = context.getInitParameter("databaseURL");  //注意是init
int maxConnections =
Integer.parseInt(context.getInitParameter("maxConnections"));
```

HttpServlet类

```
@WebServlet("/exampleServlet")
public class ExampleServlet extends HttpServlet {

    // 处理 GET 请求
```

```
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html><body>");
        out.println("<h1>Welcome to ExampleServlet - GET Request</h1>");
        out.println("</body></html>");
    }

    // 处理 POST 请求
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String name = request.getParameter("name");
        out.println("<html><body>");
        out.println("<h1>Welcome, " + name + " - POST Request</h1>");
        out.println("</body></html>");
    }
}
```

Filter接口 Filter原理

包含3部分 init doFilter destroy

可以在xml文件中定义

分为两部分 `<filter> </filter> <filter-mapping> </filter-mapping>`

前一部分定义过滤器的基本信息 后一部分定义路径和适用范围，两个部分中的filter-name必须一致

```
<filter>
    <filter-name>MyFilter</filter-name>
    <filter-class>com.example.MyFilter</filter-class>
</filter>

<filter-mapping>
    <filter-name>MyFilter</filter-name>
    <url-pattern>/myServlet</url-pattern>
</filter-mapping>
```

或者直接使用注解@WebFilter

```
import javax.servlet.*;
import javax.servlet.annotation.WebFilter;
import java.io.IOException;

@WebFilter("/*") // 应用于所有请求
public class EncodingFilter implements Filter {

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        // 初始化编码参数
    }

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

        // 放行请求
        chain.doFilter(request, response);
    }

    @Override
    public void destroy() {
        // 清理资源
    }
}
```

ServletConfig接口

简简单单 只有几个函数 Ps：getServletConfig( )是在HttpServlet中实现的

```
getInitParameter(string name)
getInitparameterNames() 返回Enumeration 所有初始变量的名称集合
还可以获取ServletContext      getServletContext()   getServletName()
```

```
package a;

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ServletConfigDemoServlet extends HttpServlet {
        protected void doGet(HttpServletRequest request, HttpServletResponse
```

```
response) throws ServletException, IOException {
                ServletConfig sc = getServletConfig();
                Enumeration<String> names = sc.getInitParameterNames();

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

                while(names.hasMoreElements()) {
                        String param_name = names.nextElement();
                        String param_value =
sc.getInitParameter(param_name);
                        initParameters.put(param_name, param_value);
                }

                response.setContentType("text/html;charset=utf-8");
                PrintWriter out = response.getWriter();
                out.println(initParameters);
        }
}
```

ServletRequestEvent类, HttpSessionEvent类

Listener组件

ServletRequest, ServletResponse,HttpServletRequest, HttpServletResponse

```
/**
 * Servlet implementation class RequestDemoServlet
 */
@WebServlet({ "/RequestDemoServlet", "/request" })
public class RequestDemoServlet extends HttpServlet {
        private static final long serialVersionUID = 1L;

        /**
         * @see HttpServlet#doGet(HttpServletRequest request,
HttpServletResponse response)
         */
        //ServletRequest 没有cookie
        protected void doGet(HttpServletRequest request, HttpServletResponse
```

```
response) throws ServletException, IOException {
                response.setContentType("text/html;charset=utf-8");
                PrintWriter out = response.getWriter();

                out.println("url: " + request.getRequestURL());//
http://localhost:8080/request/RequestDemoServlet 类型: string buffer
                out.println("<br>");
                out.println("uri: " + request.getRequestURI());//
/request/RequestDemoServlet string
                out.println("<br>");
                out.println("Context Path: " + request.getContextPath());//
/request
                out.println("<br>");
                out.println("Servlet Path: " + request.getServletPath());//
/request   如果只提到这个网址 有可能在根路径下所以返回“”也有可能返回“request”
                out.println("<br>");
                out.println("parameter a: " + request.getParameter("a"));//
                out.println("<br>");
                out.println("parameters b: " +
request.getParameterValues("b"));// 返回字符串数组 values
                out.println("<br>");
                out.println("QueryString: " + request.getQueryString());
                out.println("<br>");
        }

        /**
         * @see HttpServlet#doPost(HttpServletRequest request,
HttpServletResponse response)
         */
        protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
                // TODO Auto-generated method stub
                doGet(request, response);
        }

}
```

除此以外

request可以用于转发，response可以用于重定向

```
request.getRequestDispatcher(url地址/转发到资源的地址).forward(req, res);
```

```
//测试1: 从当前Servlet (day10/TestRedirect) 重定向到day10/index.jsp
// http://localhost/day10/TestRedirect
// http://localhost/day10/index.jsp
response.sendRedirect( "http://localhost/day10/index.jsp" );
response.sendRedirect( "/day10/index.jsp" );
response.sendRedirect( "/index.jsp" ); //错误路径
response.sendRedirect( "index.jsp" ); //正确路径
```

```
//测试2: 从当前Servlet重定向到day09/index.jsp
response.sendRedirect( "http://localhost/day09/index.jsp" );

//测试3: 从当前Servlet重定向到百度首页
response.sendRedirect( "http://www.baidu.com" );
```

ServletContext接口

Web Annotations， WebServlet, WebFilter, WebListener

路径匹配

Tag, SimpleTag, TagSupport, SimpleTagSupport

Tag原理

Spring控制器

Spring视图

Cookie相关函数，相关操作。

```
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;
import java.io.IOException;
```

```java
@WebServlet("/cookieExample")
public class CookieExampleServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        // 添加 Cookie
        Cookie cookie = new Cookie("user", "Alice");
        cookie.setMaxAge(24 * 60 * 60); // 1 day
        response.addCookie(cookie);

