# J2EE笔记

@author：ZZMine

## 介绍

j2ee是java se 的扩展，包名均以javax开头。java SE（桌面应用），java EE（企业应用），java ME （嵌入式应用）

javaEE平台很多都是接口，是一个抽象的平台称为“定义平台”。

**基于组件的开发**，组件（如servlet）在容器中被调用。应用服务器软件：开源免费: Apache-tomcat, 收费：IBM-WebSphere Application Server （WAS ）； Bea WebLogic

没有main函数，需要将项目部署(deploy)到Web服务器。打包成.war文件。

*URL的格式（三个部分）：**协议+主机(IP+端口)+文件路径** (例如： https://127.0.0.1:8080/SearchPage/index.html?username=zzmine&password=123456)。 url地址中？后面的内容叫作 查询字符串 (Query String)

request和response的组成：headers和content，用两次回车换行分隔(\r\n)

## Servlet

[interface] 用于处理请求和发送响应，实现动态页面。

组件不能自己写构造函数，应用服务器会使用默认构造函数！！若要构造，应该**重写**其init(ServletConfig config)函数。HttpServlet已经实现了init()方法。

```
//两种init方法
//1,带参数
public void init(ServletConfig config){
    super.init(config);
    ...
}
//2,不带参数
public void init(){
    ...
}
```

Servlet的service()方法，参数，返回类型以及抛出什么异常。

```
public void service(ServletRequest request, ServletResponse response) throws
ServletException, IOException{
    HttpServletRequest req = (HttpServletRequest)request;
    HttpServletResponse res = (HttpServletResponse)response;
    // HttpServletReuqest的getMethod方法可以获得用户请求的方式(GET,POST等)
    String method = req.getMethod();
    if("GET".equals(method)){
        ...
    }else if("POST".equals(method)){
        ...
    }
```

```
    res.setContentType("text/html;charset=utf-8");
    PrintWriter out = res.getWriter();
    out.print("<p>Hello World</p>");
    // 不建议写流的关闭，还有其他程序需要输出
    // 输出形式为jpeg图片
    res.setContentType("image/jpeg;charset=utf-8");
    OutputStream out = res.getOutputStream();
    out.write(...);
}
```

Servlet的生命周期

```
(1)web容器加载Servlet,生命周期开始。（仅调用一次）说明Servlet是单实例的
(2)调用servlet的init()方法，servlet初始化。（仅调用一次）
(3)调用service()方式，处理请求（被多次调用）
(4)结束服务，web容器调用servlet的destory()方法（仅调用一次）
```

部署(deploy)的方法:

1, 将web应用(HelloServlet)直接复制在Tomcat的WebApps文件夹下, 则用户访问路径：http://localhost:8080/HelloServlet/abc/h.html

2, 在Tomcat配置中的server.xml下部署：在Host元素下建子标签 。path是url，docBase是文件的物理地址。则用户访问路径: http://localhost:8080/abc/abc/h.html

```
<servlet>
    <servlet-name>HelloWorld</servlet-name>
    <servlet-class>com.a.HelloWorld</servlet-class>
</servlet>
<!--备注:同一个Servlet可以被映射到多个URL上，配置多个servlet-mapping-->
<servlet-mapping>
    <servlet-name>HelloWorld</servlet-name>
    <url-pattern>/hello</url-pattern>
</servlet-mapping>
<!--
    url-pattern可以使用通配符，但既有斜杠又有扩展名是不合法的，如/*.html
-->
```

*javax.websocket包下，Endpoint, Encoder, Decoder。双向发送数据，而Servlet是请求应答式单向的服务。

**ServletConfig, ServletContext**

ServletConfig有两个重要的方法：

- getInitParameter(String name) -获得初始化参数name的值，若没有对应于name的参数，那么返回null
- getServletContext() -获得代表应用全局的ServletContext类

ServletContext中有一系列add，set，get方法，用于Servlet与container的交流，**可以当成一个web应用中所有构件的共享资源**

