- Servlet
  - **HttpServlet 考试重点**
- **ServletConfig**
  - getInitParameter()
- **ServletContext**
- ServletRequest
  - **HttpServletRequest（强制转换得到）** 考试重点
    - Cookie 与 Session 相关
      - getCookies() 得到一个 *Cookie 对象数组*
      - getSession() 得到与请求相关的 Session。*如果 Session 不存在就创建一个*
  - 多组件共享数据
    - setAttribute()
    - getAttribute()
  - 获取 Get 或 Post 请求的数据
    - getParameter(String name) 得到 name 对应的值
    - getParameters(String name) 得到 name 对应的值的数组
    - getParameterMap() 返回一个 Map，里面是请求参数的键值对
    - getParameterNames() 返回一个枚举对象
    - getParameterValues() 返回一个 String 数组
  - setCharacterEncoding()
- ServletResponse
  - **HttpServletResponse（强制转换得到）** 考试重点
    - 不要关闭流，谁打开的谁关闭
    - setContentLength 只接受 int 类型
    - **sendRedirect()** 考试重点
- **RequestDispatcher** 考试重点
  - 派发请求到其他 Servlet
  - forward
  - include
  - 不能两个组件各自处理一部分内容
- **HttpSession（重点查看所含的方法）**
  - invalidate()
- URLEncoder
- URLDecoder

Web 应用程序

文件夹结构：

```
/first/                     // 名称，随便起
    WEB-INF/
        classes/            // class 文件
        lib/                // jar 文件
        tags/               // 自定义标记
        web.xml             // 配置信息与描述性信息（重点）,受控 XML 文件
    META-INF/
        context.xml         // 声明上下文地址
```

xml

```
<?xml version="" encoding="" ?>
    <root color="red">
        <childNode></childNode>
        <childNode></childNode>
        <childNode></childNode>
    </root>
```

属性可以用 XML 属性和子节点表示，官方推荐使用子节点

Well-format

- 物理格式良好：遵守 XML 语法规则
- 逻辑格式良好：符合应用程序定义的格式
- DTD(Document Type Definition)

```
<!--一个 Students 中至少有一个 Student-->
<!ELEMENT students(student+)>
<!--一个 Student 中至少有 id, name, gender, age 属性-->
<!ELEMENT student(id, name, gender, age)>
```

- XML Schema

命名空间

先声明再使用

```
<student
    xmlns="https://www.google.com/"
    xmlns:p="https://soulike.tech/">

    <p:childNode p:attr="value"></p:childNode>
    <!--以上等价于-->
    <https://soulike.tech:childNode
        https://soulike.tech/:attr="value">

    </https://soulike.tech:childNode>

    <!--默认命名空间-->
    <childNode></childNode>
    <!--以上等价于-->
    <https://www.google.com/:childNode></https://www.google.com/:childNode>

</student>
```

受控 XML 文件

```
<?xml version="" encoding="" ?>
    <root
        schemaLocation="[namespace] [xsd file URL]"
        xmlns:x="namespacex"
        xmlns:y="namespacey">
        <x:a></x:a>
        <y:a></y:a>
    </root>
```

会话

实现方式

- Cookies
- Url rewriting
  - SessionID 附加在 URL 中
- Hidden field
  - 在表单中隐藏一个域包含 Session 信息

Session 过期时间设置

通过在 web.xml 中设置。单位为**分钟**

```
<session-config>
    <session-timeout>30</session-timeout>
</session-config>
```

设置 30 分钟后 Session 过期

Filter

预处理请求，后处理响应

用户 => Filter1 => ... => FilterN => Servlet => Filter1 => ... => FilterN => 用户

也可以就地处理请求（拦截）

- `destroy()`
- `doFilter(ServletRequest request, ServletResponse response, FilterChain chain)`
- `init(FilterConfig filterConfig)`

一般用 HttpFilter

FilterChain

- `doFilter(ServletRequest request, ServletResponse response)` 下一个过滤器的 doFilter() 方法

Filter 在 web.xml

```
<filter>...</filter>
<filter-mapping>
    <filter-name>...</filter-name>
    <url-pattern>...</url-pattern>
    <dispatcher>...</dispather>
</filter-mapping>
```

