## Jsp

+++++++++++++++++++++++++++++++++++++++++++++++++++++

绝对路径

${pageContext.request.contextPath} // el表达式取法

request.getContextPath();

// 顺带着附赠两个

request.getServletPath(); 取项目名后的路径 （直到.jsp）

request.getRequestURI(); 这个方法取得的路径最为完整，从项目名到 .jsp

+++++++++++++++++++++++++++++++++++++++++++++++++++++

Import语法

<%@ page import=*"java.util.\*, com.jojo.model.\*"*%>

中间用逗号隔开

+++++++++++++++++++++++++++++++++++++++++++++++++++++

JSTL

// 为什么他们的关闭方式都不一样啊！

***C标签***

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

* <c:forEach></c:forEach>

遍历list

<c:forEach var=*"comment"* items=*"*${positiveList }*"*></c:forEach>

遍历map

* <c:if>

<c:if test=*"*${ }*"* var=*""* scope=*""*></c:if>

判空

<c:if test=*"*${**not empty** \_SESSION\_KEY\_CURRENT\_MENU}*"*>

* <c:choose>、<c:when>、<c:otherwise>

// 高级版c:if

<c:choose>

<c:when test=*"*${**not empty** generalBibliosList}*"*>

</c:when>

<c:otherwise>

</c:otherwise>

</c:choose>

***Fmt标签***

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

<fmt:formatDate type=*"date"* pattern=*”yyyy-MM-dd”* value=*"*${}*"* />

很他妈吊诡，我这么写的时候啊，居然是12小时制，把pattern删掉后，就变成了24小时制，很他妈吊诡啊。

<fmt:formatDate type=*"both"*

value=*"*${generalHolding.createtime}*"*

pattern=*"yyyy-MM-dd hh:mm:ss"*>

</fmt:formatDate>

h   12   小时制的小时。一位数的小时数没有前导零。      
  hh   12   小时制的小时。一位数的小时数有前导零。      
  H   24   小时制的小时。一位数的小时数没有前导零。      
  HH   24   小时制的小时。一位数的小时数有前导零。       
 QQ图片20141219230058

// 还有不懂的去这个网站看详细数据

http://www.jb51.net/article/45362.htm

++++++++++++++++++++++++++++++++++++++++++++++++++

EL表达式

${param.userName} // 取请求参数的值（url后面的）

${pageScope.userName} // pageContext.setAttribute()

${requestScope.userName} // requestScope.setAttribute()

${sessionScope.userName} // sessionScope.setAttribute()

${applicationScope.userName} // applicationScope.setAttribute()

${map[“userName”]} // 取map中的值

${list[0]} // 取集合中的值

////////////////////////////////////////////////////////////////////////////////////////

## Servlet

Session中变量的生存周期更改

1. web.xml

// 若为0或负数则永不失效，只针对某项目

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

1. 程序设置

// 程序设置，但仅设置当前会话

session.setMaxInactiveInterval(30 \* 60);

1. Server.xml

// 针对服务器，全项目

<Context path="/test" docBase="/test"

　　defaultSessionTimeOut="3600" isWARExpanded="true"

　　isWARValidated="false" isInvokerEnabled="true"

　　isWorkDirPersistent="false"/>

++++++++++++++++++++++++++++++++++++++++++++++++++

初始参数的配置

这些东西通通放在web.xml中，使用标签配置。因为这样方便修改，使用代码的话，一旦修改就会造成重新编译。

// 哎，我真的是好久没写过这种代码了，手生啊。

<servlet>

<servlet-name><servlet-name>

<servlet-class><servlet-class>

<init-param>

<param-name></param-name>

<param-value></param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name></servlet-name>

<url-pattern></url-pattern>

</servlet-mapping>

初始参数保存在ServletConfig对象里，注意ServletConfig是接口不是类。这个接口所有Servlet都实现了。

ServletConfig sc = this.getServletConfig();

Enumeration params= sc.getParameterNames();

while(params.hasMoreElements()){

sc.getParameter((String)params.next()); // 要强转

}

其实Servlet也有getInitParameter(String name)方法，内部是这样实现的：

**return** getServletConfig().getInitParameter(name);

这种就比较适合提取单个参数，提多个参数的话效率低。

++++++++++++++++++++++++++++++++++++++++++++++++++

老实讲，我不觉得我有多懂Servlet，虽然写了个订餐管理系统，但仔细想想，真得没什么技术含量在里面。因为我只是很单纯的处理request，返回response，Servlet特性什么的几乎没用到。就看看我用了哪些类吧：