该对象代表当前的web应用，可以从中获取到该web应用的各种信息 ① 配置，获取 web应用的初始化参数 该初始化参数可以被所有Servlet获取，而Servlet的初始化参数只供那个Servlet获取 在< web-app> 中配置 < context-param> servletContext.getInitParameter()

② 获取当前web应用某个文件**在服务器上**的绝对路径，而不是部署前的路径： getRealPath(String path)

③ 获取当前web应用的名称： getContextPath() 若web应用直接放在root目录下，则request.getContextPath()返回为空串。

④ 获取当前web应用的某一个文件对应的输入流 getResourceAsStream(String path) path的 / 是相对于当前web应用的根目录

⑤ 跟attribute相关的几个方法

https://blog.csdn.net/wangqing84411433/article/details/71131608 1、在web.xml配置文件中，对个servlet的配置里，有一项< load-on-startup> < /load-on-startup>，它的含义是：标记容器是否在启动的时候就加载这个servlet。当值为0或者大于0时，表示容器在应用启动时就加载这个servlet；当是一个负数时或者没有指定时，则指示容器在该servlet被选择时才加载。正数的值越小，启动该servlet的优先级越高。

servlet_2虽然配置在servlet_1后面，但是它的load-on-startup为0，启动的优先级高于servlet_1，所以servlet_2先启动。

2、获取ServletContext的两种方式，直接获取 this.getServletContext()和间接获取config.getServletContext，得到的对象都是同一个。同时在servlet1和servlet2中取得的ServletContext对象都是同一个对象，**说明整个web应用都只有一个唯一的ServletContext实例**；

3、servlet1与servlet2的serveletConfig对象是不一样的，说明ServletConfig对象的作用范围仅在servlet中。

**HttpServlet** [abstract class]: 不去重写service()方法，而是去override doGet(), doPost()之类的方法。在多线程中注意并发资源的保护

在doGet()方法中注意处理content type以及encoding

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

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

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

**ServletRequest** [interface]:

**HttpServletRequest** [interface]: 继承ServletRequest，为Http servlet提供请求信息。

该对象提供的数据包括，参数名称和值，属性以及输入流

```
Object getAttribute(String name){ }   // 从request域中取出对应name的数据
String getParameter(String name){ }   // 从url中获取对应name的查询字符串的值，如果name
参数不存在则返回null
String[] getParameterValues(String name){}  // 对应name的多个值
String getMethod(){}                  // 获得请求方式
HttpSession getSession(){}           // 返回Session
ServletContext getServletContext(){}  // ServletRequest中的方法
```

**ServletResponse** [interface]:

**HttpServletResponse** [interface]: 继承ServletResponse，在发送响应式提供特定于http的功能。

用于向客户端返回数据。若要返回字符数据，可以通过getWriter()方法返回的PrintWriter对象来操作，它的write()方法不会抛出异常

```
void setCharacterEncoding(String charset){}    // 设置response的字符编码，如utf-8
void setContentType(String type){ }    // 设置response的内容格式，给定的内容可以包含字符编码规范。例如 “text/html;charset=utf-8”   如果在调用getWriter之前调用此方法，则仅从给定的内容类型设置响应的字符编码
// 注意： This method has no effect if called after the response has been committed. It does not set the response's character encoding if it is called after getWriter has been called or after the response has been committed.
```

# Session

HttpSession Api

可以设置session的生存/超时时间，但注意单位。tiemout通常单位是分钟。

session在内存中如何被创建?

