SSM框架知识点

spring

优点

方便解耦，简化开发

Spring就是一个大工厂，专门负责生成Bean，可以将所有对象创建和依赖关系维护由Spring管

AOP编程的支持

Spring提供面向切面编程，可以方便的实现对程序进行权限拦截、运行监控等功能

声明式事务的支持:

只需要通过配置就可以完成对事务的管理，而无需手动编程

方便程序的测试:

Spring对Junit4支持，可以通过注解方便的测试Spring程序

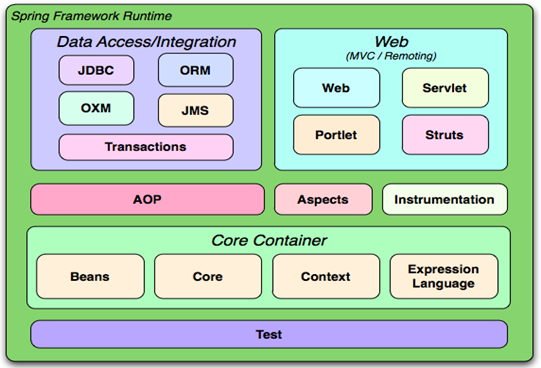
方便集成各种优秀框架:

Spring不排斥各种优秀的开源框架，其内部提供了对各种优秀框架（如：Struts、Hibernate、MyBatis、Quartz等）的支持

降低JavaEE API的使用难度 Spring:

对JavaEE开发中一些难用的API（JDBC、JavaMail、远程调webservice用等），都提供了封装，使这些API应用难度大大降低

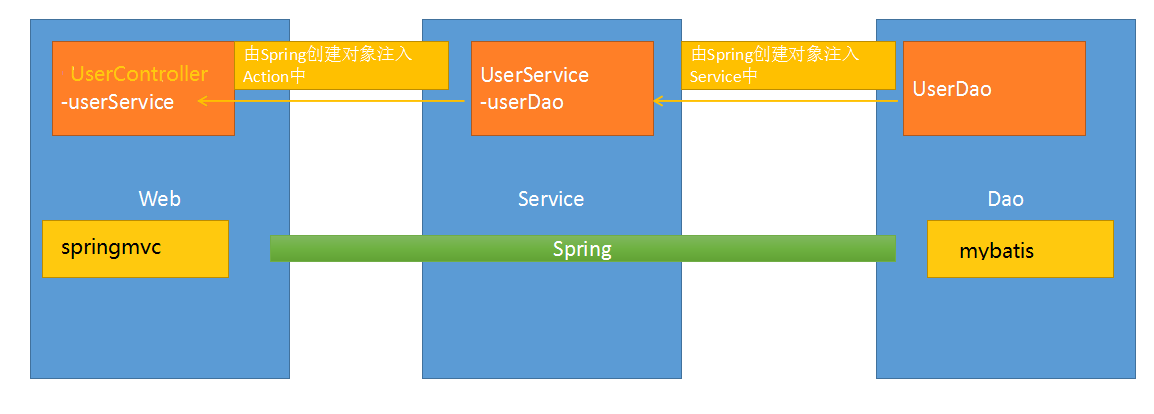
框架结构



在项目中的架构

web层：Struts,SpringMVC

dao层：Hibernate,mybatis



IoC(控制反转)

配置

导包(pom.xml)

<!--spring的jar包依赖-->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>4.3.9.RELEASE</version>

</dependency>

spring.xml配置

<?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">

</beans>

空构造器方式实例化

1.实例化的Bean中需要空参构造器

2.<bean id="" class=""></bean>

静态工厂方式实例化

1.需要一个生产Bean的工厂，静态的方法 有返回值Bean

2.<bean id="需要创建bean的id" class="工厂类的全限定名" factory-method="静态方法名"></bean>

实例化工厂方式实例化

1.需要一个生产Bean的工厂，非静态方法 有返回值Bean

2.<bean id="工厂id" class="工厂类的全限定名"></bean>

3.<bean id="需要创建bean的id" factory-bean="工厂id" factory-method="非静态方法名"></bean>

DI(依赖注入)

Set注入

bean对象中要有set方法

<bean id="需要注入的属性id" class=""></bean>

<bean id="" class="">

<property name="需要注入的属性名称" ref="需要注入的属性id" value=""></property>

</bean>

构造器注入

bean对象中要有带参构造器

<bean id="需要注入的属性id" class=""></bean>

<bean id="" class="">

<constructor-arg name="需要注入的属性名称" ref="需要注入的属性id" value=""> </constructor-arg >

</bean>

静态工厂方式注入

IoC静态工厂和DI Set方式的组合

bean对象中要有set方法

需要一个生产Bean的工厂，静态的方法 有返回值Bean

<!-- 静态工厂注入 -->

<bean id="userDao01" class="com.mage.demo.StaticFactory" factorymethod="createuserDao"></bean>

<bean id="userService01" class="com.mage.demo.UserService">

<property name="userDao" ref="userDao01"></property>

</bean>

实例化工厂方式注入

IoC实例化工厂和DI Set方式的组合

需要一个生产Bean的工厂，非静态方法 有返回值Bean

bean对象中要有set方法

<!-- 实例化工厂 -->

<bean id="instanceFactory" class="com.mage.demo.InstanceFactory"></bean>

<bean id="userDao3" factory-bean="instanceFactory" factory-method="createUserDao">

</bean>

<bean id="userService02" class="com.mage.demo.UserService">

<property name="userDao" ref="userDao3"></property>

</bean>

注解方式注入Bean

对于bean的注入常用注解类型

@Autowired属性字段或set方法上

@Resource属性字段或set方法上

区别：@Autowired默认按bean的类型匹配可以修改按名称匹配和@Qualifier配合使用@Resource默认按名称进

行装配，名称可以通过name属性进行指定，如果没有指定name属性，当注解写在字段上时，默认取字段名进行匹配注入，如果注解写在setter方法上默认取属性名进行装配。当找不到与名称匹配的bean时才按照类型进行装配。但是需要注意的是，如果name属性一旦指定，就只会按照名称进行装配。