HttpServlet doGet()

doPost()

HttpServletRequest request.getParameter()

request.setAttribute()

request.getRequestDispatcher().forward()

request.getContextPath()

HttpServletResponse response.sendRedirect()

response.getWriter() // 好像不对

HttpSession session.setAttribute()

session.invalidate()

然后就没了，就算还有，但我现在也想不出来。所以，我根本就不懂Servlet，我只是了解。

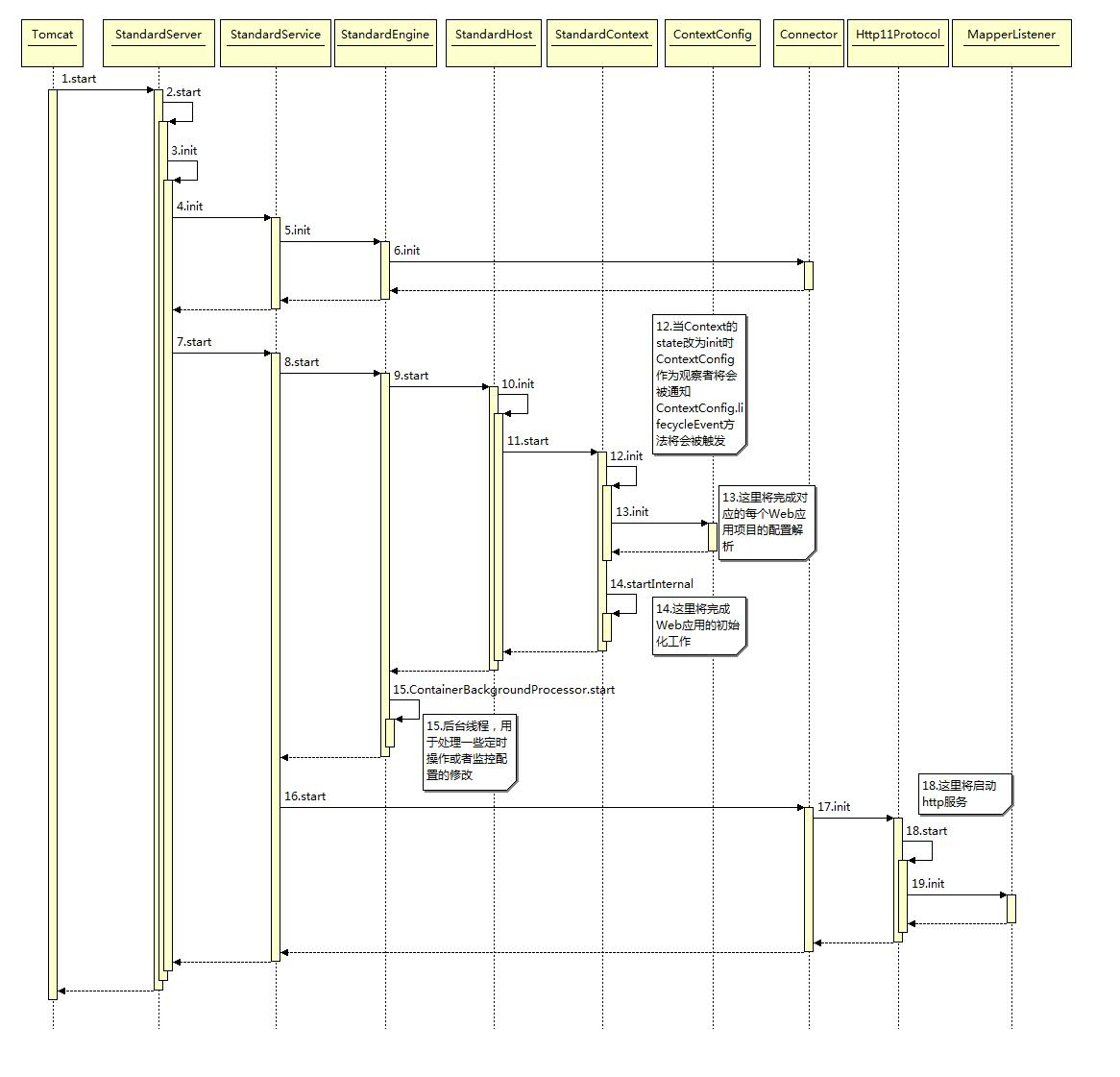
现在OK，nice，找到了一篇好文章，Servlet工作原理解析，来自IBM developers。

