# JAVAWEB

## web路径

访问页面地址：http://127.0.0.1:8080/jyl/

页面中包含连接：resources/jyl.jpg，等价于http://127.0.0.1:8080/jyl/resources/jyl.jpg

## maven

### 配置

#### 依赖获取

依赖查询地址：

<http://mvnrepository.com/>

#### 环境变量

M2\_HOME，MAVEN\_HOME （部分项目使用）：maven所在目录

path \*\*\*\*\*\bin

#### 检查安装

命令行输入mvn -v

#### 本地仓库目录

maven的conf目录下：settings.xml

设置<localRepository>标签的仓库目录（默认为${user.home}/.m2/repository）：

<localRepository>C:\JaveWeb\mvn\repository</localRepository>

#### 中央仓库镜像

<mirror>

<id>nexus-aliyun</id>

<mirrorOf>central</mirrorOf>

<name>Nexus aliyun</name>

<url>http://maven.aliyun.com/nexus/content/groups/public</url>

</mirror>

#### 本地依赖

部分依赖在中心仓库无法找到，需要本地配置

##### 安装方式

下载kaptcha并解压到c:\kaptcha-3.2.5.jar，运行以下命令

mvn install:install-file -Dfile=c:\kaptcha-3.2.5.jar -DgroupId=com.google.code -DartifactId=kaptcha -Dversion=3.2.5 -Dpackaging=jar

即可使用：

<dependency>

<groupId>com.google.code</groupId>

<artifactId>kaptcha</artifactId>

<version>2.3.2</version>

</dependency>

表示仓库在/com/google/code/kaptcha/2.3目录

##### 相对路径方式

直接放在了项目的lib文件夹

<dependency>

<groupId>com.abc</groupId>

<artifactId>def</artifactId>

<scope>system</scope>

<version>1.0</version>

<systemPath>${basedir}\src\lib\def.jar</systemPath>

</dependency>

### 错误处理

正常运行，xml异常

相应包的\_remote.repositories修改：

json-20180130.pom>nexus-aliyun=

json-20180130.jar>nexus-aliyun=

json-20180130-sources.jar>nexus-aliyun=

变为：

json-20180130.pom>

json-20180130.jar>

json-20180130-sources.jar>

### 版本定义

<properties> <spring.version>4.2.4.RELEASE</spring.version>

<version>${spring.version}</version>

### eclipse配置

window-》preferences-》Maven

Installations设置maven所在目录

User Settings设置仓库的settings.xml所在目录

All Catalogs选择org.apache.maven.archetypes maven-archetype-webapp 1.0

Grop Id: com.\*\*\*

Artifact Id: PrjName

无法导入时：mvn eclipse:eclipse -Dwtpversion=2.0

eclipse打开一个workspace，对eclipse的设置是对应该worksapce的配置。

在导入maven项目前，设置仓库地址。

Window->Preferences->Maven->User settings

默认是用户目录下的.m2目录下的settings.xml，在该位置可以自定义settings.xml

通过该设置可以定位本地的仓库目录

### IDEA配置导出war

Mave Projects -> Lifecycle -> package

### 导出jar运行包

#### POM导出信息

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-shade-plugin</artifactId>

<version>1.2.1</version>

<executions>

<execution>

<phase>package</phase>

<goals>

<goal>shade</goal>

</goals>

<configuration>

<transformers>

<transformer

implementation=*"org.apache.maven.plugins.shade.resource.ManifestResourceTransformer"*>

<mainClass>com.test.MainClass</mainClass>

</transformer>

</transformers>

</configuration>

</execution>

</executions>

</plugin>

</plugins>

</build>

#### POM属性版本

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

</properties>

#### eclipse导出运行

右键工程-》Run As -》Maven install

项目的target目录中运行：

java -jar \*\*\*\*\*\*-0.0.1-SNAPSHOT.jar

### 私服

#### nexus配置

https://www.sonatype.com/download-oss-sonatype

参数修改：

修改数据存储路径，文件目录：bin/nexus.vmoptions

修改IP、端口、访问根目录，文件目录：etc/nexus-default.properties

#### 启动

运行bin目录下的exe

./nexus.exe /run

默认端口8081

<http://127.0.0.1:8081>

使用命令可以安装服务：

nexus.exe /install

#### 仓储设置

使用管理员账号登录，默认登录名及密码：admin admin123

maven-central：maven中央库，默认从https://repo1.maven.org/maven2/拉取jar

maven-releases：私库发行版jar

maven-snapshots：私库快照（调试版本）jar

maven-public：仓库分组，把上面三个仓库组合在一起对外提供服务，在本地maven基础配置settings.xml中使用。

##### maven配置

settings.xml文件内容：

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

<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.apache.org/xsd/settings-1.0.0.xsd">

<localRepository>C:\JaveWeb\mvn</localRepository>

<pluginGroups>

<pluginGroup>org.sonatype.plugins</pluginGroup>

</pluginGroups>

<proxies>

</proxies>

<servers>

<server>

<id>nexus</id>

<username>admin</username>

<password>admin123</password>

</server>

</servers>

<mirrors>

<mirror>

<id>nexus</id>

<mirrorOf>\*</mirrorOf>

<url>http://localhost:8081/repository/maven-public/</url>

</mirror>

</mirrors>

<profiles>

<profile>

<id>nexus</id>

<repositories>

<repository>

<id>central</id>

<url>http://central</url>

<releases><enabled>true</enabled></releases>

<snapshots><enabled>true</enabled></snapshots>

</repository>

</repositories>

<pluginRepositories>

<pluginRepository>

<id>central</id>

<url>http://central</url>

<releases><enabled>true</enabled></releases>

<snapshots><enabled>true</enabled></snapshots>

</pluginRepository>

</pluginRepositories>

</profile>

</profiles>

<activeProfiles>

<activeProfile>nexus</activeProfile>

</activeProfiles>

</settings>

##### 工程配置pox.xml

<distributionManagement>

<repository>

<id>nexus</id>

<name>Releases</name>

<url>http://localhost:8081/repository/maven-releases</url>

</repository>

<snapshotRepository>

<id>nexus</id>

<name>Snapshot</name>

<url>http://localhost:8081/repository/maven-snapshots</url>

</snapshotRepository>

</distributionManagement>

<build>

<defaultGoal>compile</defaultGoal>

<finalName>page</finalName>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>2.4.2</version>

<configuration>

<skipTests>true</skipTests>

</configuration>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.3</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

#### 编译项目

项目右单击->Run As->Maven build..