需要注意的

- `<dispather>` 的作用：设定什么时候调用这个 Filter，值有 REQUEST, INCLUDE, FORWARD, ERROR 四种

Listener

各种 Event（考试重点）

- `ServletContextEvent` 与 `ServletContextListener`
  - `contextInitialized(ServletContextEvent sce)` 应用程序启动后*第一个调用*的函数，可以用来做各种初始化
- `HttpSessionEvent`
- `HttpSessionBindingEvent`
  - 调用 `HttpSession.setAttribute()` 时触发

- `HttpSessionAttributeListener`
  - `attributeReplaced(HttpSessionBindingEvent event)`
    - **`event.getValue()` 返回值为修改前的值**

Web 应用程序部署方法（考）

1. 复制文件夹到 tomcat 安装目录下的 webapps 文件夹
2. 在 server.xml 的 `<Host>` 标签下添加 `<Context path="" docBase="" />`
   - path 是 URL 路径，如果是根可留空
   - docBase 是 Servlet 文件夹路径

```
<Host name="localhost" appbase="webapps" ……>
    <Context path="/hello" docBase="/home/soulike/hello" />
</Host>
```

3. 在 `META-INF` 文件夹下的 context.xml 中添加 `<Context path="/abc" />`

Web 应用程序打包方法

```
jar cvf hello.war -C /home/soulike/hello .
```

JSP (Java Server Page)

- `<%@ %>` 内嵌 JSP 指令
  - page 指令：指定页面属性
    - import，注意不同导入项用逗号分割
    - contentType: 废话，一般不写
    - pageEncoding: 指定页面编码
    - session: 默认为 true，可不写
    - isErrorPage: 默认为 false，可不写
    - errorPage
    - buffer：缓冲区大小，默认为 8KB。位于输出流之前的缓冲区
    - extends：指定页面的父类
  - include 指令：宏替换，**在 JSP 转换为 Java 文件过程中发生**，相当于把指定页面代码直接放在当前文件中
  - taglib 指令：引入自定义标签
    - prefix：标记前缀
    - uri：命名空间

```
<%@ page import="java.util.*, java.io.*"
         contentType="text/html;charset=utf-8"
         pageEncoding="utf-8"
         session="true"
         isErrorPage="false"
         errorPage="/error.jsp"
         buffer="none"
%>
```

```
<%@ include file="copyright.jsp" %>
```

```
<%@ taglib prefix="soulike" uri="http://mycompany.com" %>
```

- 标签
  - `<%= %>` 任何合法 Java 表达式
    - 避免声明变量，因为变量会变为成员变量，而 Servlet 是单实例的，**线程不安全**
  - `<%! %>` 声明标签，可声明变量、函数、类
    - 声明的变量是类的成员变量

    - `<% %>` 任何合法的 Java 代码
        - 声明的变量是函数的局部变量
        - 转换时会出现在 `service()` 函数中
    - `<%-- --%>` 注释，转换后被去除
        - 但是 HTML 与 Java 代码中的注释会保留

- **JSP 隐含对象（implicit object）（考）**

    - `HttpSevletRequest request`
    - `HttpServletResponse response`
    - `HttpSession session`
    - `ServletConfig config`
    - `ServletContext application`
    - `JspWriter out`：利用 `Buffer` 的输出，相比 `PrintWriter` 多声明抛出 `IOException`，但是并不需要就地处理
    - `PageContext pageContext`：各种相关的对象
    - `page`：未知类型，等价于 `this`
    - `Exception exception`：只能在 ErrorPage 中使用，在一般页面中使用为错误

- JSP Action

```
<jsp:forward page="/a.jsp" />
<jsp:include page="/b.jsp" />
```

- 对应 `RequestDispatcher` 的 `forward` 与 `include` 方法。

```
<jsp:useBean id="a" class="A" scope="page|request|session|application" />
```

- useBean 加入一个类
    - id 对象的名，相当于在指定作用域里面新建了一个叫指定名字的对象
    - class 类名，在对象不存在时用于以 id 为名新建对象
    - scope 作用域

```
<jsp:getProperty name="a" property="name" />
```

- getProperty
    - name 指定对象
    - property 指定要输出对象的属性
    - 以上代码相当于以下两行之一

```
<% out.print(a.getName()); %>
${a.name}
```

- setProperty
    - value 与 param
    - `*` 号，把请求里面的属性对应放入 `a` 对象中

```
<jsp:setProperty name="a" property="name" value="张三" />
<jsp:setProperty name="a" property="name" param="name" />
<jsp:setProperty name="a" property="*" />
```

- 前两行分别相当于

```
<%
    a.setName("张三");
    a.setName(request.getParameter("name"));
%>
```