**创建**：sessionid第一次产生是在直到某server端程序调用 HttpServletRequest.getSession(true)这样的语句时才被创建。

**删除**：超时；程序调用HttpSession.invalidate()；程序关闭；

**session存放在哪里**：服务器端的内存中。不过session可以通过特殊的方式做持久化管理（memcache，redis）。

**session的id是从哪里来的**，sessionID是如何使用的：当客户端第一次请求session对象时候，服务器会为客户端创建一个session，并将通过特殊算法算出一个session的ID，用来标识该session对象

**session会因为浏览器的关闭而删除吗?**

不会，session只会通过上面提到的方式去关闭。

*memcached分布式缓存

HttpSession底层实现的三种方法（https://blog.csdn.net/qfs_v/article/details/2652119）：

1. *cookie技术
2. URL重写(rewrite) (浏览器不支持cookie时，可以用URL重写来代替)
3. 隐藏表单域 （form表单有一个隐藏的input）

**Annotation**: 参考https://j2eereference.com/webservlet-annotations/

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

@WebServlet(value="/hello",
initParams = {
   @WebInitParam(name="param1", value="Hello "),
   @WebInitParam(name="param2", value=" World!")
})
public class HelloServlet extends GenericServlet {

    public void service(ServletRequest req, ServletResponse res)
            throws IOException, ServletException
    {
        PrintWriter out = res.getWriter();
        out.println(getInitParameter("param1"));
```

```
        out.println(getInitParameter("param2"));
    }
}
```

## RequestDispatcher

用于请求转发，页面重定向。

```
//Forwards a request from a servlet to another resource (servlet, JSP file, or
HTML file) on the server.
public void forward(ServletRequest request, ServletResponse response) throws
ServletException,IOException
//Include： 将当前请求派发到另一个组件，处理完毕后，回到该组件
public void include(ServletRequest request, ServletResponse response) throws
ServletException, IOException
```

若需要传送数据到另一个组件，不要在response的输入输出流中进行，可以利用request.setAttribute()方法，在request域中存放数据。

注意：路径中'/'表示应用的根目录。request的请求转发是应用组件内的转发。而ServeltContext的转发方法可以在用一个tomcat进程中在不同web应用进行转发。

```
@WebServlet("/Servlet/a");
public class AServlet extends HttpServlet throws...{
    public void service(){
         response.setContentType("text/html;charset=utf-8");
        // ! response.getWriter().print("...1");
        // text/html  MIME类型

        request.setAttribute("obj",obj);
        request.getRequestDispathcer("/Servlet/b").forward(request,response);
        // ! response.getWriter().print("88");
        // 若是forward方法，则不能在response的输出流中输出（该组件不能处理response流），
应该将要输出的对象先存在reqeust域中，再传给另一个组件。
    }
}
```

(必考) response的sendRedirect与request的转发的区别：

```
request的是由服务端进行转发的，浏览器地址栏不变，只发出一次请求
response的是由客户端进行重定向，浏览器地址栏会改变，相当于发多次请求，效率较低
```

## Filter

(必考) Filter，其服务函数doFilter()，在Servlet运行之前执行

```
public void doFilter(ServletRequest r1, ServletResponse r2, FilterChain chain)
throws ServletException{

    long start = System.currentTimeMillis();
    HttpServletRequest request = (HttpServletRequest)r1;
    HttpServletResponse response = (HttpServletResponse)r2;
    // 预处理

    // 过滤后，交给Servlet,最后进行后处理。若没有chain.doFilter()，那么该Filter就会产生拦截，不会访问到后面的Servlet
    chain.doFilter(requeset,response);    // 往后传

    //后处理
    long end = System.currentTimeMillis();
}
```

FilterChain: 用来唤醒下一个filter

```
Filter use the FilterChain to invoke the next filter in the chain, or if the calling filter is the last filter in the chain, to invoke the resource at the end of the chain.
```

按提供的服务分类有：

1. Authentication Filters
2. Logging and Auditing Filters
3. Image conversion Filters
4. Data compression Filters
5. Encryption Filters...

Filter的拦截配置：

```
<filter>
    <filter-name>f4</filter-name>
    <filter-class>a.MyFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>f4</filter-name>
    <servlet-name>hello</servlet-name>    <!--拦截某个servlet-->
    <url-pattern>/sss/*</url-pattern>    <!--url模式拦截-->
</filter-mapping>
```

两个过滤器，EncodingFilter负责设置编码，SecurityFilter负责控制权限，服务器会按照web.xml中过滤器定义的先后循序组装成一条链，然后一次执行其中的doFilter()方法。执行第一个过滤器的chain.doFilter()之前的代码，第二个过滤器的chain.doFilter()之前的代码，请求的资源，第二个过滤器的chain.doFilter()之后的代码，第一个过滤器的chain.doFilter()之后的代码，最后返回响应。