在界面中填写Goals为：deploy -e

POM配置为<version>0.0.1-SNAPSHOT</version>，则编译到私服的maven-snapshots里。<version>0.0.1</version>则编译到maven-release里。

#### 上传jar包

mvn deploy:deploy-file

-DgroupId=com.aliyun.oss -DartifactId=aliyun-sdk-oss -Dversion=2.2.3 -Dpackaging=jar -Dfile=D:\aliyun-sdk-oss-2.2.3.jar -Durl=http://127.0.0.1:8081/repository/maven-3rd/ -DrepositoryId=nexus-releases

## Tomcat

### 配置

webapps中存放web项目文件

conf下server.xml中 <Connector connectionTimeout="20000" port="8080" protocol="HTTP/1.1" redirectPort="8443"/>

定义连接端口,默认8080

### 运行

运行环境需要配置环境变量：JRE\_HOME

可以通过tomcat下bin目录中startup.bat启动

默认打开index文件

http://127.0.0.1:8080/webPrjName/

项目的根目录就是该地址（项目的webapp目录）

导出WAR文件，放在tomcat的webapps目录下，使用bin目录下startup.bat运行

tomcat运行环境的conf目录下server.xml Host标签可以设置项目文件目录:

<Context docBase="E:\apache-tomcat-9.0.0.M21\webapps\webTest" path="/webTest" reloadable="true" source="org.eclipse.jst.jee.server:webTest"/>

### Eclipse配置

==============加入tomcat环境

Window->Preferences->server->runtime environment

add... apache Tomcat v9.0

设置tomcat installation directory(bin目录上级)

===============web项目配置

右键工程build path libraries页签

add libray...

Server Runtime 选择tomcat版本

JRE System Library 选择jre版本

### IDEA配置

不需要安装tomcat，会自动自动下载tomcat插件相关相关内容

pom.xml:

<plugin>

<groupId>org.apache.tomcat.maven</groupId>

<artifactId>tomcat7-maven-plugin</artifactId>

<version>2.2</version>

<configuration>

<port>8080</port>

<path>/</path>

<uriEncoding>UTF-8</uriEncoding>

<server>tomcat7</server>

</configuration>

</plugin>

Edit Configurations.. -> + ->Maven

Name: web

Command line:tomcat7:run

## javaWeb

### JSP元素

========指令标识，java代码，jsp表达式使用

<%@ page import = *"java.text.SimpleDateFormat"* %>

<%@ page import = *"java.util.Date"* %>

<%

Date date = **new** Date();

SimpleDateFormat simpleDateFormat = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

String time = simpleDateFormat.format(date);

%>

body中可以通过jsp表达式直接使用java代码：

curTime=<%=time %>

=======jsp代码嵌套html

body中可以用java代码逻辑嵌套html

<% if(false){ %>

<p>true</p>

<% }else{ %>

<p>false</p>

<% } %>

#### java代码

<% \*\*\*\* %>

当前页面有效，页面关闭时销毁

#### 指令标识

<%@ 指令名 属性1=“val1” %>

##### page

定义整个jsp页面相关属性，指令属性包括：

language： <%@ page language="java" %> 目前只包含java

extends：所有jsp页面在执行之前都会被服务器解析成servlet，而servlet是由java类定义的，所以jsp和servlet都可以继承指定的基类。该属性可以设置jsp页面继承的java类。

import：用于导入java代码使用的java类包 <%@page import="java.util.List" %>

pageEncoding：指定页面编码格式

contentType：设置jsp页面的MIME类型和字符编码 <%@ page contentType="text/html; charset=utf-8"%>

session：指定当前页面是否使用session会话对象，默认true

buffer：设置jsp的out输出对象使用的缓冲区大小，默认8kb，单位只能使用kb。

autoFlush：设置jsp页面缓存满时是否自动刷新。默认值true。如果使用false，则缓冲满时抛出异常。

isErroPage：可以将当前jsp页面设置成错误处理页面来处理另一个jsp页面的错误。

errorPage：指定一个目标jsp页面来处理当前页面错误，目标jsp页面必须设置isErroPage属性为true。该属性的值是一个url。

##### include

可以引入另一个jsp,将另一个jsp在当前页面展开合并为一个jsp.

<%@ include file=”\*\*.jsp” %>

被包含的jsp除了必要内容，只包含<%@ page pageEncoding="utf-8"%>即可

##### taglib

可以引入标签库

#### JSP表达式

<%= 表达式 %>

表达式可以是任何完整的java表达式。结果被转为字符串

#### 注释

<%-- 注释 --%>

该注释方法在显示时是隐藏的

对代码片断的注释

<!-- <%= new Date() %> -->

动态注释

<!-- <%=new Date() %> -->

/\*\* 说明内容

\*/

在方法或变量上加入/\*\*注释可以被javadoc获取，当鼠标移动到该方法或变量时会提示注释内容

#### 声明标识

<%! 全局变量或方法 %>

服务器执行JSP页面时，会将JSP页面转换为Servlet类，把页面中定义的全局变量和方法转换为类的成员变量和方法。

声明的标识一直有效，当前页面有效，服务器关闭时销毁

### 动作标识

#### 包含文件

<jsp:include page="Def.jsp"></jsp:include> page属性支持jsp表达式

<%@ include file=”\*\*.jsp” %>一样可以包含其它jsp，file属性不支持任何表达式，该指令是将文件原样插入，合成的最终文件再进行编译

不同点在于jsp:include是分开编译,是在不同页面执行进行返回.不存在变量同名问题.