        // 读取 Cookie
        Cookie[] cookies = request.getCookies();
        String username = null;
        if (cookies != null) {
            for (Cookie c : cookies) {
                if ("user".equals(c.getName())) {
                    username = c.getValue();
                    break;
                }
            }
        }

        // 修改 Cookie
        Cookie updatedCookie = new Cookie("user", "Bob");
        updatedCookie.setMaxAge(24 * 60 * 60); // 1 day
        response.addCookie(updatedCookie);

        // 删除 Cookie
        Cookie deleteCookie = new Cookie("user", null);
        deleteCookie.setMaxAge(0); // 删除 Cookie
        response.addCookie(deleteCookie);

        // 响应输出
        response.setContentType("text/html;charset=UTF-8");
        response.getWriter().println("Cookie 操作完成！");
    }
}
```

获取客户端提交参数相关函数

tld文件结构

struts 1, ActionServlet, Action, ActionForm, DynaActionForm

猜数游戏 action.java

```java
public class GuessAction extends Action {

    @Override
    public ActionForward execute(ActionMapping mapping, ActionForm form,
                                 HttpServletRequest request,
HttpServletResponse response) {

        // 1. 将传入的ActionForm对象转换为DynaActionForm以便动态获取表单数据
        DynaActionForm guessForm = (DynaActionForm) form;

        // 2. 使用guessForm获取用户的输入值 "guess"，即用户的猜数
        String guess = (String) guessForm.get("guess");

        // 3. 获取当前会话session，用于存储和读取用户的猜数游戏状态
        HttpSession session = request.getSession();

        // 4. 获取或创建NumberGuessBean实例，用于存储用户的猜数、提示和状态
        NumberGuessBean numguess = null;
        if (session.getAttribute("numguess") != null) {
            // 如果用户已有存储在session中的NumberGuessBean实例，则获取它
            numguess = (NumberGuessBean) session.getAttribute("numguess");
        } else {
            // 如果没有，则创建一个新的实例并存储到session中
            numguess = new NumberGuessBean();
            session.setAttribute("numguess", numguess);
        }

        // 5. 调用NumberGuessBean的setGuess方法来处理用户的猜测，并更新状态
        numguess.setGuess(guess);

        // 6. 将更新后的numguess对象重新设置到session中，保存最新状态
        session.setAttribute("numguess", numguess);

        // 输出当前用户的猜测和相应提示，方便调试
        System.out.println("User guess: " + guess + ", Hint: " +
numguess.getHint());

        // 7. 判断用户是否猜中，如果成功，则转向"success"页面
        if (numguess.getSuccess()) {
            // 猜中后重置NumberGuessBean，开始新的猜数游戏
            numguess.reset();
            session.setAttribute("numguess", numguess); // 更新重置后的状态
            return mapping.findForward("success"); // 转向配置的success页面
        } else {
            // 如果未猜中，更新表单提示信息并返回继续页面
            guessForm.set("hint", numguess.getHint()); // 设置提示信息
            guessForm.set("numGuesses",
String.valueOf(numguess.getNumGuesses())); // 设置猜测次数

            return mapping.findForward("continue"); // 转向配置的continue页面
        }
    }
}
```

DispatchAction.java,MappingDispatchAction.java,

LookupDispatchAction.java, ActionForward, ActionMapping

Action.java

```
public class LoginAction extends Action {
    @Override
    public ActionForward execute(ActionMapping mapping, ActionForm form,
                                 HttpServletRequest request,
HttpServletResponse response) throws Exception {
        // 从请求中获取用户名和密码
        String username = request.getParameter("username");
        String password = request.getParameter("password");

        if ("admin".equals(username) && "password".equals(password)) {
            // 登录成功，返回 "success" 的 ActionForward
            return mapping.findForward("success");
        } else {
            // 登录失败，返回 "failure" 的 ActionForward
            request.setAttribute("error", "Invalid username or password.");
            return mapping.findForward("failure");
        }
    }
}
```

struts-config.xml

```
<action path="/login"
        type="com.example.LoginAction">
    <!-- 成功的转向路径 -->
    <forward name="success" path="/welcome.jsp"/>
    <!-- 失败的转向路径 -->
    <forward name="failure" path="/login.jsp"/>
</action>
```

struts1 配置文件 Web应用文件结构 HttpSession JspWriter

猜数游戏程序（Struts1, Spring)

其中猜数游戏的两个实现代码，还有DispatchAction.java, MappingDispatchAction.java,LookupDispatchAction.java, 是考核重点。

servletcontext和servletconfig都有getinitparam

不属于 Action 接口中定义的字符串常量的是(B)

A.INPUT B.FAILURE C.SUCCESS D.ERROR

相对路径和绝对路径

```
从当前Servlet转发到index.jsp(http://localhost/day10/index.jsp)
request.getRequestDispatcher("/index.jsp").forward(request, response);
request.getRequestDispatcher("index.jsp").forward(request, response);
```

相对路径：*不以* `/` 开头。相对路径是相对于当前 Servlet 的路径。

- `request.getRequestDispatcher("index.jsp")` 表示相对于当前 Servlet 路径的 `index.jsp` 文件。
- 例如，假设当前 Servlet 路径是**`/day10/someServlet`，这个相对路径会指向 **`/day10/index.jsp`，如果这个路径下没有 **`index.jsp` 文件，页面将无法找到资源。
- **绝对路径**： 以 `/` 开头。绝对路径是相对于应用根目录的路径。
- `request.getRequestDispatcher("/index.jsp")` 表示从应用的根目录开始寻找 `index.jsp` 文件。
- 如果**`index.jsp` 在** **`/day10/`** **目录下，正确的写法应为** `request.getRequestDispatcher("/day10/index.jsp")`。