```
// 这个路径仍然是绝对路径
File file = new File("/WEB-INF/contact/a.txt");
// 这个是可以将web应用的根路径转换成操作系统的路径
String s = getServletContext().getRealPath("/WEB-INF/contact/a.txt");
// 若打包到war，这样的路径仍是不对的，可以下面或ClassLoader
InputStream in = getServletContext().getResourceAsStream("/WEB-INF/contact/a.txt");
```

分页功能，List的subList方法，取出子集合

# listener

### JavaWeb的监听器(接口)总共有8个，划分为三种类型

| 类型 | 对应的接口 |
| --- | --- |
| 监听三个作用域创建和销毁<br>(三个作用域:request、session、application) | 作用域request --- **ServletRequestListener**<br>作用域session --- **HttpSessionListener**<br>作用域application --- **ServletContextListener** |
| 监听三个作用域属性状态变更 | request --- **ServletRequestAttributeListener**<br>session --- **HttpSessionAttributeListener**<br>servletContext --- **ServletContextAttributeListener** |
| 监听HttpSession里面存值的状态变更 | **HttpSessionBindingListener**<br>**HttpSessionActivationListener** |

构造listener，要么适用WebListener进行注解声明，要么使用ServletContext的addListener()方法进行添加。

```
@WebListener
public class ApplicationIntializer implements ServletContextListener{
    //在filter以及servlet之前，ServletContextListener会收到servlet
    public void contextInitialized(ServletContextEvent sce){
        ServletContext context = (ServletContext)sce.getServletContext();

        DataSource ds = ...;
    }
    public void contextDestroyed(ServletContextEvent sce){
        ...
    }
}
```

ServletContextEvent仅有一个方法——getServletContext()，返回**改变了的**ServletContext（发送event的那个ServletContext）

它的父类EventObject还有一个getSource()的方法。

HttpSessionBindingEvent的getValue()返回**旧的值**

# Jsp

Jsp本质上是Servlet，是服务器中的组件。tomcat运行jsp时将其转化为.java再编译为.class文件。

JspPage类的生命周期：jspInit(), jspDestroy()

HttpJspPage的服务函数，_jspService(HttpServletRequest request, HttpSerletResponse response) {}

二加一方法：

_jspService()方法依赖于特定的协议，因此在java中并不是通用的方法

```
// time.jsp
<%= new java.util.Date()%>
```

jsp文件内嵌java代码，语法包括，jsp指令，jsp声明，jsp表达式，jsp注释，jsp脚本

```
<%@     %>              jsp指令的语法（三大指令：page,include,taglib）
<%=     %>              jsp表达式的语法
<%!     %>              jsp声明的语法
<%--    --%>            jsp注释
<%      %>              jsp中内嵌java代码 ()
```

（必考）jsp的九个隐式对象：


exception用在错误处理页面，在声明了isErrorPage="true"时，才能使用

（重点1）JspWriter

jsp中的数据和模板都是用JspWriter来输出的，由隐式对象out来引用。

1. 与System.out.println以及response.getWriter().print()的区别：JspWriter会抛出异常，IOException。记得一定要try-catch。而PrintWriter并不会抛出异常。
2. out对象存在一个新的缓冲区

（重点2）PageContext

fascade pattern 门面模式，将复杂的系统功能抽象为一个“门面”——更加友好的接口。PageContext封装了jsp在运行过程中使用的所有对象。将jsp的对象传给java只需要一个pageContext，由此可以获得其他八个隐式对象。

（注意）下面的程序可能会有NullPointerException