#### 请求转发/参数传递

<jsp:forward page="login.jsp">

<jsp:param value="admin" name="user"/>

</jsp:forward>

中转页面,直接跳转到login.jsp ,将参数user=admin传递

等价于跳转时文件名后直接加 ?user=admin

### 内置对象

Request是客户端请求的封装，服务器用reques取出请求信息。服务器接受请求后添加信息到response，用来返回给浏览器。

#### request

封装了由客户端传来的html请求

|  |  |
| --- | --- |
| **功能** | **函数** |
| **获取传递参数** | String val = request.getParameter("x") |
| **获取客户端地址** | getRequestURL () |
| **获取客户端地址，不包括参数** | getRequestURI() |
| **获取请求方式(post/get)** | getMethod() |
| **获取http协议** | getProtocol() |
| **获取客户端ip** | getRemoteAddr() |
| **获取客户端名称** | getRemoteHost() |
| **获取服务器名称** | getServerName() |
| **获取服务器端口** | getServerPort() |
| **获取客户端语言类型** | java.util.Locale locale = request.getLocale();  String ch = "";  if(locale.equals(java.util.Locale.CHINA))  ch="china"; |

#### response

重定向网站

response.sendRedirect("https://www.baidu.com");

禁用缓存

response.setHeader("Cache-Control","no-store");

response.setDateHeader("Expires",0);

自动刷新

response.setHeader("refresh","1"); 每秒刷新一次

定时跳转

response.setHeader("refresh","2;https://www.baidu.com");

设置缓冲区32kb

response.setBufferSize(32);

#### cookie

cookie会放在客户机器，创建成功后每次打开网页都可以读取cookie

======创建

<%

Cookie cookie = new Cookie("name","jyl");

cookie.setMaxAge(60\*60\*24\*10);//有效期10天

response.addCookie(cookie);

%>

======读取

<%

Cookie[] cookies = request.getCookies();

String val = "";

if(cookies != null)

{

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

{

val = cookies[i].getValue();

val = cookies[i].getName();

}

}

%>

#### session

session可以保存用户状态，直到关闭浏览器，时间超时session对象会自动消失

设置属性：

session.setAttribute("name","xx");

获取属性（obj对象）：

session.getAttribute("name")

移除属性

session.removeAttribute("name");

删除会话（删除后调用session对象会抛出异常）

session.invalidate();

#### application

保存所有应用程序中的公共数据，在服务器启动时自动创建，在服务器停止时销毁。

========读取初始化参数

web应用目录下WEB-INF目录下web.xml文件

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

<web-app xmlns="http://java.sun.com/xml/ns/javaee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee

http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"

version="3.0"

metadata-complete="true">

<context-param>

<param-name>val</param-name>

<param-value>111</param-value>

</context-param>

</web-app>

====获取初始值

String ret = application.getInitParameter("val");

====获取初始所有键值对

java.util.Enumeration enema = application.getInitParameterNames();

while(enema.hasMoreElements())

{

String name = (String)enema.nextElement();

String value = application.getInitParameter(name);

}

======设置和获取全局参数

application.setAttribute("a",1);//第二个参数是obj

String str = application.getAttribute("a").toString();

#### out

向浏览器客户端输出信息

out.print(out.getBufferSize());

out.println("xxx");

#### pageContext

pageContext容器内置对象,可以通过该对象获取response/session/application/exception等对象

包括forward，setAttribute等方法

#### config

通过pageContext对象的getServletConfig()方法可以获取一个config对象,用来读取web.xml配置信息

getInitParameter()/getInitParameterNames()

#### page

page对象代表jsp本身,只有在jsp页面内才合法.是包含当前ServLet接口引用的变量,this关键字的别名.

page.getClass()返回当前Object类

#### exception

如果jsp页面中没有捕捉到异常，就会生成exception对象，并把exception传输到指定的错误页面。

指定的错误页面在page指令中设置isErrorPage属性为true的页面才可以被使用（一般的jsp页面中使用无法编译）。

<%@page errorPage="e.jsp" %>

<%@page isErrorPage="true" %>

### javaBean

javaBean类需要遵守规范：

提供一个默认的无参构造函数。

可能有一系列可读写属性,首字母必须小写

可能有一系列的"getter"或"setter"方法。

=====创建bean类

**package** com.test;

**public** **class** bb {

**private** String val = **null**;

**public** String getVal() {

**return** val;

}

**public** **void** setVal(String val) {

**this**.val = val;

}

}

=====使用

<%--创建对象objID --%>

<jsp:useBean id=*"ObjID"* class=*"com.test.bb"*/>

<%--设置对象objID的val属性 --%>

<jsp:setProperty name=*"ObjID"* property=*"val"* value=*"aaabbbccc"*/>

<%--显示 --%>

val = <jsp:getProperty name=*"ObjID"* property=*"val"*/>

### EL表达式

开启el表达式（默认可能是关闭的）

<%@page isELIgnored="false" %>

${\*\*\*}

等价于<%=request.getAttribute("\*\*\*")%>

jsp页面中输出$符号时:\${ 或者 ${"${"}

可以访问pageContext对象

除0后返回Infinity,不返回错误

=======禁用

禁用后将原内容输出

禁用一个表达式:\${\*\*}

禁用所有页面(web.xml):

<jsp-config>

<jsp-property-group>

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

<el-ignored>true</el-ignored>

</jsp-property-group>

</jsp-config>

====判空

判断对象是否是null或者””