推荐使用@Resource注解是属于J2EE的，减少了与spring的耦合。

Spring IoC容器自动扫描管理bean

Dao层：@Repository

Service层：@Service

视图控制层：@Controller

如果对于开发的类实在不明确到底属于哪个层，可以使用@Component注解定义。

Bean的作用域问题

类别 说明

singleton 在Spring IoC容器中仅存在一个Bean实例，Bean以单例方式存在，默认值

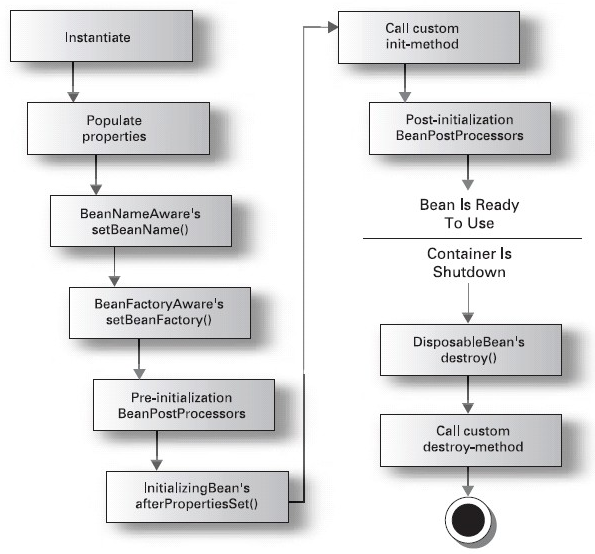
prototype 每次从容器中调用Bean时，都返回一个新的实例，即 每次调用getBean()时 ，相当于执行new XxxBean()

request 每次HTTP请求都会创建一个新的Bean，该作用域仅 适用于WebApplicationContext环境

session 同一个HTTP Session 共享一个Bean，不同Session 使用不同Bean，仅适用于WebApplicationContext 环境

globalSession 一般用于Portlet应用环境，该作用域仅适用于WebApplicationContext

环境Bean的生命周期问题



生命周期图解释

1.instantiate bean对象实例化

2.populate properties 封装属性

3.如果Bean实现BeanNameAware 执行 setBeanName

4.如果Bean实现BeanFactoryAware 执行setBeanFactory ，获取Spring容器

5.如果存在类实现 BeanPostProcessor（后处理Bean） ，执行postProcessBeforeInitialization

6.如果Bean实现InitializingBean 执行 afterPropertiesSet

7.调用<bean init-method="init">指定初始化方法 init

8.如果存在类实现 BeanPostProcessor（处理Bean） ，执行postProcessAfterInitialization,执行业务处理

9.如果Bean实现 DisposableBean 执行 destroy

10.调用<bean destroy-method="customerDestroy">指定销毁方法 customerDestroy

代理模式

静态代理

为某个对象提供一个代理，代理角色固定，以控制对这个对象的访问。代理类和委托类有共同的父类和父接口，这样在任何使用委托类对象的地方都可以用代理对象代替。代理类负责请求的预处理、过滤、将请求分派给委托类处理、以及委托类执行完请求后的后续处理。

动态代理

相比于静态代理，动态代理在创建代理对象上更加的灵活，它会根据需要通过反射机制在程序运行期动态的为目标对象创建代理对象，代理的行为可以代理多个方法， 即满足生产需要的同时又达到代码通用的目的。动态代理的两种实现方式 。

jdk方式实现

import java.lang.reflect.InvocationHandler;

import java.lang.reflect.Method;

import java.lang.reflect.Proxy;

/\*\*

\*jdk 动态代理

\*/

public class JdkHandler implements InvocationHandler{

// 目标类

private Object target;

public JdkHandler(Object target) {

this.target = target;

}

/\*\*

\* 程序运行期动态创建代理角色

\* @return

\*/

public Object getProxy(){

/\*\*

\* 获取代理对象

\* 1.类加载器

\* 2.目标类 实现的接口数组

\* 3.当前类

\* @return

\*/

return Proxy.newProxyInstance(this.getClass().getClassLoader(),

target.getClass().getInterfaces(),this);

}

public void before(){

System.out.println("婚礼现场紧张布置中......");

}

@Override//InvocationHandler内部参数 当前目标类的方法 参数

public Object invoke(Object proxy, Method method, Object[] args)throws Throwable {

before();//增强真实角色行为

Object result= method.invoke(target, args);// 执行真实角色方法

after();//增强真实角色行为

return result;

}

public void after(){

System.out.println("恭喜您成功进入人生第二阶段.....");

}

}

cglib方式实现

导包

<dependency>

<groupId>cglib</groupId>

<artifactId>cglib</artifactId>

<version>2.2.2</version>

</dependency>

实现

import java.lang.reflect.Method;

import net.sf.cglib.proxy.Enhancer;

import net.sf.cglib.proxy.MethodInterceptor;

import net.sf.cglib.proxy.MethodProxy;

public class CglibInterceptor implements MethodInterceptor {

private Object target;

public CglibInterceptor(Object target) {

this.target = target;

}

// 运行期动态创建代理类

public Object getProxy(){

Enhancer enhancer=new Enhancer();

//设置父类 class

enhancer.setSuperclass(target.getClass());

//设置回调对象实现接口的类

enhancer.setCallback(this);

return enhancer.create();

}

public void before(){

System.out.println("婚礼现场紧张布置中......");

}

@Override

public Object intercept(Object arg0, Method arg1, Object[] arg2,MethodProxy arg3)

throws Throwable {

before();//增强真实角色行为

Object result= arg3.invoke(target, arg2);

after();//增强真实角色行为

return result;

}

public void after(){

System.out.println("恭喜您成功进入人生第二阶段.....");

}

}

AOP

AOP概述

在软件业，AOP为Aspect Oriented Programming的缩写，意为：面向切面编程，通过预编译方式和运行期动态代理实现程序功能的统一维护的一种技术。