```
<%
    String parameter = request.getParameter("page");
    int page = Integer.paraseInt(parameter);
%>
// http://localhost:8080/SearchPage?page=&id=22
// 若url中有page=但没有参数，那么返回空串；若没有page=则返回null
```

（注意）jsp声明一般不声明变量，因为在并发情况下这些变量是不安全的，除非加上syncronized修饰

```
// 页面转发
<jsp:forward page="/servlet/a" />
<%
    request.getRequestDispatcher("/servlet/a").forward(request,response);
    pageContext.forward("/servlet/a");
%>
```

（考）<jsp:include page="/serlvet/a" /> 与 <%@ include file="/serlvet/a"%>的区别，第二个是在编译时包含——静态包含，只能包含静态文件，在jsp转换到serlvet过程中进行包含；而第一个是动态包含。

*useBean转换到java代码

```
<jsp:useBean id="a" class="com.abc.bean.Student"
scope="page|request|session|application" />
// 猜数游戏的javaBean应该在session域中
// jsp中取对象属性
<jsp:getProperty name="a" property="age" />
<%= a.getAge() %>
${a.age}
// 设置属性
<jsp:setProperty name="a" property="age" value="26"/>
<jsp:setProperty name="a" property="*" />
```

注意：javaBean的属性都是小写开头！

```
// (考试坑)
<input name="Age"></intput>
<jsp:setProperty name="a" property="age" param="Age"/>
// 注意param与html中name的大小写！
```

另外，jsp也支持简化的表达式语言（EL），使输出数据更方便${expression}

脚本变量从pageContext, request, session, application这四个作用域中自动依次寻找，如果没有定义，就什么也不输出，不报错。

pageContext：属性作用范围仅限于当前JSP页面 request：属性作用范围仅限于同一个请求 session：属性作用范围限于一次会话，浏览器打开直到关闭称之为一次会话（在此期间会话不失效）
application：属性作用范围限于当前web应用，是范围最大的属性作用范围

EL表达式预定于了11个对象，除了pageContext对象是PageContext类型（考试重点），其余都是Map类型（考试必考）

```
pageContext
param
paramValues
header
headerValues
cookie
initParam
pageScope
requestScope
sessionScope
applicationScope
```

*MVC(模型-视图-控制器)

分页逻辑：定义一个PageBean，对分页所需要的数据进行管理

# 自定义标记

(javax.servlet.jsp.tagext)

```
// 标记处理器类

import ...
import javax.servlet.jsp.tagext.*;

/*public class TimeTag extends TagSupport{

    public int doStartTag() throws JspException{
        // pageContext是TagSupport中的保护成员
        JspWriter out = pageContext.getOut();
        out.println(new Date());
        // 但是此时需要对JspWriter做异常处理
        // 不过这里只能通过try-catch来解决
    }

}*/

// SimpleTag不提供doStartTag()和doEndTag()方法，而是提供了更简单的doTag()方法，有更简单的生命周期和接口

public class ATag extends SimpleTagSupport{
    pubilc String color = "black";
    public void getData(string data){
        this.color = data;
    }

    public void doTag() throws JspException, IOException{
        JspWriter out = getJspContext.getOut();
        ...
    }
}
```

# Tag handler

...

```
// tag的配置  ---创建标签库
// WEB_INF/a.tld
<tag-lib>
    // 给整个标记库定义一个名字
    <uri>Http://www.abc.com/javaee</uri>
```

```
    <tag>
        <name>reset</name>
        <tag-class>ResetTag</tag-class>
        <body-content>JSP</body-content>
    </tag>
</tag-lib>

// 引入到jsp中的方法（必考）
<%@ taglib uri="\WEB-INF\a.tld" prefix="t"%>
// 若用网上下载的，只有jar包，那么使用整体定义的名字,支持打包
<%@ taglib uri="Http://www.abc.com/javaee" prefix="t"%>
```

*xml命名空间

（问答题）与数据库建立连接时，两种类加载方式的区别：