照作者原话，理解Servlet要先搞懂Tomcat，要我说这是一个DFS和BFS谁优谁劣的问题，这次我站BFS，毕竟毛都不懂还能写项目呢。。。

Tomcat容器模型共有四个等级，从高到低分别是Container、Engine、Host、Context。其中真正管理Servlet的是Context容器，一个Context容器就对应一个web project，这也是为什么取绝对路径是request.getContextPath()方法。

因此，当一个web project被添加到Tomcat时，Tomcat会创建一个StandardContext容器，并给这个容器设置必要的参数如url（访问路径）、path（实际物理路径）。最重要的配置是ContextConfig，这个类会负责整个web project的配置解析工作。各项参数准备妥当后这个Context容器将被添加至父容器Host中。

最后Tomcat启动，这里有一个时序图，东西还是比较多的。



// warning，往下我要加私货了，不一定对的私货！

Context容器初始化时会启动ContextConfig，这个类会负责整个web project的配置文件的解析工作。该类会完成以下几个工作：

1.创建用于解析xml配置文件的contextDigester对象，类是这个org.apache.tomcat.util.digester，另外我思前想后还是不准备下Tomcat的文档。

2. 读取默认的context.xml文件，如果存在解析它

3. 读取默认Host配置文件，如果存在解析它

4. 读取Context自身的配置文件，如果存在解析它

5. 设置Context的DocBase

DocBase是文件实际存在的物理路径，说来搞笑，server.xml里<context>标签下path代表访问路径，docbase代表物理路径，然后到创建StandardContext容器时，path又成了物理路径，用url代表访问路径，妈的你们真的是一个项目组的吗。

ContextConfig结束后，StandardContext对象会执行startInternal()方法，执行内部，会做以下几个事：

1. 创建读取资源文件的对象
2. 创建ClassLoader对象
3. 设置应用的工作目录
4. 启动相关的辅助类：logger等
5. 修改启动状态，通知感兴趣的观察者
6. 子容器的初始化
7. 获取ServletContext并设置必要的参数
8. 初始化<load-on-startup>的Servlet

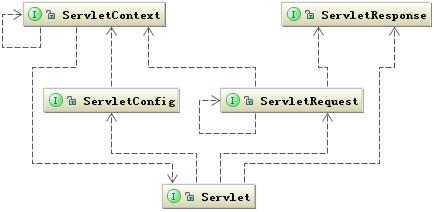
嗯，梦游着打完了这几句话。。。。

下面到web project的初始化，这个过程是在ContextConfig的configureStart方法中实现的，具体情况参照IMG_256那个时序图。项目的初始化主要是解析web.xml文件，这个文件描述了web project的关键信息，也是一个web project的入口。相应的也很臃肿，不过struts.xml里东西好像更多啊。

web.xml文件中的各个配置项将会被解析成相应的属性保存在WebXml对象中，接下去会将WebXml对象中的属性设置到Context容器中，包括创建Servlet，filter，listener对象。但Servlet什么的不是直接添加至Context中的，它会先被包装成wrapper。

此处省略Servlet实例的创建过程，及初始化过程，等真正有能耐了再说吧。

Servlet的顶层类关联图（这个箭头标的。。。）



和Servlet直接关联的类有ServletConfig、ServletRequest、ServletResponse。其中，ServletConfig在Servlet初始化时就传给Servlet了（详情见GenericServlet源码）。ServletConfig接口只有4个方法：

getInitParameter()、getInitParameterNames()、getServletName()、getServletContext()

ServletContext接口，对应JSP application对象，用来与Servlet Container也就是Context交互，像addServlet()、addFilter()、addListener()等方法。通过ServletContext又可以拿到Context容器中一些必要的信息，比如工作路径，但这个工作路径和request.getContextPath()又是什么区别呢？这个接口还是以后再说吧。