AOP是OOP（面向对象编程）的延续，是软件开发中的一个热点，也是Spring框架中的一个重要内容，是函数式编程的一种衍生范型。

利用AOP可以对业务逻辑的各个部分进行隔离，从而使得业务逻辑各部分之间的耦合度降低，提高程序的可重用性，同时提高了开发的效率。

AOP采取横向抽取机制，取代了传统纵向继承体系重复性代码

经典应用：事务管理、性能监视、安全检查、缓存 、日志等

Spring AOP使用纯Java实现，不需要专门的编译过程和类加载器，在运行期通过代理方式向目标类织入增强代码

AspectJ是一个基于Java语言的AOP框架，Spring2.0开始，Spring AOP引入对Aspect的支持，AspectJ扩展了Java语言，提供了一个专门的编译器，在编译时提供横向代码的织入

AOP实现原理

a. aop底层将采用代理机制进行实现。

b. 接口 + 实现类 ：spring采用 jdk 的动态代理Proxy。

c. 实现类：spring 采用 cglib字节码增强。

AOP基本概念

1.target：目标类，需要被代理的类。例如：UserService

2.Joinpoint(连接点):所谓连接点是指那些可能被拦截到的方法。例如：所有的方法

3.PointCut 切入点：已经被增强的连接点。例如：addUser()

4.advice 通知/增强，增强代码。例如：after、before

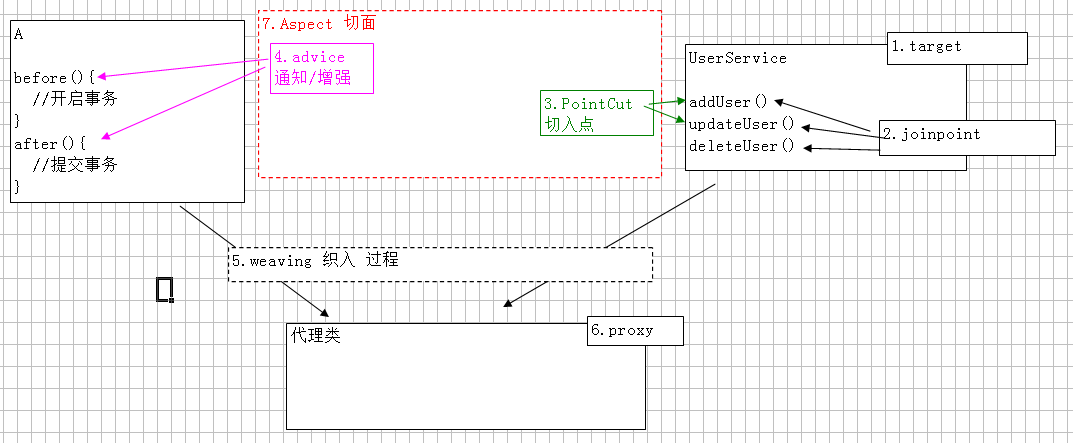
5. Weaving(织入):是指把增强advice应用到目标对象target来创建新的代理对象proxy的过程.

6.proxy 代理类

7. Aspect(切面): 是切入点pointcut和通知advice的结合

​ 一个线是一个特殊的面。

​ 一个切入点和一个通知，组成成一个特殊的面。



AOP解决日志处理问题

注解方式实现

jar包坐标引入

<dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjweaver</artifactId>

<version>1.8.9</version>

</dependency>

spring.xml配置

添加命名空间

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

1

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

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

配置AOP代理

<aop:aspectj-autoproxy/>

代码实现

import org.aspectj.lang.ProceedingJoinPoint;

import org.aspectj.lang.annotation.After;

import org.aspectj.lang.annotation.AfterReturning;

import org.aspectj.lang.annotation.AfterThrowing;

import org.aspectj.lang.annotation.Around;

import org.aspectj.lang.annotation.Aspect;

import org.aspectj.lang.annotation.Before;

import org.aspectj.lang.annotation.Pointcut;

import org.springframework.stereotype.Component;

/\*\*

\* 声明切面组件

\*/

@Component

@Aspect

public class LogCut {

/\*\*

\* 定义切入点 匹配方法规则定义

\* 匹配规则表达式含义 拦截 com.xxx.service 包下 以及子包下 所有类的所有

方法

\*/

@Pointcut("execution (\* com.xxx.service..\*.\*(..))")

public void cut(){}

/\*\*

\* 声明前置通知 并将通知应用到定义的切入点上

\* 目标泪方法执行前 执行该通知

\*/

@Before(value="cut()")

public void before(){

System.out.println("前置通知.....");

}

/\*\*

\* 声明返回通知 并将通知应用到切入点上

\* 目标类方法执行完毕执行该通知

\*/

@AfterReturning(value="cut()")

public void afterReturning(){

System.out.println("返回通知....");

}

/\*\*

\* 声明最终通知 并将通知应用到切入点上

\* 目标类方法执行过程中是否发生异常 均会执行该通知 相当于异常中的 finally

\*/

@After(value="cut()")

public void after(){

System.out.println("最终通知....");

}

/\*\*

\* 声明异常通知 并将通知应用到切入点上

\* 目标类方法执行时发生异常 执行该通知

\*/

@AfterThrowing(value="cut()",throwing="e")

public void afterThrowing(Exception e){

System.out.println("异常通知....方法执行异常时执行:"+e);

}

/

\*\*

\* 声明环绕通知 并将通知应用到切入点上

\* 方法执行前后 通过环绕通知定义相应处理

\*/

@Around(value="cut()")

public Object around(ProceedingJoinPoint pjp) throws Throwable{

System.out.println("环绕前置...");

System.out.println("环绕通知");

System.out.println(pjp.getTarget()+"--"+pjp.getSignature());

Object result=pjp.proceed();//执行目标对象方法

System.out.println("环绕后置...");

return result;

}

}

配置方式实现

配置切面，切入点，通知

<!-- aop 相关配置 -->

<aop:config>

<!-- aop 切面配置 -->

<aop:aspect ref="logCut">

<!-- 定义 aop 切入点 -->

<aop:pointcut expression="execution (\* com.mage.service..\*.\*(..))"