```
// 1, 这种方式需要在编译前就把类放到classpath中
Driver driver = new com.mysql.jdbc.Driver();
// 2, 通过反射机制
Class.forName("com.mysql.jdbc.Driver");
```

# JNDI

（必考）Java命名和目录接口（Java Naming and Directory Interface，JNDI）。主要功能：注册，查找.

JNDI的功能简单说就是可以用简单的方式去查找某种资源。JNDI提供一套标准，服务商去实现SPI，去实现技术细节；程序员去调用API中的方法，不用关心怎么实现的功能。

```
// 利用命名服务来获得DataSource
Context ctx = new InitialContext();
DataSource ds = (DataSource)ctx.lookup("java:comp/env/jdbc/ExamDB");
// lookup中填写的是 java计算机环境变量  其中“java:comp/env/”为命名前缀
```

# Structs 1

a rapid framework for web development based on JavaEE，基于MVC设计模式。降低软件的粒度，提高了Controller的复用性。分为前端控制器和后端控制器。

（曾经考题）structs 框架的配置文件名 不一定要叫 structs-config.xml

前端控制器（ActionServlet），后端控制器（Action）。控制器功能写在execute()函数中。控制器的另一个功能：选择视图输出。

```
前端控制器名称：org.apache.structs.action.ActionServlet
后端控制器名称：org.apache.structs.action.Action
```

Structs1开发步骤：

1. 创建ActionForm的实现，ActionForm是数据表单的对象实现
2. 创建Action的实现，完成其execute()方法

3. 创建数据表单jsp和结果jsp
4. 配置structs-config.xml

创建后端控制器Action，实现execute方法:

```
public ActionForward execute(ActionMapping actionMapping, ActionForm actionForm,
HttpServletRequest request, HttpServletResponse response){

}
```

Action负责页面流程控制、前后台交互。将编写的Action类配置在页面流程控制struts-config.xml：

```
<action  name="xxxForm" path="/xxxAction" scope="request"
type="com.jdon.XxxViewAction">
    <forward name="forward" path="/xxx.jsp" />
</action>
```

*（考）DispatchAction (由汇聚到支流)，降低Action数量

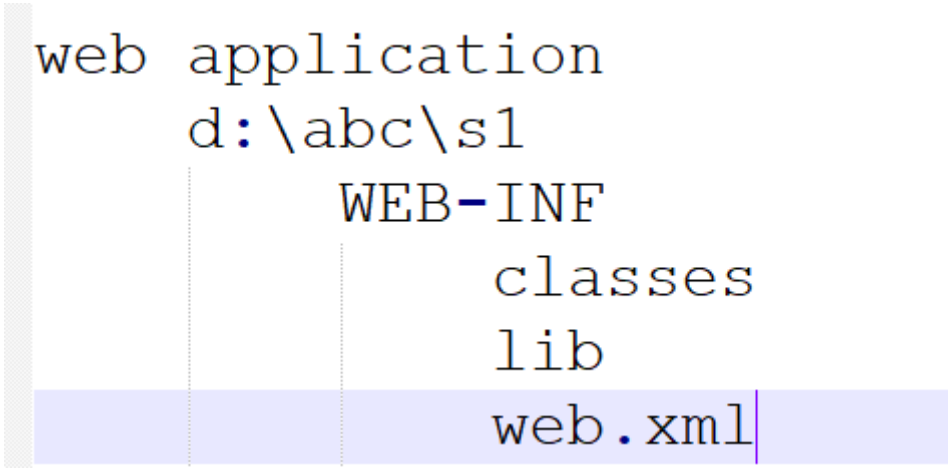
Struts1的工作流程:

- 客户端发送请求（Http Request），被struts1的核心控件器ActionServlet接收，ActionServlet根据 struts-config.xml里的映射关系找到对应的Action，若找不到就返回500错误到JSP页面。若有就在 Action里的 excute()方法里执行相应的逻辑操作，比如调用Model层的方法，然后通过ActionForward，跳转到对应的JSP页面。

# Structs 2

使用structs 2 的步骤：

1. 创建文件夹目录结构：




2. 导入structs2，import struct 2
3. config, deployment

structs2的简单处理流程：