ServletRequest接口，获取设置数据（attribute，parameter），获取设置字符集，获取本地/远程地址、host、端口，获取服务器名字端口，还有getInputstream。HttpServletRequest在这些基础上增加了Http协议的内容，cookie，session，header。

ServletResponse接口，印象最深的就是setCharacterEncoding然后再getOutputStream接着发送json数据。此外还负责设置响应体的长度，类型。HttpServletResponse接口与HttpServletRequest同理，多了些http协议的内容，addCookie，addHeader等。还有一堆静态常量（状态码）。

Servlet接口，只有5个方法init（注意，这个init是带参数的，形参为ServletConfig），service，destroy，getServletConfig，getServletInfo。GenericServlet在实现Servlet后有添加了日志相关的方法，重载了init，并直接添加了获取初始参数的方法，原理与ServletConfig是一致的。HttpServlet就是添加了Http协议的内容，即8种请求方法（put，delete，get，post，...）。

所以看到现在，明白了吗，HTTP协议赛高啊！

////////////////////////////////////////////////////////////////////////////////////////

## Listener

老夫仔细研究了J2EE中与Servlet Listener有关的接口（其实就是用listener作为关键词查了一下。。。。）大致有这么9个。

javax.servlet.http.HttpSessionActivationListener

// 这个接口吊了，没听说过，session的钝化与激活，为了支持分布式，不常用的session被写入持久设备（比如硬盘）。。。bla bla 然而我还是不懂

javax.servlet.http.HttpSessionAttributeListener

javax.servlet.http.HttpSessionBindingListener

javax.servlet.http.HttpSessionIdListener

javax.servlet.http.HttpSessionListener

javax.servlet.ServletContextAttributeListener

javax.servlet.ServletContextListener

javax.servlet.ServletRequestAttributeListener

javax.servlet.ServletRequestListener

其他的还有，但一时半会也看不懂。找出的9个接口里可以很明显的看出是监听session，application（ServletContext），request。监听的内容一般是创建、销毁、属性改变事件。

接口里有3个方法的肯定是Attribute相关的，只有这玩意会有创建，销毁，更改这三种状态。

接口里只有1个方法的是HttpSessionIdListener，当session id改变时会bla bla。

// 人生几大错觉之一，我懂了。。。。

那么问题来了，什么是session id呢？session是一段会话，指用户访问网站这个过程。但，用户又不止一个，服务器怎么确保每次都能正确处理某个用户的session呢。这就要用到session id，session id可以唯一标识一段session会话，一般保存在服务器内存或客户端cookie中。

所以完整的过程是这样的，客户端发出请求，服务器先检查cookie中有没有session id。若没有，创建session并将相应的session id插入至cookie中；若有，则直接根据这个session id从服务器中将session检索出来（没找到会再次新建）。不过有些用户会禁cookie，这时就要重写url或用隐藏表单域之类的方法。

// 老实讲，我到现在都不懂干嘛非要<servlet-name>这个标签，就不能直接让<url-pattern>与<servlet-class>对应吗？

最后是web.xml的配置，相对于Servlet与filter，listener的配置简直良心。

<listener>

<listener-class>

com.jojo.web.MyListener

</listener-class>

</listener>

是的，这就没了（我花了好久才说服我自己 \_(:з」∠)\_ ）

-- 所以有时候人也不能太死板，那个接口就两三个方法又不意味着我新建的Listener类也只能有这两三个方法，搞编程，尤其忌讳思维定式啊。工资高的关键在于你对这家公司有不可替代性。--2017.1.24

////////////////////////////////////////////////////////////////////////////////////////

## 复选框批量删除

原理说难不难，可关键是我确实没想到。现在就从前后端两个角度细致的讲解这个过程。

前端主要是获取并发送要批量删除的数据（一般是id），难点主要在于一次获取多个。其实现在想想，要一次获取多个值，就跟一次打印多个值一样，肯定是用循环处理问题，跑不了的。其实这个问题细化一下是这样的，“一次选中多个‘被选中’的复选框”。

var arr = [];

$(“input[name=’xxx’]:checked”).each(function(){

arr.push($(this).val());

});

var str = arr.join(“ , ”)；

然后把这个字符串传回后台，并用MySQL批量删除。