**Web.xml配置**

配置DispatcherServlet 这个类接管所有请求。

<servlet>

<servlet-name>mvc</servlet-name>

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

<!-- 指定spring mvc配置文件的名称和位置 -->

<init-param>

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

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

</init-param>

</servlet>

**controller的配置**

@Controller

**public** **class** HelloController {

//通过使用@RequestMapping 注解 可以指定请求的url映射到该方法上

@RequestMapping(value="/hello")

**public** ModelAndView hello(String name,int age){

//。。。。。。。。

ModelAndView mv = **new** ModelAndView();

mv.setViewName("hello");//对应hello.jsp页面

mv.addObject("msg", "annotation ------");//返回信息

**return** mv;

}

}

**mvc.xml配置**

**<bean class=*"org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter"*/>**

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

<property name=*"viewClass"* value=*"org.springframework.web.servlet.view.JstlView"*/>

<!-- 将视图名 渲染后视图的前缀 -->

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

<!-- 渲染后视图的后缀 -->

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

<!-- 例：视图名为:hello 渲染后：/WEB-INF/jsp/hello.jsp 该页面-->

</bean>

<!-- spring容器扫描指定包下的所有类，如果类上有注解 那么根据注解产生相应bean对象已经映射信息 -->

<context:component-scan base-package="cn.tx.controller"/>

</beans>

**前端参数传递**

在springmvc中 获取页面数据不需要象servlet那样，通过req.getParameter来获取，在处理方法上，直接将参数名设置为表单中的域名，这样表单中的该域值会被直接注入到处理方法的参数上。注意类型一致

<form action=*"hello"* method=*"post"*>

username:<input type=*"text"* name=*"****name****"*/><br>

age:<input type=*"text"* name=*"****age****"*/><br>

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

</form>

@RequestMapping("/hello")

**public** ModelAndView add(String **name**,**int** **age**){

System.*out*.println("name="+name+" age="+age);

ModelAndView mv = **new** ModelAndView("success");

mv.addObject("msg", "添加成功");

**return** mv;

}

后端Controller决定跳转方式

1，通过ModelAndView 对象可以控制页面跳转，默认转发

2，通过ServletApI进行跳转 返回值为void，可以转发也可以重定向

@RequestMapping(value="/hello")

**public** **void** hello(HttpServletRequest req,HttpServletResponse resp) **throws** ServletException, IOException{

//req.getRequestDispatcher("/index.jsp").forward(req, resp);

resp.sendRedirect("index.jsp");

}

3,返回值为String类型，我们可以通过前缀指定为转发还是重定向

@RequestMapping("/hello")

**public** String hello3(ModelMap map){

map.addAttribute("msg", "helloworld");

**return** "redirect:index.jsp";

}

**使用springmvc来完成json数据的展示**

在处理方法上直接返回java对象即可，由spring mvc来进行转换，需要注解 @ResponseBody

@RequestMapping("/json.do")

@ResponseBody

**public** List<User> json(){

List<User> list = **new** ArrayList<User>();

list.add(**new** User(1,"zhansan",22));

list.add(**new** User(2,"wangwu",21));

list.add(**new** User(3,"zhaosi",33));

list.add(**new** User(4,"wangdana",14));

**return** list;

}

需mvc.xml添加配置

<!-- json配置 -->

<!-- 用于将对象转换为 JSON -->

<bean id=*"stringConverter"* class=*"org.springframework.http.converter.StringHttpMessageConverter"*>

<property name=*"supportedMediaTypes"*>

<list>

<value>text/plain;charset=UTF-8</value>

</list>

</property>

</bean>

<bean id=*"jsonConverter"* class=*"org.springframework.http.converter.json.MappingJackson2HttpMessageConverter"*></bean>

<bean class=*"org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter"*>

<property name=*"messageConverters"*>

<list>

<ref bean=*"stringConverter"* />

<ref bean=*"jsonConverter"* />

</list>

</property>

</bean>

<!-- spring容器扫描指定包下的所有类，如果类上有注解 那么根据注解产生相应bean对象已经映射信息 -->

<context:component-scan base-package=*"cn.tx.controller"*/>

或配置

<mvc:annotation-driven/>

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

**Restful调用接口**

//使用Restful方式获取参数

@RequestMapping("/{id}/hello")

**public** **void** helloRestful(@PathVariable("id") **int** id){

System.*out*.println("id=="+id);

}

传统方式的请求：[http://localhost:8080/testmvc/hello?id=3](http://localhost:8080/restful/hi.do?id=3)

