人类是一种

不知道自己要干什么的时候，在心底默默问自己是什么、为什么、怎么办总归不会错的。

SSM是什么？Spring、SpringMVC、Mybatis的统称。

为什么要用SSM？为了方便大型web项目的开发。

怎么用呢？

既然SSM是为web服务的，那说到底都是对Servlet的封装，所以这个问题就变成了SSM是如何代替Servlet的？Mybatis替代了JDBC，SpringMVC替代了Servlet，然后通过Spring，它们运行在服务器里。所以你看，奋斗在第一线的框架就被筛选出来了，Mybatis与SpringMVC。那么第一步，只靠Mybatis与SpringMVC写一个简单的web项目，仅涉及一个模块的增删查改就好。

神圣的 alt + / 永远守护着我

***1. 先跑起来***

创建一个空的dynamic web project。

* Mybatis

这一块我是真忘了，但不要立马百度教程或看笔记，先用 alt+/ 试着回忆。

mybatis真的很“轻”，准备一个jar包就能立马飞起来。然后建好相应的model类以及Mapper（DAO），最后准备mybatis-config.xml。

为了避免因基础类型与包装类转换过多而造成的效率低下，后台统一使用包装类。

Mapper方面，象征性的写4个方法。至于xml，这个肯定要去百度一下，那个DTD谁会记得啊。

StudentMapper.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.jojo.mapper.StudentMapper"*>

<resultMap id=*"BaseResultMap"* type=*"com.jojo.model.Student"*>

<id column=*"id"* property=*"id"* jdbcType=*"INTEGER"*/>

<result column=*"name"* property=*"name"* jdbcType=*"VARCHAR"*/>

<result column=*"age"* property=*"age"* jdbcType=*"INTEGER"*/>

</resultMap>

<insert id=*"add"* parameterType=*"com.jojo.model.Student"*>

INSERT INTO t\_student VALUES(null, #{name}, #{age})

</insert>

<delete id=*"delete"* parameterType=*"Integer"*>

DELETE FROM t\_student WHERE id = #{id}

</delete>

<update id=*"update"* parameterType=*"com.jojo.model.Student"*>

UPDATE t\_student SET name = #{name}, age = #{age} WHERE id = #{id}

</update>

<select id=*"selectOneById"* parameterType=*"Integer"* resultMap=*"BaseResultMap"*>

SELECT \* FROM t\_student WHERE id = #{id}

</select>

</mapper>

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>

<!-- 引入存放数据库连接信息的properties，参数一定要放在properties里，因为修改xml是要重新编译的。而且放properties里也方便统一管理 -->

<properties resource=*"com/jojo/resource/jdbc.properties"*></properties>

<typeAliases>

<package name=*"com.jojo.model"*/>

</typeAliases>

<environments default=*"develop"*>

<environment id=*"develop"*>

<!-- 事务 -->

<transactionManager type=*"JDBC"*></transactionManager>

<!-- 数据库连接池，mybatis不能像Spring那样随意更换数据库连接池，非要换也行，比较烦 -->

<dataSource type=*"POOLED"*>

<property name=*"driver"* value=*"${jdbc.driverClassName}"* />

<property name=*"username"* value=*"${jdbc.username}"* />

<property name=*"url"* value=*"${jdbc.url}"* />

<property name=*"password"* value=*"${jdbc.password}"* />

</dataSource>

</environment>

</environments>

<mappers>

<package name=*"com.jojo.mapper"* />

</mappers>

</configuration>

以上便是跑mybatis之前的准备工作，已经缩减至最少，再少就要出问题了。

* SpringMVC

jar包方面我比较简单粗暴，springframework全贴上去。

跑Servlet很简单，写个类继承HttpServlet然后在web.xml里配置<servlet>标签。SpringMVC的话只是在这中间多了自己的配置文件，不要百度，靠文档把SpringMVC跑起来。

springMvc-config.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:context=*"http://www.springframework.org/schema/context"*