<% request.setAttribute("nn", ""); %>

${**empty** nn}

====访问作用范围的隐含对象

<% request.setAttribute("nn", ""); %>

需要使用requestScope.nn

不能通过request.nn访问到保存到request范围内的变量nn

根据设置的访问范围进行获取

<jsp:useBean id="tt" scope="page" class="com.wt.tt"></jsp:useBean>

${ pageScope.tt.val}

====访问环境中的值

获取提交的标单中txt变量的值

${param.txt}

获取提交的标单中一组值

${paramValues.ck[0]}

${paramValues.ck[1]}

## JSTL

### 依赖

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

<version>1.2</version>

</dependency>

<dependency>

<groupId>taglibs</groupId>

<artifactId>standard</artifactId>

<version>1.1.2</version>

</dependency>

### 使用

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

|  |  |
| --- | --- |
| <c:set var="pageurl" value="${pageContext.request.contextPath}"></c:set> | 获取运行路径  使用：  '${pageurl}/resources/icons/x.png' |
| <c:out value="xxx"/>  <c:out value=”${aaa}”/>  <c:out value=”${aaa}” default=”xxx”/> | 输出字符串 |
| <c:set var=”aaa” value=”xxx”/> | 在pageContext中添加值 |
| <c:set var=”a” value=”hello” scope=”session”/> | 在session中添加值 |
| <c:remove var="a" scope=”page”/> | 删除pageContext中数据  无scope时删除所有域中的数据 |
| <c:if test="${booleanVal}"> \*\*\*\* </c:if>  <c:if test="${strval.equals('ss')}"> \*\*\*\* </c:if> | if语句 |
| <c:set var="score" value="${param.score }"/>  <c:choose>  <c:when test="${score > 100 || score < 0}">错误的分数：${score }</c:when>  <c:when test="${score >= 90 }">A级</c:when>  <c:when test="${score >= 80 }">B级</c:when>  <c:when test="${score >= 70 }">C级</c:when>  <c:when test="${score >= 60 }">D级</c:when>  <c:otherwise>E级</c:otherwise>  </c:choose> | if/else if/else结构 |
| <c:forEach var="i" begin="1" end="10">  。。。。  </c:forEach> | for(i =0 ; i < 10;++i) |
| <c:forEach items="${listObj}" var="obj>  <img src="${pageurl }/resources/image/${obj.imageurl }" />  </c:forEach> | for(obj : listObj)  {  var = obj.imageurl  }  需要强转url为字符串时：  src="'+url+'" |

### 自定义标签

#### Java类

**package** com.test.domain;

**import** java.io.IOException;

**import** javax.servlet.jsp.JspWriter;

**import** javax.servlet.jsp.PageContext;

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

**public** **class** TeTag **extends** SimpleTagSupport {

**private** String strParam;

**private** **int** intParam;

**public** String getStrParam() {

**return** strParam;

}

**public** **void** setStrParam(String strParam) {

**this**.strParam = strParam;

}

**public** **int** getIntParam() {

**return** intParam;

}

**public** **void** setIntParam(**int** intParam) {

**this**.intParam = intParam;

}

@Override

**public** **void** doTag() **throws** IOException

{

PageContext pc = (PageContext)getJspContext();

JspWriter out = pc.getOut();

out.print(strParam + "=====" + intParam);

}

**public** **static** String funEL(String val){

**return** val + "===EL===";

}

}

#### tld文件

目录：/WEB-INFO/TeTag.tld

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

<taglib xmlns=*"http://java.sun.com/xml/ns/j2ee"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary\_2\_0.xsd"*

version=*"2.0"*>

<tlib-version>1.1</tlib-version>

<short-name>ssname</short-name>

<uri>\*\*\*\*\*\*\*</uri>

<tag>

<name>fun</name>

<tag-class>com.test.domain.TeTag</tag-class>

<body-content>empty</body-content>

<attribute>

<name>strParam</name>

<required>true</required>

<rtexprvalue>true</rtexprvalue>

</attribute>

<attribute>

<name>intParam</name>

<required>true</required>

<rtexprvalue>true</rtexprvalue>

</attribute>

</tag>

<function>

<name>funEL</name>

<function-class>com.test.domain.TeTag</function-class>

<function-signature>java.lang.String funEL(java.lang.String)</function-signature>

</function>

</taglib>

#### Jsp页面

<%@ taglib prefix=*"Te"* uri=*"*/WEB-INF/TeTag.tld*"* %>

<Te:fun strParam=*"aaaa"* intParam=*"15"*/>

<%@page isELIgnored=*"false"* %>

${Te:funEL('abcd')}

## Servlet

### 创建

实现 Servlet 接口或者继承 HttpServlet 方法

init方法和destroy方法为Servlet初始化与生命周期结束所调用。doGet和doPost方法用于http的get与post请求。

=========创建serverlet类

创建com.test.servletTe

**package** com.test;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**public** **class** servletTe **extends** HttpServlet {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** String message;

/\*\*

\* **@see** HttpServlet#HttpServlet()

\*/

**public** servletTe() {

**super**();

message = "servletTe";

}

/\*\*

\* **@see** HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)

\*/

**protected** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {

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

response.setCharacterEncoding("utf-8");

PrintWriter out = response.getWriter();

String str = "我是网页内容";

out.write(str);

}

/\*\*

\* **@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);

}

}

=========调用方式1

web.xml中加入以下配置：

<servlet>

<servlet-name>servletTe</servlet-name>

<servlet-class>com.test.servletTe</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>servletTe</servlet-name>

<url-pattern>/servletTe</url-pattern>

</servlet-mapping>

url访问/servletTe时调用serverlet

=========调用方式2

直接在类上加入注解

@WebServlet("/servletTe")

或者@WebServlet(name="servletTe",urlPatterns="/servletTe")

### 过滤器

Servlet过滤器实现Filter接口，可以配置对指定页面进行过滤，当请求指定过滤页面时先调用过滤器的doFilter

===========访问页面计数，每次请求页面时计数增加1==========

===========配置

<filter>

<filter-name>SF</filter-name>

<filter-class>com.WebTest.SF</filter-class>

<init-param>

<param-name>count</param-name>

<param-value>10</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>SF</filter-name>

<url-pattern>/index.jsp</url-pattern>

</filter-mapping>

在后面继续配置第二个<filter>同时定位到同一个<url-pattern>，可以做过滤链。

@WebFilter("/\*") 指定所有页面都使用

@WebFilter("/index.jsp") 指定index.jsp使用过滤

import javax.servlet.annotation.WebInitParam;