Restful方式的请求：[http://localhost:8080/testmvc/3/hello](http://localhost:8080/restful/3/hello.do)

**springmvc中使用session**

@RequestMapping("/session")

**public** String login(HttpSession session,String name){

session.setAttribute("user", name);

**return** "forward:/index.jsp";

}

**拦截器：主要是拦截请求，对请求进行过滤。**

在springmvc中拦截器的实现有两种方式：第一实现HandlerInterceptor接口，第二继承HandlerInterceptorAdapter类

**public** **class** LoginInterceptor **implements** HandlerInterceptor{

/\*\*

\* 在Controller的处理方法之前被执行

\* 执行一些初始化，权限判断，日志等处理

\* 如果返回false 将不会执行Controller中的处理方法

\* \*/

@Override

**public** **boolean** preHandle(HttpServletRequest req, HttpServletResponse resp,

Object object) **throws** Exception {

//登录拦截器

/\*1判断session中是否有值，有值-----放行

\*2session中没有值 ，判断请求是否是toLogin和login如果是----放行

\*3、跳转到toLogin

\* \*/

HttpSession session = req.getSession();

**if**(session.getAttribute("user")!=**null**)

**return** **true**;

String path=req.getRequestURI();

**if**(path.endsWith("toLogin")||path.endsWith("login"))

**return** **true**;

resp.sendRedirect("toLogin");

**return** **false**;

}

/\*\*

\* 在Controller处理方法执行之后，在被渲染之前 执行

\* 关闭，释放，处理一些Controller执行中需要的资源

\* \*/

@Override

**public** **void** postHandle(HttpServletRequest arg0, HttpServletResponse arg1,

Object arg2, ModelAndView arg3) **throws** Exception {

// **TODO** Auto-generated method stub

}

/\*\*

\* 在渲染器渲染后被执行

\* 处理一个请求后的一些清理

\* \*/

@Override

**public** **void** afterCompletion(HttpServletRequest arg0,

HttpServletResponse arg1, Object arg2, Exception arg3)

**throws** Exception {

}

}

拦截器配置

<mvc:interceptors>

<!-- 在 mvc:interceptors中可以配置多个拦截器

多个拦截器按顺序一一执行

-->

<mvc:interceptor>

<!-- 指明哪些请求需要被拦截

/login表示login需要被拦截

/user/add 表示/user/add请求要被拦截

/\* 表示项目路径下的所有请求被拦截，可以拦截： /login /add /user 等

/\*\*表示项目路径及其子路径下的所有请求被拦截=所有请求被拦截

可以拦截： /login /add /user /user/add -->

<mvc:mapping path=*"/\*\*"*/>

<!-- 指定拦截器的位置 -->

<bean class=*"cn.tx.interceptor.LoginInterceptor"*></bean>

</mvc:interceptor>

</mvc:interceptors>

mybatis是一个轻量级的，半自动化的orm（对象关系映射）框架

mybatis核心配置文件

<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/test"* />

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

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

</dataSource>

</environment>

</environments>

<mappers>

<mapper resource=*"cn/tx/mapper/UserMapper.xml"* />

</mappers>

</configuration>

映射文件

<mapper namespace=*"cn.tx.mapper.UserMapper"*>

<!-- 查询单个对象 -->

<select id=*"selectUser"* parameterType=*"int"* resultType=*"cn.tx.vo.User"*>

select \* from t\_user where id = #{id}

</select>

<!-- 查询所有数据 -->

<select id=*"list"* resultType=*"cn.tx.vo.User"*>

select \* from t\_user

</select>

<!-- 添加对象

useGeneratedKeys=true 表示主键使用自增

-->

<insert id=*"addUser"* parameterType=*"cn.tx.vo.User"* useGeneratedKeys=*"true"*>

insert into t\_user(name,age) values(#{name},#{age})

</insert>

<!-- 删除指定对象 -->

<delete id=*"deleteUser"* parameterType=*"int"*>

delete from t\_user where id=#{id}

</delete>

<!-- 修改对象

当参数是对象时，sql语句中的参数名称和对象属性名一致

-->

<update id=*"update"* parameterType=*"cn.tx.vo.User"*>

update t\_user set name=#{name},age=#{age} where id=#{id}

</update>

</mapper>

不能使用>和<,因为和配置文件的括号一样，可以使用<![CDATA[

select \* from t\_user where id < #{id}

]]>

来操作

**测试代码**

**public** **class** Test {

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

Reader reader = Resources.*getResourceAsReader*("mybatis.cfg.xml");