id="cut"/>

<!-- 配置前置通知 指定前置通知方法名 并引用切入点定义 -->

<aop:before method="before" pointcut-ref="cut"/>

<!-- 配置返回通知 指定返回通知方法名 并引用切入点定义 -->

<aop:after-returning method="afterReturning" pointcut-ref="cut"/>

<!-- 配置异常通知 指定异常通知方法名 并引用切入点定义 -->

<aop:after-throwing method="afterThrowing" throwing="e" pointcutref="cut"/>

<!-- 配置最终通知 指定最终通知方法名 并引用切入点定义 -->

<aop:after method="after" pointcut-ref="cut"/>

<!-- 配置环绕通知 指定环绕通知方法名 并引用切入点定义 -->

<aop:around method="around" pointcut-ref="cut"/>

</aop:aspect>

</aop:config>

定义Bean

import org.aspectj.lang.ProceedingJoinPoint;

import org.springframework.stereotype.Component;

/\*\*

\* 声明切面组件

\*/

@Component

public class LogCut {

public void before(){

System.out.println("前置通知.....");

}

public void afterReturning(){

System.out.println("返回通知....");

}

public void after(){

System.out.println("最终通知....");

}

public void afterThrowing(Exception e){

System.out.println("异常通知....方法执行异常时执行:" + e);

}

public Object around(ProceedingJoinPoint pjp) throws Throwable{

System.out.println("环绕前置...");

System.out.println("环绕通知");

System.out.println(pjp.getTarget()+"--"+pjp.getSignature());

Object result=pjp.proceed();

System.out.println("环绕后置...");

return result;

}

}

Spring JDBC事务管理配置

注解实现

配置事务管理器

<!-- spring 注解式事务声明 -->

<!-- 事务管理器定义 -->

<bean id="txManager"

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

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

</bean>

配置注解支持

<tx:annotation-driven transaction-manager="txManager"/>

\*\*Service 方法上在需要添加事务的方法上加入事物注解 \*\*

@Override

@Transactional(propagation=Propagation.REQUIRED)

public void saveUser(String userName,String userPwd){

User user1=new User();

user1.setUserName(userName);

user1.setUserPwd(userPwd);

userDao.saveUser(user1);

userDao.delUserById(2);

}

\*\*备注：\*\*默认 spring 事务只在发生未被捕获的 runtimeexcetpion 时才回滚。spring aop 异常捕获原理：被拦截的

方法需显式抛出异常，并不能经任何处理，这样 aop 代理才能捕获到方法的异常，才能进行回滚，默认情况下 aop只捕获runtimeexception 的异常，但可以通过配置来捕获特定的异常并回滚 换句话说在service 的方法中不使用try catch 或者在 catch 中最后加上 throw new RunTimeexcetpion（），这样程序异常时才能被 aop 捕获进而回滚.

mybatis

mybatis框架核心

1、 mybatis配置文件，包括Mybatis全局配置文件和Mybatis映射文件，其中全局配置文件配置了数据源、事务等信息；映射文件配置了SQL执行相关的 信息。

2、 mybatis通过读取配置文件信息（全局配置文件和映射文件），构造出SqlSessionFactory，即会话工厂。

3、 通过SqlSessionFactory，可以创建SqlSession即会话。Mybatis是通过SqlSession来操作数据库的。

4、 SqlSession本身不能直接操作数据库，它是通过底层的Executor执行器接口来操作数据库的。Executor接口有两个实现类，一个是普通执行器，一个是缓存执行器（默认）。

5、 Executor执行器要处理的SQL信息是封装到一个底层对象MappedStatement中。该对象包括：SQL语句、输入参数映射信息、输出结果集映射信息。其中输入参数和输出结果的映射类型包括HashMap集合对象、POJO对象类型。

mybatis环境搭建

新建maven项目 ,pom文件添加依赖jar

<!-- mybatis jar 包依赖 -->

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

<version>3.4.1</version>

</dependency>

<!-- 数据库驱动 -->

<dependency>

<groupId>mysql</groupId>

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

<version>5.1.39</version>

</dependency>

<!-- log4j日志打印 -->

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.16</version>

</dependency>

在src/main/resources 资源包下添加log4j日志输出properties文件，便于查看日志输出信息

# Global logging configuration

log4j.rootLogger=DEBUG, stdout

#Consoleoutput...

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

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

log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n

新建mybatis.xml文件，并加入配置信息如下(数据库名mybatis,表 user)

<?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">

<!-- 添加数据库连接相关配置信息 -->

<configuration>

<environments default="development">

<environment id="development">

<!-- 加入事务控制 -->

<transactionManager type="jdbc" />

<!-- 配置数据库连接信息 -->

<dataSource type="pooled">

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

<property name="url" value="jdbc:mysql://127.0.0.1:3306/spring-test" />

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

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

</dataSource>

</environment>

</environments>

<!-- mapper 配置文件指定 文件数量可配置多个-->

<mappers>

<mapper resource="com/mage/mapper/CustomerMapper.xml" />

</mappers>

</configuration>

新建CustomerMapper.xml配置文件

<?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">

<!--

1.命名空间配置 全局唯一 包名+文件名

2.配置Mapped Statement

3. statement配置

id 值声明statement编号 同一命名空间(同一文件)下不可重复

parameterType 输入参数即占位符的数据类型 类型可以是 基本数据类型,字符串,java bean ,map,list

等

resultType 输出结果 类型可以是基本数据类型,字符串,java bean,map等

statement 描述 即待执行的sql

#{id} 占位符 变量名为id/value 均可 ${value} 变量名必须为value 字符串拼接形式 无法避免

sql 注入

-->

<mapper namespace="com.mage.mapper.customerMapper">

<!-- 查询客户-->

<select id="queryCustomerById" parameterType="int" resultType="com.mage.pojo.Customer">

SELECT id,user\_name 'userName',user\_balance 'userBalance' FROM yg\_customer WHERE

id=#{id}

</select>

</mapper>

在父类工程pom获取资源

<build>

<resources>

<resource>

<directory>src/main/java</directory>

<includes>

<include>\*\*/\*.xml</include>

</includes>

</resource>

<resource>

<directory>src/main/resources</directory>

<includes>

<include>\*\*/\*.xml</include>

<include>\*\*/\*.properties</include>

</includes>

</resource>

</resources>

</build>

Mapper.xml映射文件

<?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.mage.dao.CustomerDao">

<select id="queryCustomerByCustomer" parameterType="customer" resultType="customer">