- 第三行把用户提交属性对应赋值给 a 对象。如果名字不对应就不赋值
- 第二行的前提是，`getParameter` 的返回值不是 `null`。如果是 `null` 不执行

Java Bean 命名规范

Java Bean 中的类数据成员应当首字母小写。例如对于 `a.getNumberGuess()` 方法，一定要有一个数据成员名为 `numberGuess` 。即便是 `NumberGuess` 也会报错。

表达式语言

- ${expression}
  - 基本算术运算
  - 比较运算
  - 隐含对象
    - `requestScope` （重点）
    - `param` （重点）
    - `paramValues` （重点）
    - 其他 10 个……
    - **除了 `PageContext` 之外，都是 `Map` 类型**
  - 变量查找作用域顺序
    a. Page 作用域 `pageContext.setAttribute()`
    b. Request 作用域 `request.setAttribute()`
    c. Session 作用域 `session.setAttribute()`
    d. Application 作用域 `application.setAttribute()`
    - 当然，可以利用 `PageContext.**_SCOPE` 常量修改调用 `setAttribute()` 方法定义属性的作用域
      - `pageContext.setAttribute('attr', obj, PageContext.APPLICATION_SCOPE)` 相当于调用 `application.setAttribute('attr', obj)`

自定义标记

- 用标记来替代 JSP 中的 Java 代码，将 Java 代码放到 Java 源文件中
- 组成部分
  - Java 程序：业务逻辑
    - JSP 中的对象由 `pageContext` 对象传入
  - 映射文件：把标记映射到 Java 程序中的某个方法
- **javax.servlet.jsp.tagext**
  - 接口 `Tag`
  - 适配器 `TagSupport` ，含有 `PageContext`
  - 适配器 `SimpleTagSupport`

```
public class Tag extends TagSupport
{
    void doTag() throws JspException, IOException
    {
        // ...codes here...
    }

    void release()
    {
        // 因性能考虑，Tag 对象在使用完成后不会销毁，留待下次使用。因此需要在每次释放对象后重新设置对象的值
        // ...initializing codes here...
    }

    // 标准库设计失误，少一个 IOException
    // 返回值为枚举值，内容见下文
    int doStartTag() throws JspExcetion     // 起始标记
    {

    }

    int doEndTag() throws JspException      // 结束标记
    {

    }
}
```

例如猜数游戏中的 `reset()` 调用

```
public class ResetTag extends TagSupport
{
    void doTag() throws JspException, IOException
    {
        NumberGuessBean numguess;
        //numberguess = pageContext.getAttribute("numberguess", PageContext.PAGE_SCOPE);
        // 上面这句等价于下面，下面的可能更清晰一点
        HttpSession session = pageContext.getSession();
        numberguess = (NumberGuessBean) session.getAttribute("numberGuess");
        numberguess.reset();
    }
}
```

- 枚举返回值（见文档 `Interface Tag` 接口中常量）
    - `EVAL_BODY_INCLUDE`
    - `EVAL_PAGE`
    - `SKIP_BODY`
    - `SKIP_PAGE`
- 调用自定义标记
    - 需要添加 WEB-INF/xxx.tld
    - 标记属性
        - 如果有自定义属性，需要在标记处理器类文件中加入对应的 set 函数
        - `rtexprvalue` ： runtime expression value
            - 属性值是否可以在运行时决定（动态生成的）
    - 注意 `<tei-class></tei-class>`

```
<tag>
        <name>nowTime</name>
        <tag-class>TimeTag</tag-class>
        <tei-class>MyTagExtraInfo</tei-class>
        <body-content>JSP</body-content>

        <attribute>
            <name>color</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
            <type>java.lang.String</type>
        </attribute>
    </tag>
```

- Tag 向 JSP 对象传输数据
    - 通过 `PageContext`
        - setAttribute()
        - getAttribute()
- `Class TagExtraInfo`
    - 在自定义标记处理器中向 JSP 传递变量
    - 使用时覆盖 `getVariableInfo(TagData td)` ，注意返回一个数组
        - `VariableInfo`
        - 在编译的时候调用
        - 在自定义标记处理器中需要使用 `Beans.instantiate()` 来实例化赋值

```
public class MyTagExtraInfo extends TagExtraInfo
{
    public VariableInfo[] getVariableInfo(TagData td)
    {
        return new VariableInfo[] {
            new VariableInfo(
                td.getAttributeString("var"),
                td.getAttributeString("type"),
                true,
                VariableInfo.NESTED
            )
```

```
        }
    }
}
```

```
<t:time var="d" type="java.util.Date">
```