SqlSessionFactory sqlMapper = **new** SqlSessionFactoryBuilder().build(reader);

SqlSession session = sqlMapper.openSession();

**try** {

User user = (User) session.selectOne(

"cn.tx.mapper.UserMapper.selectUser", 1);

//session.update("cn.tx.mapper.UserMapper.update", user)

//session.insert("cn.tx.mapper.UserMapper.addUser", user)

//session.delete("cn.tx.mapper.UserMapper.deleteUser", 5);

// List<User> list=session.selectList("cn.tx.mapper.UserMapper.list");

**if**(result>0){

System.*out*.println(user);

} **finally** {

session.close();

}

}

}

1. 如果参数是对象时，需要注意sql语句中的参数和对象的属性名一致
2. 在插入对象时，是否使用主键自增。如果使用自增需要添加useGeneratedKeys="true"。
3. 注意事务的提交。可以设置为自动提交
4. 注意关闭Session

*当对象与表字段不一致时，通过resultMap来进行映射*

<select id=*"selectUser"* parameterType=*"int"* resultMap=*"UserMap"*>

<![CDATA[

select \* from t\_user where uid = #{id}

]]>

</select>

<!-- 对结果集进行映射

id 是结果集映射的标识符

type指明最终要转换为java类型

-->

<resultMap type=*"User"* id=*"UserMap"*>

<!-- id表示主键映射

column指表字段 property 指java对象的属性-->

<id column=*"uid"* property=*"id"*/>

<!-- result表示其他字段映射 -->

<result column=*"uname"* property=*"name"*/>

<result column=*"uage"* property=*"age"*/>

</resultMap>

多对一关联

<mapper namespace=*"cn.tx.mapper.UserMapper"*>

<select id=*"selectUser"* parameterType=*"int"* resultMap=*"UserMap"*>

<![CDATA[

select \* from t\_user where id = #{id}

]]>

</select>

<resultMap type=*"User"* id=*"UserMap"*>

<!--

association 关联一个对象

property 指java对象的属性

column 对应数据库中表的字段

javaType 指java类型

select 表示该对象需要查询后获得

-->

<association property=*"role"* column=*"roleId"* javaType=*"Role"*

select=*"cn.tx.mapper.RoleMapper.selectRole"*

></association>

</resultMap>

</mapper>

**一对多映射**

UserMapper.xml

<mapper namespace=*"cn.tx.mapper.UserMapper"*>

<select id=*"listByRoleId"* parameterType=*"int"* resultType=*"User"*>

select \* from t\_user where roleId=#{roleId}

</select>

</mapper>

RoleMapper.xml

<mapper namespace=*"com.tx.mapper.RoleMapper"*>

<select id=*"selectRole"* parameterType=*"int"* resultMap=*"RoleMap"*>

<![CDATA[

select \* from t\_role where id = #{id}

]]>

</select>

<resultMap type=*"Role"* id=*"RoleMap"*>

<!-- collection表示集合 也就是一对多 -->

<collection property=*"users"* column=*"id"* javaType=*"ArrayList"*

select=*"com.tx.mapper.UserMapper.listByRoleId"*

></collection>

</resultMap>

</mapper>

动态sql查询

<select id=*"list"* parameterType=*"map"* resultType=*"User"*>

select \* from t\_user

<!-- 如果有条件成立 则加上where 如果没有条件成立则不加where -->

<where>

<!-- test中的属性 会去找对应属性的get方法来获取值 -->

<if test=*"id>2"*>

id=#{id}

</if>

<!-- 模糊查询 -->

<if test=*"name!=null"*>

and name like concat(#{name},'%')

</if>

</where>

</select>

<select id=*"list"* parameterType=*"map"* resultType=*"User"*>

select \* from t\_user

<!-- 如果有条件成立 则加上where 如果没有条件成立则不加where -->

<where>

<!-- choose选择其中一个做为查询条件,因为<if>...</if>并没对应的<else>标签，所以要达到<if>...<else>...</else> </if>的效果 -->

<choose>

<when test=*"age>23"*>

age=#{age}

</when>

<otherwise>

and name like concat(#{name},'%')

</otherwise>

</choose>

</where>

</select>

<select id=*"list1"* parameterType=*"list"* resultType=*"User"*>

select \* from t\_user where id in

<!-- collection 参数的类型 一般为list或array item是每次迭代的元素变量

open以什么开始 close以什么结束 separator 每个元素间的分隔符-->

<foreach collection=*"list"* item=*"bean"* open=*"("* close=*")"* separator=*","*>

#{bean}

</foreach>

</select>