```
1）浏览器发送请求
2）中心处理器（FilterDispatcher）根据struts.xml文件查找对应的处理请求的Action类
3）WebWork的拦截器链自动对请求应用通用功能，例如：WorkFlow、Validation等功能
4）如果Struts.xml文件中配置Method参数，则调用Method参数对应的Action类中的Method方法，否则调用通用的Execute方法来处理用户请求
5）将Action类中的对应方法返回的结果响应给浏览器
```

structs 1 与 structs 2的区别：

```
1）Action类的实现方式：Struts1的Action在实现的时候必须扩展Action类或者Action的子类（DispatchAction），Struts2的Action类实现的时候可以不用实现任何类和接口
2)Struts1的Action类是单例模式，必须设计成线程安全的，Struts2则为每一个请求产生一个实例
3)Struts1的Action类依赖与Servlet API，execute方法有HttpServletRequest和HttpServletResponse，Struts2则不依赖于Servlet API；
4）Struts1的Action与View通过ActionForm或者其子类进行数据传递 而struts2去掉了formbean
5)Struts1绑定了JSTL，为页面的编写带来方便，Struts2整合了ONGL，也可以使用JSTL
```

ResourceBundle 支持国际化，

# Spring

是一个轻量级控制反转（IoC）和面向切面编程（AOP）的容器。

（考试）什么是IoC，DI?

```
IoC(Inversion of Control)控制反转：将对象的生命周期控制，交给第三方容器
DI(Dependency injection)依赖注入：组件之间的依赖关系由容器在运行期决定，即　　由容器动态地将某种依赖关系注入到组件之中
```

*工厂模式， 工厂方法模式， 抽象工厂模式

（考）ClassPathXmlApplicationContext ,FileSystemXmlApplicationContext 前者的xml文件在classpath中（支持打包），后者通过绝对路径指定xml文件。

web应用中创建spring容器，使用XmlWebApplicationContext。听众接口：ServletContextListener，其中两个事件处理函数，

```
// 必考要记
public void contextInitialized(ServletContextEvent sce){
    Context ctx = new InitialContext();
    DataSource ds = (DataSource)ctx.lookup("java:comp/env/jdbc/ExamDB");
    sce.getServletContext().setAttribute("ds",ds);
}
public void contextDestroyed(ServletContextEvent sce){}
```

*（必考）web.xml中声明一个组件：

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

AOP编程

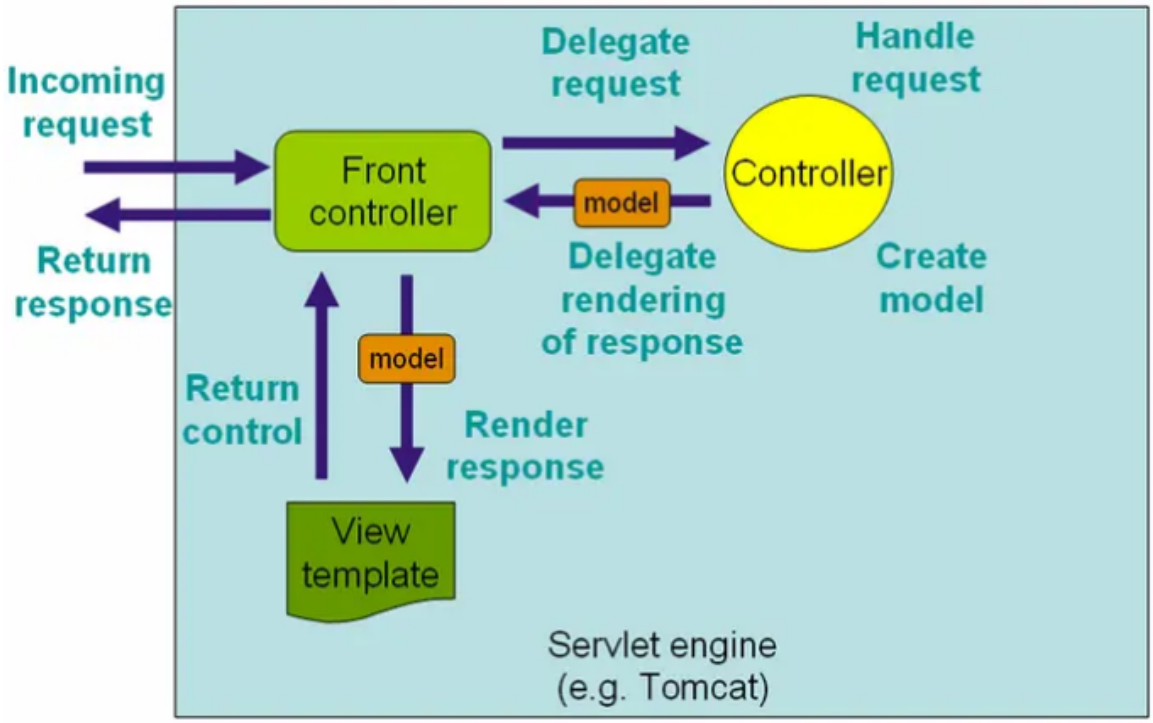
（任务）