- **新接口： `SimpleTag` （看文档）**
    - `JspFragment`
        - `invoke(writer out)`
            - `out` 可以是 `null`，如果是 `null` 则向 `jsp` 默认 `out` 输出
    - 获取 `PageContext` 利用 `getJspContext()` 来进行强制转换
    - `IterationTag`
    - `BodyTag`
        - `EVAL_BODY_BUFFERED`： Body 解析结果将不会输出到默认 out 对象，而是输出到一个新建的 out 中
            - 也就是说，自定义标记里面的内容不一定输出到页面里面
            - `getPageContext().getOut()` 获取的也是新建的 out 对象
        - `bodyContent` 获取另外 out 对象输出过的内容，可以进行进一步处理
            - `getString()` 获取缓冲区内容
            - `getEnclosingWriter()` 获取原始 out 对象
- Tag 文件
    - 不需要写 tld 文件

```
<%@ taglib tagdir="/WEB-INF/tags" prefix="a">
```

邮件编程

- javax.mail
    - javax.mail.internet
        - Message
            - MimeMessage
        - Address
        - MultiPart

MVC 设计模式

- MVC
- MVC2
    - struts

Struts 1.3.10

- 拆分控制器
    - 前端控制器
    - 后端控制器
- 在 web.xml 里加载 Struts

```
<servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    <init-param>
        <param-name>config</param-name>
        <param-value>WEB-INF/struts-config.xml</param-value>
</servlet>
```

Struts 配置文件名可以随意修改。

- struts-config.xml

不是必需的。

对于以下代码

- 遇到 `calc.do` 就会分配到 `CalcAction` 类进行处理
- `name` 指定 `formBeans`，封装输入输出数据
- `formBean` 并不是必须的

```
<struts-config>
    <form-beans>
        <form-bean …… />
    </form-beans>
    <action-mappings>
        <action path="/calc" type="com.abc.action.CalcAction" name="calcForm"       input="input.jsp">
            <forward name="result" path="/result.jsp" redirect="false"></forward>
        </action>
    </action-mappings>
<struts-config>
```

- `Class ActionServlet` （前端控制器）
- **`Class Action` （后端控制器共有父类，看 API 文档）**
  - `execute()`，返回值 `ActionForward`
    - `ActionForm`
- `Class LookupDispatchAction`
- `Class MappingDispatchAction`
- `Class ActionForm`
  - `reset()`
  - `validate()`
- `Class DynaActionForm`
- `Class ActionForward`
- `Class ActionMapping`
  - `findForward ()`
- `Class ActionErrors`

注解 javax.servlet.annotation

- 注解，可以自动生成配置信息 **（替代 web.xml）**
- 所有的 annotation 都有默认属性 value

WebServlet

- value
- urlPatterns
- initParameters

```
@WebServlet(urlPatterns = {"/a", "/b"}, initParameters = {
    @WebInitParam(name1 = value1),
    @WebInitParam(name2 = value2)
})
public class A extends HttpServlet
{

}
```

其他看 API 文档

Spring

JDBC

```
import java.sql.*;
import javax.sql.*;     // 填空题，可以引入 DataSource 接口

public class TestJdbc
{
```

```
    public void test()
    {
        Class.forName("com.mysql.jdbc.Driver");
        Connection con = DriverManager.getConnection("jdbc:mysql……");
        // Statement
        // PreparedStatement

        // 以下示例查询 J2EE 不及格的男生
        PreparedStatement pstmt = con.prepareStatement("SELECT * FROM student WHERE j2ee<? AND gender=?");

        // 占位符 ? 从 1 开始计算
        pstmt.setInt(1, 60);
        pstmt.setInt(2, 1);

        ResultSet rs = pstmt.executeQuery();

        while(rs.next())
        {
            System.out.println(rs.getString("name"));
        }

        rs.close();
        pstmt.close();
        con.close();
    }
}
```

DataSource

- 一般放在 `ServletContextListener` 中

```
public void contextInitialized(ServletContextEvent sce)
{
    DataSource ds = new BasicDataSource();

    // 传给 Sevlet 的方法
    sce.getSevletContext().setAttribute("ds", ds);

    // 获取连接方法
    Connection con = ds.getConnection();
    con.close();
}
```

Spring: JdbcTemplate

JSTL(Java 通用标签) （考）

- 流程控制标签
  - `<c:if>`
  - `<c:when>`