@WebFilter(urlPatterns = "/index.jsp",initParams= {@WebInitParam(name="aa",value="a")})

指定index.jsp使用过滤，设置初始参数aa

===========类定义

包含count成员变量

init函数中使用fConfig对象获取web.xml中<init-param>中配置的参数

String Count = fConfig.getInitParameter("count");

**if**(**null** == Count) **return**;

count = Integer.*valueOf*(count);

doFilter函数中进行计数，并写入到application对象

++count;

HttpServletRequest req = (HttpServletRequest)request;

ServletContext context = req.getSession().getServletContext();

context.setAttribute("count", count);

===========index.jsp中获取参数

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

### 监听器

实现监听接口HttpSessionBindingListener

public class SL implements HttpSessionBindingListener

java文件中注解@WebListener加入绑定

或者web.xml加入配置

<listener>

<listener-class>com.WebTest.SL</listener-class>

</listener>

注解中可以加入描述

@WebListener("description")

在jsp页面中调用session对象的setAttribute/removeAttribute会触发SL的valueBound/valueUnbound方法

session.setAttribute("name", **new** SL());

session.removeAttribute("name");

### 监听器上传文件

@MultipartConfig(location="D:/") //临时目录必须有访问权限

======上传界面

<form action=*"SL"* enctype=*"multipart/form-data"* method=*"post"*>

<br/>

<input type=*"file"* name=*"f1"*/>

<br />

<input type=*"submit"* name=*"upload"* value=*"upload"* />

</form>

====== *SL* 类的doPost

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

PrintWriter out = response.getWriter();

String path = **this**.getServletContext().getRealPath("/");

javax.servlet.http.Part p = request.getPart("f1");

**if**(p.getContentType().contains("image"))//只处理image文件

{

ApplicationPart ap = (ApplicationPart)p;

String PathFileName = ap.getSubmittedFileName();

**int** path\_idx = PathFileName.lastIndexOf("\\") + 1;

String fileName = PathFileName.substring(path\_idx,PathFileName.length());

p.write("D:/workspace/" + fileName); //存储目录必须存在

out.write("suc");

}

**else**

{

out.write("failed");

}

## spring

### 地址

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>4.0.3.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>4.0.3.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>4.0.3.RELEASE</version>

</dependency>

### 反向控制/依赖注入

反向控制IoC（Inversion of Control），指实现必须依赖抽象，而不是抽象依赖实现。

依赖注入DI（Dependency Injection），用来实现反向控制功能。包含三种实现方式：接口注入、Set注入、构造注入。spring实现Set注入和构造注入。

### javabean类的编写

定义一个接口（通过接口进行调用）

public interface AA {

void show();

}

编写实现类AAa

public class AAa implements AA{

public void show() {

System.out.println(msg);

System.out.println(val);

System.out.println(val2);

}

String val;

String val2;

public AAa(String co1,String co2)

{

val = co1;

val2 = co2;

}

String msg;

public String getMsg() {

return msg;

}

public void setMsg(String msg) {

this.msg = msg;

}

}

编写实现类AAb

public class AAb implements AA {

public void show() {

System.out.println(date);

}

Date date = null;

public Date getDate() {

return date;

}

public void setDate(Date date) {

this.date = date;

}

}

### xml编写

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

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd">

<!--

class包含类的全路径，通过property使用SetXX方法设置成员变量msg

通过constructor-arg进行构造注入

-->

<bean id="idName1" class="inf.AAa" >

<constructor-arg>

<value>arg1</value>

</constructor-arg>

<constructor-arg>

<value>arg2</value>

</constructor-arg>

<property name="msg">

<value>xxx</value>

</property>

</bean>

<!--

Set注入方式设置AAb类中的date对象

可以使用depends-on强制初始化依赖的类date

-->

<bean id="idName2" class="inf.AAb" depends-on="date">

<property name="date">

<ref bean="date"/>

</property>

</bean>

<bean id="date" class="java.util.Date"></bean>

</beans>

### 调用

import org.springframework.context.support.FileSystemXmlApplicationContext;

FileSystemXmlApplicationContext ctx = new FileSystemXmlApplicationContext("setting.xml");

xml放在项目根目录

AA a = (AA)ctx.getBean("idName1");

a.show();

AA b = (AA)ctx.getBean("idName2");

b.show();

## springMVC

### 框架搭建

#### 配置

##### web.xml

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

<web-app xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd "

version="2.5">

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<servlet>

<servlet-name>spring</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>spring</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

</web-app>

<mvc:resources mapping="/resources/\*\*" location="/WEB-INF/classes/static/"/>

可以将WEB-INF下的classes/static目录映射到根目录的resources目录

web.xml在WEB-INF下

定义了使用springframework进行监听和处理

默认使用当前目录下applicationContext.xml和spring-servlet.xml

通过以下代码可以重定义applicationContext.xml位置

（classpath表示java代码生成结果目录【WEB-INF\classes】）

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:applicationContext.xml</param-value>

</context-param>

通过以下代码可以重定义spring-servlet.xml路径

<servlet>

<servlet-name>spring</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:spring-servlet.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

##### applicationContext.xml

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

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:p="http://www.springframework.org/schema/p"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:aop="http://www.springframework.org/schema/aop"

xmlns:tx="http://www.springframework.org/schema/tx"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-4.0.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-4.0.xsd

http://www.springframework.org/schema/aop

http://www.springframework.org/schema/aop/spring-aop-4.0.xsd

http://www.springframework.org/schema/tx

http://www.springframework.org/schema/tx/spring-tx-4.0.xsd">

</beans>

##### spring-servlet.xml

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

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:p="http://www.springframework.org/schema/p"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:mvc="http://www.springframework.org/schema/mvc"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-4.0.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-4.0.xsd

http://www.springframework.org/schema/mvc

http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd">

<mvc:annotation-driven/>

<context:component-scan base-package="com.test.controller"/>

<mvc:default-servlet-handler/>

<bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="prefix" value="/WEB-INF/jsp/"/>