```
理解BeanFactory
理解ApplicationContext
理解FileSystemXmlApplicationContext
理解ClassPathXmlApplicationContext
理解XmlWebApplicationContext
重点理解JavaEE Web Application如何集成Spring框架
理解Spring framework的配置
```

spring mvc 前端控制器的mapping 不要使用/* , 而应该使用/ (优先级最低) ,否则会拦截.jsp

# SpringMVC

基本处理流程：




1. 用户请求首先发送到前端控制器DispatcherServlet，DispatcherServlet根据请求的信息来决定使用哪个页面控制器Controller来处理该请求。找到控制器之后，DispatcherServlet将请求委托给控制器去处理。
2. 接下来页面控制器开始处理用户请求，页面控制器会根据请求信息进行处理，调用业务层等等，处理完成之后，会把结果封装成一个ModelAndView返回给DispatcherServlet。
3. 前端控制器DispatcherServlet接到页面控制器的返回结果后，根据返回的视图名选择相应的视图模板，并根据返回的数据进行渲染。
4. 最后前端控制器DispatcherServlet将结果返回给用户。

备注：

1. DispatcherServlet相当于前端控制器: org.springframework.web.servlet.DispatcherServlet
2. Handler（需要自己开发）是继DispatcherServlet前端控制器的后端控制器，在DispatcherServlet的控制下Handler对具体的用户请求进行处理。
3. 前端控制器的url-pattern应该是/，而不是/*

Spring的后端控制器可以用xml配置，也可以用注解来声明

```
package com.abc.project.controller;
```

```
@Controller
public class MyController{
    // 路径到处理函数的映射
    @RequestMapping("/calc/add")
    public String add(@RequestParam(value="v1") String v1,
                      @RequestParam(value="v2") String v2){

    }
}

在配置文件中开启mvc
<mvc:annotation-driven/>
在配置文件中启动扫描组件，注意是包的名字
<context:component-scan base-package="com.abc.project.controller"/>
```

MVC 各自的作用以及关系

```
View 访问模型，显示模型层的内容
Controller 前端控制器：访问View(可复用) 后端控制器：访问模型(不可复用)
Model 模型层主要负责保存和访问业务逻辑，执行业务逻辑和操作。（不可复用）
特点：
1）多个视图可以对应一个模型，增强了了代码的重用性，减少代码量，易于维护
2）分离视图层和业务逻辑层，可维护性高
3）降低了各层之间的耦合性，提供了应用的可扩展性
4）MVC更符合软件工程化管理的精神，各层各司其职
5）使用MVC模式使开发时间得到相当大的缩减，部署加快
```

# Hibernate

是一个对象关系映射框架，对JDBC进行了轻量级的对象封装。可以自动生成SQL语句，自动执行。

Hibernate使用**反射机制**实现持久化对象的操作。