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

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

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

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

<!-- 注解扫描器 -->

<context:component-scan base-package=*"com.jojo"*></context:component-scan>

<!-- 视图解析器 -->

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

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

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

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

</bean>

</beans>

上面那两个是必须，没了就不叫SpringMvc了。

web.xml

<servlet>

<servlet-name>dispatcherServlet</servlet-name>

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

<init-param>

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

<param-value>classpath:com/jojo/resource/springMvc-config.xml</param-value>

</init-param>

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

</servlet>

<servlet-mapping>

<servlet-name>dispatcherServlet</servlet-name>

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

</servlet-mapping>

单纯把SpringMVC跑起来的话，配置文件写成上面那样也就结束了，但展现出来的功能很差，比如乱码问题就没解决。

* 整合完毕

可能你看了想打人，但从功能角度讲，我确实整合完毕了。Mybatis与SpringMVC都能正确运行，那么，写个mybatisUtil类，再写个StudentService，最后在Controller里调，完美不？

你问applicationContext.xml，那东西目前还没出现。。。。

顶多就是烦，很烦。所以才需要下面这个东西。

***2. 整合***

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis-spring</artifactId>

<version>1.2.5</version>

</dependency>

这个jar包将不可避免的出现在我的眼前，因为只有靠这个才能做到无缝整合，也就是说mybatis-config.xml没了，只留applicationContext.xml。

无缝整合至少要在spring的xml里配置两样东西，SqlSessionFactory和数据映射器类。

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

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

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

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

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

<!-- 配置数据源 -->

<context:property-placeholder location=*"classpath:com/jojo/resource/jdbc.properties"*/>

<bean id=*"dataSource"* class=*"org.springframework.jdbc.datasource.DriverManagerDataSource"*>

<property name=*"driverClassName"* value=*"${jdbc.driverClassName}"* />

<property name=*"url"* value=*"${jdbc.url}"* />

<property name=*"username"* value=*"${jdbc.username}"* />

<property name=*"password"* value=*"${jdbc.password}"* />

</bean>

<!-- 数据库连接 -->

<bean id=*"sqlSessionFactory"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*>

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

</bean>

<bean id=*"StudentMapper"* class=*"org.mybatis.spring.mapper.MapperFactoryBean"*>

<property name=*"mapperInterface"* value=*"com.jojo.mapper.StudentMapper"* />

<property name=*"sqlSessionFactory"* ref=*"sqlSessionFactory"* />

</bean>

</beans>

测试代码：

String resource = "classpath:com/jojo/resource/applicationContext.xml";

ApplicationContext applicationContext = **new** ClassPathXmlApplicationContext(resource);

StudentMapper studentMapper = (StudentMapper) applicationContext.getBean("StudentMapper");

Student student = **new** Student();

student.setName("jo");

student.setAge(3);

student.setId(2);

studentMapper.update(student);

经测试，表内的数据被修改。

// 简直完美啊，前提是不看性能。。。

使用spring提供的数据源进行连接，当然也可以换成别得，比如dbcp，c3p0。配置完SQLSessionFactory后直接获取StudentMapper，有意思的是连mapper.xml都没有配置竟然也成功运行了，估计是直接去mapper所在的接口处寻找，找不到肯定就要报错了。

*org.mybatis.spring.SqlSessionFactoryBean*这个类有很多值得挖掘的地方，我列举两个常用的

<property name=*"mapperLocations"* value=*"classpath\*:sample/config/mappers/\*\*/\*.xml"* />

<property name=*"configLocation"* value=*"classpath\*:sample/config/mappers/\*\*/\*.xml"* />

分别用来指定mapper.xml与mybatis-config.xml，注意“s”。还要留着mybatis-config.xml是因为可能要配置别名与基础设置（缓存）

事务，连事务都没有，这个例子裸奔的还真是彻底啊

用mybatis-spring的好处之一就是事务，它爽到你原来怎么用spring事务的，你现在还怎么用，不用费心思去配什么乱七八糟的玩意。

<!-- 事务管理 -->

<!-- 定义事务管理器 -->

<bean id=*"transactionManager"*

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

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

</bean>

<!-- 事务通知 -->

<tx:advice id=*"txAdvice"* transaction-manager=*"transactionManager"*>

<tx:attributes>

<tx:method name=*"\*"* propagation=*"REQUIRED"*/>

</tx:attributes>

</tx:advice>

<!-- 将通知切到何处 -->

<aop:config>

<aop:advisor advice-ref=*"txAdvice"* pointcut=*"execution(\* com.jojo.service.\*.\*(..))"* />