SELECT id,user\_name 'userName',user\_balance 'userBalance' from yg\_customer where id=#{id}

</select>

<select id="queryCustomerByName" parameterType="string" resultType="customer">

SELECT id,user\_name 'userName',user\_balance 'userBalance' from yg\_customer where user\_name=#{userName}

</select>

<select id="queryCustomerByNameForMap" parameterType="customer" resultType="customer">

SELECT id,user\_name 'userName',user\_balance 'userBalance' from yg\_customer where user\_name=#{userName}

</select>

<select id="queryCustomerForCount" resultType="int">

SELECT count(1) from yg\_customer

</select>

<resultMap type="customer" id="customer\_map">

<result column="id" property="id"/>

<result column="user\_name" property="userName"/>

<result column="user\_balance" property="userBalance"/>

</resultMap>

<sql id="customer\_column">

id,user\_name,user\_balance

</sql>

<select id="queryCustomerByName2" resultMap="customer\_map" parameterType="string">

SELECT <include refid="customer\_column"/>

FROM yg\_customer where user\_name like concat("%",#{userName},"%")

</select>

<select id="queryCustomerReturnMap" resultType="map" parameterType="int">

SELECT id,user\_name 'userName',user\_balance 'userBalance' from yg\_customer where id=#{id}

</select>

<insert id="addCustomerNoKey" parameterType="customer">

insert into yg\_customer (user\_name , user\_balance ) values (#{userName},#{userBalance})

</insert>

<insert id="addCustomerKey" parameterType="customer" useGeneratedKeys="true" keyProperty="id">

insert into yg\_customer (user\_name, user\_balance) values (#{userName},#{userBalance});

</insert>

<insert id="addCustomerBatch" parameterType="list">

insert into yg\_customer (user\_name, user\_balance) values

<foreach collection="list" item="item" separator=",">

(#{item.userName},#{item.userBalance})

</foreach>

</insert>

<update id="updateCustomerByCustomer" parameterType="customer">

update yg\_customer set user\_name=#{userName},user\_balance=#{userBalance} where id=#{id}

</update>

<update id="updateCustomerBatch" parameterType="list">

update yg\_customer set user\_balance='100' where id in

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

#{item}

</foreach>

</update>

<delete id="deleteCustomerById" parameterType="int">

delete from yg\_customer where id = #{id}

</delete>

<delete id="deleteCustomerBatch" parameterType="list">

delete from yg\_customer where id in

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

#{item}

</foreach>

</delete>

<!--动态sql-->

<select id="selectCustomerByName2" parameterType="map" resultType="customer">

select id,user\_name 'userName',user\_balance 'userBalance' from yg\_customer where 1=1

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

and user\_Name like '%${userName}%'

</if>

</select>

<select id="selectCustomerParams" resultType="customer" parameterType="map">

select id

<choose>

<when test="user!=null and user!=''">

,user\_name 'userName'

</when>

<otherwise>

,user\_balance 'userBalance'

</otherwise>

</choose>

from yg\_customer where 1=1

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

and user\_Name like '%${userName}%'

</if>

</select>

<select id="selectCustomerParams2" resultType="customer" parameterType="map">

select id

<choose>

<when test="user!=null and user!=''">

,user\_name 'userName'

</when>

<otherwise>

,user\_balance 'userBalance'

</otherwise>

</choose>

from yg\_customer

/\*使用where标签可以省略where 1=1\*/

<where>

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

and user\_Name like '%${userName}%'

</if>

</where>

</select>

<update id="updateCustomerById2" parameterType="customer">

update yg\_customer

<set>

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

user\_name=#{userName},

</if>

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

user\_balance=#{userBalance},

</if>

</set>

where id=#{id}

</update>

</mapper>

映射关系的问题

一对一关系

<?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.mage.dao.UserRoleDao">

<resultMap id="baseUserRole" type="userRole">

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

<result column="user\_id" property="userId"/>

<result column="role\_id" property="roleId"/>

<association property="role" column="role\_id" select="queryRoleById"/>

</resultMap>

<resultMap id="baseRole" type="role">

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

<result column="role\_name" property="roleName"/>

<result column="role\_remark" property="roleRemark"/>

</resultMap>

<select id="queryUserRoleById" resultMap="baseUserRole" parameterType="int">

select id,user\_id,role\_id from t\_user\_role where id=#{id}

</select>

<select id="queryRoleById" resultMap="baseRole" parameterType="int">

select id,role\_name,role\_remark from t\_role where id=#{role\_id}

</select>

</mapper>

一对多关系

<?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.mage.dao.GoodsDao">

<resultMap id="goodsBase" type="goods">

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

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

<collection property="imgageList" column="id" select="queryImgById"/>

</resultMap>

<resultMap id="imgsBase" type="goodsImgage">

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

<result column="goods\_skuid" property="goodsSkuid"/>

<result column="goods\_img" property="goodsImg"/>

</resultMap>

<select id="queryGoodsById" resultMap="goodsBase" parameterType="int" useCache="true">

select id,name from yg\_goods where id=#{id}

</select>

<select id="queryImgById" resultMap="imgsBase" useCache="true" parameterType="int">

select id,goods\_skuid,goods\_img from yg\_goods\_imgage where id=#{id}

</select>

</mapper>

mybatis缓存

一级缓存（默认开启）

基于 PerpetualCache 的 HashMap 本地缓存（mybatis 内部实现 cache 接口），其存储作用域为 Session，当

Session flush 或 close 之后，该 Session 中的所有 Cache 就将清空；

二级缓存

一级缓存其机制相同，默认也是采用 PerpetualCache 的 HashMap 存储，不同在于其存储作用域为

Mapper(Namespace)，并且可自定义存储源，如 Ehcache；

对于缓存数据更新机制，当某一个作用域(一级缓存 Session/二级缓存 Namespaces)的进行了 C/U/D 操作后，默认该作用域下所有 select 中的缓存将被 clear。