<property name="suffix" value=".jsp"/>

<property name="order" value="20"/>

</bean>

</beans>

在com.test.abc包目录下找注解类进行对应（多包扫描用逗号分隔com.abc,com.cde）

在/WEB-INF/jsp/下对应Controller控制跳转的jsp【Controller操作的根目录】

#### 使用

##### Controller

com.test.controller包下建立类

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.servlet.ModelAndView;

@Controller

public class mainController {

@RequestMapping(value="/home",method=RequestMethod.GET)

public ModelAndView home(){

ModelAndView mv = new ModelAndView();

mv.addObject("msg", "spring mvc");

return mv;

}

}

##### jsp页面

WEB-INF/jsp/目录下建立home.jsp

<%@ page language="java" contentType="text/html; charset=utf-8"

pageEncoding="utf-8"%>

<%@page isELIgnored="false" %>

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Insert title here</title>

</head>

<body>

${msg}

</body>

</html>

### Controller

#### 响应路径

@RequestMapping(value="/home",method=RequestMethod.GET)

public ModelAndView home(){

ModelAndView mv = new ModelAndView();

mv.addObject("msg", "spring mvc");

return mv;

}

函数无参数时，使用：jsp目录/home.jsp,

页面请求为 ..../项目名/home 时，则访问home.jsp

<jsp:forward page="/home"/>

通过框架跳转到home.jsp

@RequestMapping(value="/run",method=RequestMethod.GET)

public String fun(){

return "home/index"; //使用：jsp目录/home/index.jsp

}

地址组合:(页面请求为 ..../项目名/abc/def)

@Controller

@RequestMapping(value="/abc")

public class mainController {

@RequestMapping(value="/def",method=RequestMethod.GET)

public ModelAndView home(){

}

}

#### 参数

函数的参数个数和类型可以任意

spring可以根据参数类型自动将请求中包含的内容传入函数。

可以添加一次请求中包含的任意对象，使用了反射，保持和界面中参数名称一致即可。

|  |  |
| --- | --- |
| 参数 | 说明 |
| 可以包含HttpServletRequest request和HttpServletResponse response及HttpSession | <dependency>  <groupId>javax.servlet</groupId>  <artifactId>servlet-api</artifactId>  <version>2.5</version>  <scope>provided</scope>  </dependency>  ======获取get或post请求中包含的参数值  String title = request.getParameter("title");  ======设置变量值  HttpSession session = request.getSession(true);  session.setAttribute("username",loginUser);  界面获取：  <% String sessionUsername = (String)session.getAttribute("username"); %> |
| @RequestParam("valA") Integer val | 将jsp界面的参数count或者ajax请求中的json直接解析出对应的值val，  等价于int Val =  Integer.valueOf(request.getParameter("valA")); |
| @RequestMapping("/login/{username}")  public String login(@PathVariable("username") String username) | 从请求路径中获取参数 |
| @RequestBody String param | 获取请求对象整个字符串数据 |

#### 返回值

|  |  |
| --- | --- |
| 返回值类型 | 作用 |
| String | 表示返回jsp文件的名字  return "abc.jsp"; |
| ModelAndView | 将jsp文件组合数据进行返回 |
| 参数名-参数值传递  ModelAndView mv = new ModelAndView();  mv.addObject("msg", "spring mvc");  return mv; |
| 实体类传递：  类的成员变量必须是小写开头，变量有get和set方法  ModelAndView mv = new ModelAndView("/run/index");  DataStu dd = new DataStu();  dd.setId(1234);  mv.addObject(dd);  return mv;  jsp页面中使用[将类名前连续的大写全变为小写]  ${dataStu.id}  <form method="post">  <input type="text" name="id"/>  <input type="submit" />  </form>  代码获取[表单内容中包含和实体类相同的变量名]  public ModelAndView fun(DataStu data){  Integer d = data.getId();  } |
| List传递：  ModelAndView mv = new ModelAndView();  List<DataStu> datObj = UserService.fun();  mv.addObject("listDat", datObj );  return mv;  <%@taglib prefix="c"  uri="http://java.sun.com/jsp/jstl/core"%>  <c:forEach items="${listDat}" var="dat">  ${dat.id}  "${dat.id}"  </c:forEach> |
| void | 通过以下代码直接创建内容：  response.setContentType("text/html;charset=utf-8");  //response.setContentType("text/plain;charset=utf-8");  response.setCharacterEncoding("utf-8");  PrintWriter out = response.getWriter();  String str = "我是网页内容";  out.write(str); |
| @ResponseBody JavaBeanClass fun() | 返回结果不会被解析为跳转路径，而是直接写入 HTTP response body 中  将javaBean对象组装成json返回，等价于用response直接写入json对象。  需要依赖：  <dependency>  <groupId>com.fasterxml.jackson.core</groupId>  <artifactId>jackson-databind</artifactId>  <version>2.4.3</version>  </dependency>  可直接调用：  ObjectMapper objectMapper = new ObjectMapper();  String json = objectMapper.writeValueAsString( javaBeanObj ); |

### javabean

spring-servlet.xml中构造全局对象

构造函数：

<bean id="devSer" class="\*\*\* ">

<constructor-arg value="1"></constructor-arg>

<constructor-arg value="2"></constructor-arg>

</bean>

### WEB-INF

resources目录下内容会拷贝到WEB-INF\classes

WEB-INF目录下的文件无法直接通过URL访问

通过在web.xml中配置，可以映射目录下的文件到根目录下，使其可以访问

spring-servlet.xml中：

<mvc:resources mapping="/resources/\*\*" location="/WEB-INF/classes/"/>

/resources/icon/a.png映射为/WEB-INF/classes/icon/a.png

### service

applicationContext.xml中：

<context:component-scan base-package="com.abc.service"/>

扫描com.abc.service下所有的service类(@Service)，包装为javabean

## mybatis

### msql操作

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

public class dbOpt {

Connection conn;

Statement stmt;