</aop:config>

这事务开的轻松而又惬意，但是别忘了在<beans>标签里新增aop与tx的xmlns，而且jar包别忘了贴。

（虽说mybatis管理事务是最轻松的。。。。）

springMvc-config.xml照旧，然后把上面的测试代码放到Controller里，完美运行。但这么做很烦，就算你写一个SpringContextUtil依旧很烦（但这个类你是一定要写的）

***3. 使用注解***

每获取一个Mapper，或一个Service，都要在applicationContext.xml中加上相关的<bean>。这是另一种折磨，违背了 DLGM 原则（“do less get more”），顺带一提还有个原则叫KISS（“keep it simple and stupid”）。而且频繁的写<bean>，很容易写错，而且倒时候一个applicationContext.xml文件超大，不好管理。

直接用这么一个注解是坠吼的。

@Autowired

**private** StudentService studentService;

或

@Resource(name = "studentServiceImpl")

**private** StudentService studentService;

原则上建议都用@Resource，但人要适当变（tou）通（lan）。

基础一般的话，看上面那两个注解应该还是挺费劲的，尤其是@Resource搞不好就漏看了，那个“s”是小写。

从xml开始，慢慢梳理，使用注解究竟需要改动哪些玩意。

* springMvc-config.xml，依旧没变化
* web.xml

新增监听器及初始参数

<listener>

<description>Spring容器加载监听器</description>

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

</listener>

<context-param>

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

<param-value>classpath:com/jojo/resource/applicationContext.xml</param-value>

</context-param>

* applicationContext.xml

这里是重头，首先修改sqlSessionFactory<bean>，将所有mapper.xml交给Spring管理。然后将原来的StudentMapper<bean>替换，改成扫描所有mapper接口的<bean>，原理不懂，我也蛮好奇的，这玩意怎么被执行的。

（有个地方不懂，在第2节里，我明明没有指定mapper.xml，Spring获取的StudentMapper是怎么正确增删查改的）

<!-- 数据库连接 -->

<bean id=*"sqlSessionFactory"* class=*"org.mybatis.spring.SqlSessionFactoryBean"*>

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

<!-- 自动扫描mapper.xml文件 -->

<property name=*"mapperLocations"* value=*"classpath:com/jojo/mapper/\*.xml"*></property>

</bean>

<!-- 自动扫描Mapper类 -->

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

<property name=*"sqlSessionFactoryBeanName"* value=*"sqlSessionFactory"*></property>

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

</bean>

* @Autowired & @Resource

处理完配置文件后就可以愉快的使用注解了。

@Autowired这个注解的意思是自动装配，一旦用该注解修饰某类型引用，

@Autowired

**private** StudentMapper studentMapper;

Spring会自动从容器中寻找符合该类型的实例，并赋值。非常方便，但随之而来的问题也很头疼，如果Spring容器里有多个同类型的实例，该怎么办？比如StudentMapper有好几个，程序是很傻逼的，遇到问题从不处理，抛个异常就shutdown。

要想不报错也很简单@Autowired(required = **false**)，遇到bug不注入就是了。但这不能解决问题，然后你会发现@Resource是这么的可爱。与@Autowired不同，@Resource这个注解需要指定要注入的<bean>。

@Resource(name = "studentMapper")

**private** StudentMapper studentMapper;

通过名字指定，因为Spring容器里的每一个实例的id属性都是不重复的，可以没有id但坚决不能重复。呐，为什么上面那个StudentMapper是小写的 s 呢。因为，StudentMapper是自动扫描的，这种<bean>，Spring会直接拿它的类名做id但首字母小写。

不过指定<bean>的方式不止这一种，用id的话方便些。

衍生： @Service、@Repository、@Component

这么用

@Service

class XXX{}

这样可以把这个类交给Spring容器管理，省的去applicationContext.xml里配置一堆东西。同样，id属性的值为小写的类名。默认的不一定好，总有那么几种情况你得自己指定id，@Service("StudentService")

@Repository、@Component与@Service的作用一样，只是代表的含义不同。@Repository用于修饰DAO及Mapper（DAO与Mapper是一个意思）。@Component用于修饰一些意义不明的类。

**SpringMVC** ####################################################

注解扫描器

视图解析器

保护被引用的资源

监听器

过滤器

Servlet

**Spring** ########################################################

连接数据库

事务

自动扫描mapper