如果二缓存开启，首先从二级缓存查询数据，如果二级缓存有则从二级缓存中获取数据， 如果二级缓存没有，从一级缓存找是否有缓存数据，如果一级缓存没有，查询数据库。

局限性

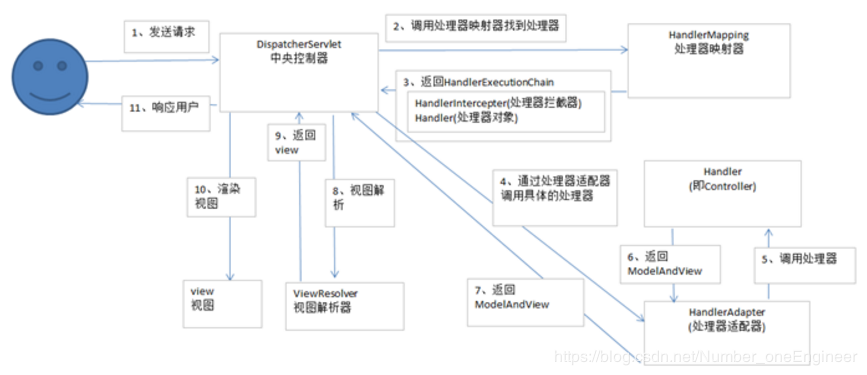
mybatis 二级缓存对细粒度的数据级别的缓存实现不好，对同时缓存较多条数据的缓存，比如如下需求：对商品信息进行缓存，由于商品信息查询访问量大，但是要求用户每次都能查询最新的商品信息，此时如果使用 mybatis 的二级缓存就无法实现当一个商品变化时只刷新该商品的缓存信息而不刷新其它商品的信息，因为 mybaits 的二 级缓存区域以 mapper 为单位划分，当一个商品信息变化会将所有商品信息的缓存数据全部清空 。

分布式缓存

如果有多条服务器 ，不使用分布缓存，缓存的数据在各个服务器单独存储，不方便系统开发。所以要使用分布式缓存对缓存数据进行集中管理。因此可是使用 ehcache memcached redis mybatis 本身来说是无法实现分布式缓存的，所以要与分布式缓存框架进行整合。 EhCache 是一个纯 Java 的进程内缓存框架，具有快速、精干等特点;Ehcache 是一种广泛 使用的开源 Java 分布式缓存。主要面向通用缓存,Java EE 和轻量级容器。它具有内存和 磁盘存储，缓存加载器,缓存扩展,缓存异常处理程序,一个 gzip 缓存 servlet 过滤器,支持REST 和 SOAP api 等特点。

springmvc

架构图



Spring MVC优势

1、清晰的角色划分：前端控制器（DispatcherServlet）、请求到处理器映射 （HandlerMapping）、处理器适配器（HandlerAdapter）、视图解析器 （ViewResolver）、处理器或页面控制器（Controller）、验证器 （

Validator）、命令对象（Command 请求参数绑定到的对象就叫命令对象）、表单对象（Form Object 提供给单展示和提交到的对象就叫表单对象）。

2、分工明确，而且扩展点相当灵活，可以很容易扩展，虽然几乎不需要；

3、和 Spring 其他框架无缝集成，是其它 Web 框架所不具备的；

4、可适配，通过 HandlerAdapter 可以支持任意的类作为处理器；

5、可定制性，HandlerMapping、ViewResolver 等能够非常简单的定制；

6、功能强大的数据验证、格式化、绑定机制；

7、利用 Spring 提供的 Mock 对象能够非常简单的进行 Web 层单元测试；

8、本地化、主题的解析的支持，使我们更容易进行国际化和主题的切换。

9、强大的 JSP 标签库，使 JSP 编写更容易。

还有比如 RESTful（一种软件架构风格，设计风格而不是标准，只是提供了一组设计原则和约束条件。它主要用于客户端和服务器交互类的软件，目前了解即可）风格的支持、简单的文件上传、约定大于配置的契约式编程支持、基于注解的零配置支持等等。

环境的搭建

springmvc 环境 jar 包依赖

<project xmlns="http://maven.apache.org/POM/4.0.0"

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

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/maven-v4\_0\_0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.xxx</groupId>

<artifactId>springmvc01</artifactId>

<packaging>war</packaging>

<version>0.0.1-SNAPSHOT</version>

<name>springmvc01 Maven Webapp</name>

<url>http://maven.apache.org</url>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

<scope>test</scope>

</dependency>

<!-- spring web -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>4.3.2.RELEASE</version>

</dependency>

<!-- spring mvc -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>4.3.2.RELEASE</version>

</dependency>

<!-- web servlet -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>3.0.1</version>

</dependency>

</dependencies>

<!-- jetty 插件 -->

<build>

<finalName>springmvc01</finalName>

<resources>

<resource>

<directory>src/main/resources</directory>

</resource>

</resources>

<plugins>

<!-- 编译环境插件 -->

<plugin>

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

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

<version>2.3.2</version>

<configuration>

<source>1.7</source>

<target>1.7</target>

<encoding>UTF-8</encoding>

</configuration>

</plugin>

<plugin>

<groupId>org.mortbay.jetty</groupId>

<artifactId>maven-jetty-plugin</artifactId>

<version>6.1.25</version>

<configuration>

<scanIntervalSeconds>10</scanIntervalSeconds>

<contextPath>/springmvc01</contextPath>

</configuration>

</plugin>

</plugins>

</build>

</project>

配置 web.xml (前端控制器配置)

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

<web-app id="WebApp\_ID" version="3.0"

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">

<!-- 表示容器启动时 加载上下文配置 这里指定 spring 相关配置 -->

<context-param>

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

<param-value>classpath:\*.xml</param-value>

</context-param>

<!-- 启用 spring 容器环境上下文监听 -->

<listener>

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

</listener>

<!-- 编码过滤 utf-8 -->

<filter>

<description>char encoding filter</description>

<filter-name>encodingFilter</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>UTF-8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>encodingFilter</filter-name>