ResultSet result;

public dbOpt(String DBName, String DBUser, String DBPass){

try {

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

String connMsg = "jdbc:mysql://localhost:3306/" + DBName + "?user=" + DBUser + "&password=" + DBPass;

conn = DriverManager.getConnection(connMsg);

}catch(Exception e){System.out.println(e.toString());}

}

public ResultSet Query(String SQLQuery){

try {

stmt = conn.createStatement();

result = stmt.executeQuery( SQLQuery );

}

catch( Exception e ){System.out.println(e.toString());}

return result;

}

public void exec(String SQLQuery){

try {

stmt = conn.createStatement();

stmt.execute( SQLQuery );

}

catch( Exception e ){System.out.println(e.toString());}

}

public void close(){

try {

stmt.close();

conn.close();

}

catch(Exception e){System.out.println(e.toString());}

}

}

===============使用

dbOpt db = new dbOpt("testdb","root","root");

ResultSet rs =db.Query("select \* from testTable");

while(rs.next()){

int id = rs.getInt("id");

System.out.println(id);

}

db.exec("insert into testTable(id) values(23)");

### 数据库表

数据库testdb中表user包含id和name

用户名和密码都是root

### 使用

#### 依赖

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

<version>3.4.6</version>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>8.0.11</version>

</dependency>

#### 实体类

创建包com.test.domain

**package** com.test.domain;

public class User {

private String name;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

}

#### mapper

创建包com.test.mapper

##### userMapper接口

package com.test.mapper;

import com.test.domain.User;

public interface userMapper {

public User fun(int id) throws Exception;

}

##### userMapper.xml

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

<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="com.test.mapper.userMapper">

<select id="fun" parameterType="int"

resultType="com.test.domain.User">

select name from user where id=#{id}

</select>

</mapper>

#### cfg.xml

将配置放在com.test.mapper

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

<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN" "http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>

<environments default="development">

<environment id="development">

<transactionManager type="JDBC" />

<dataSource type="POOLED">

<property name="driver" value="com.mysql.jdbc.Driver" />

<property name="url" value="jdbc:mysql://localhost:3306/testdb*?useUnicode=true&amp;characterEncoding=UTF-8&amp;serverTimezone=GMT%2B8*" />

<property name="username" value="root" />

<property name="password" value="root" />

</dataSource>

</environment>

</environments>

<mappers>

<package name="com/test/mapper"/>

</mappers>

</configuration>

告知映射文件方式如果使用：

<mappers>

<mapper resource="com/test/mapper/userMapper.xml"/>

</mappers>

则可以不创建userMapper的接口类

#### 调用

import java.io.IOException;

import java.io.Reader;

import org.apache.ibatis.io.Resources;

import org.apache.ibatis.session.SqlSession;

import org.apache.ibatis.session.SqlSessionFactory;

import org.apache.ibatis.session.SqlSessionFactoryBuilder;

import com.test.domain.User;

public class App{

public static void main(String[] args) throws IOException {

Reader reader = Resources.getResourceAsReader("com/test/mapper/cfg.xml");

SqlSessionFactory sessionFactory = new SqlSessionFactoryBuilder().build(reader);

SqlSession session = sessionFactory.openSession();

// String statement = "com.test.mapper.userMapper.fun";

// int id = 0;

// User user = session.selectOne(statement, id);

// session.commit();

// System.out.println(user.getName());

int idParam = 0;

userMapper mapper = session.getMapper(userMapper.class);

User userRet = mapper.fun(idParam);

System.out.println(userRet.getName());

}

}

#### MapperSql

使用<![CDATA[ ……]]> 可以在其中包含小于号

<select id="函数名" resultType="输出参数类型" parameterType="输入参数类型" >

<![CDATA[ 查询语句 ]]>

</select>

#{}将传入的参数当成一个字符串，会给传入的参数加一个双引号,能够很大程度上防止sql注入

${}将传入的参数直接显示生成在sql中，不会添加引号

<sql id="limit\_sql">

<where>

<if test="id != null and id != ''">

TempTableName.id=#{id}

</if>

</where>

</sql>

<select id="getlimit" parameterType="map" resultType="map">

select \* from TempTableName

<include refid="limit\_sql" />

ORDER BY id DESC LIMIT #{start}, #{limit}

</select>

#### parameterType

|  |  |
| --- | --- |
| 类型 | 用法 |
| 单个参数 | 可以不写parameterType(嵌套语句中引用变量)，也可以限定参数类型  如： parameterType="int" , parameterType="com.xxx.xxx"  如果是基本类型，直接引用变量：#{随便写}  如果是实体类，写成如#{id}，会自动在实体类中根据id字段拿出对应值  在嵌套语句中引用变量情况下，需要在函数声明时加@Param |
| 多个参数 | 不用写parameterType或者写parameterType="map"  需要在对应函数中为参数加@Param注解，以标识xml引用的变量名  如：  public User fun(@Param("name") String aa,@Param("age") int bb) throws Exception;  xml中：select name from user where name=#{name} AND age=#{age} |

#### resultType

|  |  |  |  |
| --- | --- | --- | --- |
| 类型 | resultType | 记录条数 | 说明 |
| 对象/List类型 | resultType="Integer"  resultType="com.\*\*" | 单条/多条 | 基本类型直接将结果赋值  实体类型时，将记录取出，并根据字段名匹配对象中的变量名进行赋值  多条数据时，返回值函数声明需要写成List<ClassName> |
| Map类型 | resultType="map" | 单条/多条 | 返回值是json  [{key1=val1, key2=value2}, {key1=val3, key2=val4}]  返回值可以用List<任意包装类型>接收 |
| Map类型 | Map中value的类型，如：  resultType="com.\*\*" | 多条 | 需要在Mapper的java文件中定义map中key的类型及对应的字段名  如以下代码定义使用age这个字段（数字类型）作为key：  @MapKey("age")  public Map<Integer,User> fun() throws Exception; |

### 结合springmvc

#### 依赖

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>4.0.3.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>4.0.3.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>4.0.3.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context-support</artifactId>

<version>4.0.3.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-tx</artifactId>

<version>4.0.3.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

<version>4.0.3.RELEASE</version>

</dependency>

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

<version>3.4.6</version>

</dependency>

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis-spring</artifactId>

<version>1.3.2</version>

</dependency>

<dependency>

<groupId>org.mybatis.caches</groupId>

<artifactId>mybatis-oscache</artifactId>

<version>1.0.2</version>

</dependency>

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>druid</artifactId>

<version>1.1.10</version>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>8.0.11</version>

</dependency>

<dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjweaver</artifactId>

<version>1.9.1</version>

</dependency>

#### applicationContext.xml

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

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:p="http://www.springframework.org/schema/p"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:aop="http://www.springframework.org/schema/aop"

xmlns:tx="http://www.springframework.org/schema/tx"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-4.0.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-4.0.xsd

http://www.springframework.org/schema/aop

http://www.springframework.org/schema/aop/spring-aop-4.0.xsd

http://www.springframework.org/schema/tx

http://www.springframework.org/schema/tx/spring-tx-4.0.xsd">

<tx:annotation-driven />

<bean id="dataSource"

class="com.alibaba.druid.pool.DruidDataSource" init-method="init"

destroy-method="close">

<property name="url"

value="jdbc:mysql://127.0.0.1:3306/testdb?useUnicode=true&amp;characterEncoding=UTF-8&amp;serverTimezone=GMT%2B8" />

<property name="username" value="root" />

<property name="password" value="root" />

<!-- 配置初始化大小、最小、最大 -->

<property name="initialSize" value="10" />

<property name="minIdle" value="5" />

<property name="maxActive" value="50" />

<!-- 配置获取连接等待超时的时间 -->

<property name="maxWait" value="60000" />

<!-- 配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒 -->

<property name="timeBetweenEvictionRunsMillis" value="60000" />

<!-- 配置一个连接在池中最小生存的时间，单位是毫秒 -->

<property name="minEvictableIdleTimeMillis" value="300000" />

<!-- 如果用Oracle，则把poolPreparedStatements配置为true，mysql可以配置为false。分库分表较多的数据库，建议配置为false。 -->

<property name="poolPreparedStatements" value="false" />

</bean>

<context:component-scan base-package="com.test">

<context:include-filter type="aspectj"

expression="com.test.service..\*" />

</context:component-scan>

<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">

<property name="basePackage" value="com.test.mapper" />

</bean>

<bean id="sqlSessionFactory"

class="org.mybatis.spring.SqlSessionFactoryBean">

<property name="dataSource" ref="dataSource" />

<property name="mapperLocations"

value="classpath:*com/test/mapper/\*.xml*" />

<property name="typeAliasesPackage"

value="com.test.domain" />

</bean>

<bean id="transactionManager"

class="org.springframework.jdbc.datasource.DataSourceTransactionManager"

p:dataSource-ref="dataSource" />

<bean id="jdbcTemplate"

class="org.springframework.jdbc.core.JdbcTemplate"

p:dataSource-ref="dataSource" />

</beans>

classpath指:WEB-INFO/classes

java类编译后会按照包的完整名称拷贝到这个目录

com.test.service..\*表示扫描com.test.service或com.test.serviceAb这类名字的包

#### 实体类

创建包com.test.domain

**package** com.test.domain;

public class User {

private String name;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

}

#### Mapper

创建包com.test.mapper

##### userMapper接口

package com.test.mapper;

import com.test.domain.User;

public interface userMapper {

public User fun(@Param("id") int id) throws Exception;

}

可以使用@Param标志从外部传来的参数为Mapper.xml使用

##### userMapper.xml

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

<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="com.test.mapper.userMapper">

<select id="fun" parameterType="int"

resultType="com.test.domain.User">

select name from user where id=#{id}

</select>

</mapper>

#### Service

创建包 com.test.service

##### userService接口

package com.test.service;

import com.test.domain.User;

public interface userService {

public User fun(int id);

}

##### userServicempl

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import com.test.domain.User;

import com.test.mapper.userMapper;

import com.test.service.userService;

@Service

public class userServicempl implements userService{

@Autowired

private userMapper UserMapper;

public User fun(int id) {

User user = null;

try {

user = UserMapper.fun(id);

}catch(Exception e)

{

}

return user;

}

}

#### Controller调用

@Autowired

private userService UserService;

String str = UserService.fun(0).getName();

#### 日志输出

##### 依赖

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.17</version>

</dependency>

##### log4j.properties

classpath目录下

log4j.rootLogger=off,Console,File

log4j.appender.Console=org.apache.log4j.ConsoleAppender

log4j.appender.Console.Target=System.out

log4j.appender.Console.layout = org.apache.log4j.PatternLayout

log4j.appender.Console.layout.ConversionPattern = [%p] [%d{yyyy-MM-dd HH\:mm\:ss}][%c]%m %l%n

log4j.appender.File = org.apache.log4j.RollingFileAppender

log4j.appender.File.File = C://logs/ssm.log

log4j.appender.File.MaxFileSize = 10MB

log4j.appender.File.Threshold = ALL

log4j.appender.File.layout = org.apache.log4j.PatternLayout

log4j.appender.File.layout.ConversionPattern = [%p] [%d{yyyy-MM-dd HH\:mm\:ss}][%c]%m %l%n

##### mybatis-config.xml

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

<!DOCTYPE configuration

PUBLIC "-//mybatis.org//DTD Config 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>

<settings>

<setting name="logImpl" value="STDOUT\_LOGGING" />

</settings>

</configuration>

##### applicationContext加入配置

<bean id=*"sqlSessionFactory"*  加入属性

<property name="configLocation" value="WEB-INF/mybatis-config.xml"></property>

## thymeleaf

public String index(ModelMap map) {

map.addAttribute("abc", "吃饭睡觉");

return "index";

}

<h1 th:text=*"${abc}"*>\*\*\*\*\*</h1>

<script th:inline=*"javascript"*>

/\*<![CDATA[\*/

**var** type = /\*[[${abc}]]\*/;

alert(type);

/\*]]>\*/

</script>