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

</filter-mapping>

<!-- servlet 请求分发器 -->

<servlet>

<servlet-name>springMvc</servlet-name>

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

<init-param>

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

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

</init-param>

<!-- 表示启动容器时初始化该 Servlet -->

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

</servlet>

<servlet-mapping>

<servlet-name>springMvc</servlet-name>

<!-- 这是拦截请求, /代表拦截所有请求,拦截所有.do 请求 -->

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

</servlet-mapping>

</web-app>

servlet-context.xml 配置

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

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

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

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/mvc

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

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

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

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

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

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

<!-- 扫描 com.xxx.controller 下包 -->

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

<!-- mvc 请求映射 处理器与适配器配置-->

<mvc:annotation-driven/>

<!--配置视图解析器 默认的视图解析器- -->

<bean id="defaultViewResolver"

class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="viewClass"

value="org.springframework.web.servlet.view.JstlView" />

<property name="contentType" value="text/html" />

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

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

</bean>

</beans>

\*\*如果返回乱码：配置消息转换器 \*\*

<!-- 消息转换器 -->

<mvc:message-converters register-defaults="true">

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

<property name="supportedMediaTypes" value="text/html;charset=UTF-8"/>

</bean>

</mvc:message-converters>

三个框架的整合

jar 包依赖添加 (pom.xml)

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

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.mage</groupId>

<artifactId>ssm-demo</artifactId>

<version>1.0-SNAPSHOT</version>

<packaging>war</packaging>

<name>ssm-demo Maven Webapp</name>

<!-- FIXME change it to the project's website -->

<url>http://www.example.com</url>

<properties>

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

<maven.compiler.source>1.7</maven.compiler.source>

<maven.compiler.target>1.7</maven.compiler.target>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.11</version>

<scope>test</scope>

</dependency>

<!-- spring 核心 jar -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>4.3.2.RELEASE</version>

</dependency>

<!-- spring 测试 jar -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-test</artifactId>

<version>4.3.2.RELEASE</version>

</dependency>

<!-- spring jdbc -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

<version>4.3.2.RELEASE</version>

</dependency>

<!-- spring 事物 -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-tx</artifactId>

<version>4.3.2.RELEASE</version>

</dependency>

<!-- aspectj 切面编程的 jar -->

<dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjweaver</artifactId>

<version>1.8.9</version>

</dependency>

<!-- c3p0 连接池 -->

<dependency>

<groupId>c3p0</groupId>

<artifactId>c3p0</artifactId>

<version>0.9.1.2</version>

</dependency>

<!-- mybatis -->

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

<version>3.4.1</version>

</dependency>

<!-- 添加 mybatis 与 Spring 整合的核心包 -->

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis-spring</artifactId>

<version>1.3.0</version>

</dependency>

<!-- mysql 驱动包 -->

<dependency>

<groupId>mysql</groupId>

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

<version>5.1.39</version>

</dependency>

<!-- 日志打印相关的 jar -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>1.7.2</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.2</version>

</dependency>

<!-- 分页插件配置 -->

<dependency>

<groupId>com.github.pagehelper</groupId>

<artifactId>pagehelper</artifactId>

<version>4.1.0</version>

</dependency>

<!-- spring web -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>4.3.2.RELEASE</version>

</dependency>

<!-- spring mvc -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>4.3.2.RELEASE</version>

</dependency>

<!-- web servlet -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>3.0.1</version>

</dependency>

<!-- 添加 json 依赖 jar 包 -->

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-core</artifactId>

<version>2.7.0</version>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

<version>2.7.0</version>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-annotations</artifactId>

<version>2.7.0</version>

</dependency>

<!-- 文件上传包依赖 -->

<dependency>

<groupId>commons-fileupload</groupId>

<artifactId>commons-fileupload</artifactId>

<version>1.3.2</version>

</dependency>

<!--jstl包-->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

<version>1.2</version>

</dependency>

</dependencies>

<!--是程序可以读取到mapper下面的xml配置文件-->

<build>

<finalName>spring-mybatis</finalName>

<resources>

<resource>

<directory>src/main/java</directory>

<includes>

<include>\*\*/\*.xml</include>

</includes>

</resource>

<resource>

<directory>src/main/resources</directory>

<includes>

<include>\*\*/\*.xml</include>

<include>\*\*/\*.properties</include>

</includes>

</resource>

</resources>

</build>

</project>

web.xml 文件配置

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

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

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

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

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

id="WebApp\_ID" version="3.0">

<!--spring和mybatis的配置文件-->

<context-param>

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

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

</context-param>

<!--spring监听器-->

<listener>

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

</listener>

<!--编码过滤器-->

<filter>

<description>char encoding filter</description>

<filter-name>encodingFilter</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>UTF-8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>encodingFilter</filter-name>

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

</filter-mapping>

<!--springmvc servlet-->

<servlet>

<servlet-name>springmvc</servlet-name>

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

<init-param>

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

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

</init-param>

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

</servlet>

<!--springmvc需要拦截的内容-->

<servlet-mapping>

<servlet-name>springmvc</servlet-name>

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

</servlet-mapping>

</web-app>

Springmvc.xml 配置文件

<?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:mvc="http://www.springframework.org/schema/mvc"

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

xsi:schemaLocation="

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

http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd

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

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

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

http://www.springframework.org/schema/context/spring-context-3.0.xsd">

<!-- 扫描 com.xxx.controller 下包 -->

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

<!-- mvc 请求映射处理器与适配器 -->

<mvc:annotation-driven />

<!--配置视图解析器 默认的视图解析器- -->

<bean id="defaultViewResolver"

class="org.springframework.web.servlet.view.InternalResourceViewResolver">

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

<property name="contentType" value="text/html" />

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

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

</bean>

<!-- json 支持 -->

<bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping">

</bean>

<bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter">

<property name="messageConverters">

<list>

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

</list>

</property>

</bean>

<!-- 文件上传配置 -->

<bean id="multipartResolver" class="org.springframework.web.multipart.commons.CommonsMultipartResolver">

<property name="maxUploadSize">

<value>104857600</value>

</property>

<property name="maxInMemorySize">

<value>4096</value>

</property>

</bean>

<!-- &lt;!&ndash;全局异常的处理1&ndash;&gt;

<bean class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">

&lt;!&ndash;默认的异常视图&ndash;&gt;

<property name="defaultErrorView" value="error"></property>

&lt;!&ndash;前台获取数据&ndash;&gt;

<property name="exceptionAttribute" value="ex"></property>

<property name="exceptionMappings">

<props>

<prop key="com.mage.exception.TestException">error1</prop>

</props>

</property>

</bean>-->

</beans>

spring.xml配置

<?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"

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.xsd

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

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

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

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

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

http://www.springframework.org/schema/tx/spring-tx-3.0.xsd">

<!-- 扫描基本包 过滤 controller 层 -->

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

<context:exclude-filter type="annotation" expression="org.springframework.stereotype.Controller" />

</context:component-scan>

<!-- 加载 properties 配置文件 -->

<context:property-placeholder location="classpath:db.properties" />

<aop:aspectj-autoproxy /><!-- aop -->

<!-- 配置 c3p0 数据源 -->

<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">

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

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

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

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

</bean>

<!-- 配置事务管理器 -->

<bean id="txManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

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

</bean>

<!-- 设置事物增强 -->

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

<tx:attributes>

<tx:method name="get\*" read-only="true" />

<tx:method name="find\*" read-only="true" />

<tx:method name="query\*" read-only="true" />

<tx:method name="load\*" read-only="true" />

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

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

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

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

</tx:attributes>

</tx:advice>

<!-- aop 切面配置 -->

<aop:config>

<aop:pointcut id="servicePointcut" expression="execution(\* com.mage.service..\*.\*(..))" />

<aop:advisor advice-ref="txAdvice" pointcut-ref="servicePointcut" />

</aop:config>

<!-- 配置 sqlSessionFactory -->

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

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

<property name="configLocation" value="classpath:mybatis.xml" />

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

</bean>

<!-- 配置扫描器 -->

<bean id="mapperScanner"

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

<!-- 扫描 com.xxx.dao 这个包以及它的子包下的所有映射接口类 -->

<property name="basePackage" value="com.mage.dao" />

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

</bean>

</beans>

mybatis.xml配置

<?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>

<!--扫描实体类路径，自动取别名-->

<typeAliases>

<package name="com.mage.pojo"/>

</typeAliases>

<plugins>

<!-- com.github.pagehelper 为 PageHelper 类所在包名 -->

<plugin interceptor="com.github.pagehelper.PageHelper">

<property name="dialect" value="mysql"/>

<!-- 该参数默认为 false -->

<!-- 设置为 true 时，会将 RowBounds 第一个参数 offset 当成 pageNum 页码使用

-->

<!-- 和 startPage 中的 pageNum 效果一样 -->

<property name="offsetAsPageNum" value="true"/>

<!-- 该参数默认为 false -->

<!-- 设置为 true 时，使用 RowBounds 分页会进行 count 查询 -->

<property name="rowBoundsWithCount" value="true"/>

<!-- 设置为 true 时，如果 pageSize=0 或者 RowBounds.limit = 0 就会查询出全

部的结果 -->

<!-- （相当于没有执行分页查询，但是返回结果仍然是 Page 类型） -->

<property name="pageSizeZero" value="true"/>

<!-- 3.3.0 版本可用 - 分页参数合理化，默认 false 禁用 -->

<!-- 启用合理化时，如果 pageNum<1 会查询第一页，如果 pageNum>pages 会查询

最后一页 -->

<!-- 禁用合理化时，如果 pageNum<1 或 pageNum>pages 会返回空数据 -->

<property name="reasonable" value="true"/>

<!-- 3.5.0 版本可用 - 为了支持 startPage(Object params)方法 -->

<!-- 增加了一个`params`参数来配置参数映射，用于从 Map 或 ServletRequest

中取值 -->

<!-- 可以配置 pageNum,pageSize,count,pageSizeZero,reasonable,不配置映射

的用默认值 -->

<property name="params" value="pageNum=start;pageSize=limit;pageSizeZero=zero;reasonable=heli;count=countsql"/>

</plugin>

</plugins>

</configuration>

db.properties配置

jdbc.driver=com.mysql.jdbc.Driver

jdbc.url=jdbc:mysql://localhost:3306/crm?useUnicode=true&characterEncoding=utf8

jdbc.username=root

jdbc.password=123456

1

2

3

4

log4j.properties配置

# Global logging configuration

log4j.rootLogger=DEBUG, stdout

# Console output...

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

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

log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n

1

2

3

4

5

6

<property name="rowBoundsWithCount" value="true"/>

<!-- 设置为 true 时，如果 pageSize=0 或者 RowBounds.limit = 0 就会查询出全

部的结果 -->

<!-- （相当于没有执行分页查询，但是返回结果仍然是 Page 类型） -->

<property name="pageSizeZero" value="true"/>

<!-- 3.3.0 版本可用 - 分页参数合理化，默认 false 禁用 -->

<!-- 启用合理化时，如果 pageNum<1 会查询第一页，如果 pageNum>pages 会查询

最后一页 -->

<!-- 禁用合理化时，如果 pageNum<1 或 pageNum>pages 会返回空数据 -->

<property name="reasonable" value="true"/>

<!-- 3.5.0 版本可用 - 为了支持 startPage(Object params)方法 -->

<!-- 增加了一个`params`参数来配置参数映射，用于从 Map 或 ServletRequest

中取值 -->

<!-- 可以配置 pageNum,pageSize,count,pageSizeZero,reasonable,不配置映射

的用默认值 -->

<property name="params" value="pageNum=start;pageSize=limit;pageSizeZero=zero;reasonable=heli;count=countsql"/>

</plugin>

</plugins>

~~~

db.properties配置

jdbc.driver=com.mysql.jdbc.Driver

jdbc.url=jdbc:mysql://localhost:3306/crm?useUnicode=true&characterEncoding=utf8

jdbc.username=root

jdbc.password=123456

log4j.properties配置

# Global logging configuration

log4j.rootLogger=DEBUG, stdout

# Console output...

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

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

log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n