DO(领域对象)：一般和数据库中的表结构对应

DTO(数据传输对象)：服务层接收的数据(Controller接收的数据)

PO(持久对象)：在ORM中，PO跟持久层的数据结构对应，PO和数据库中的表一一对应

VO(值对象)：视图层对象，把某个指定的页面的所有数据封装起来

DTO和VO：DTO是服务层(Controller)封装客服端传递的数据，而VO是服务层返回给客户端显示的数据

DO和PO类似，只是PO用在ORM框架中

客户端---DTO--->服务层---DO--->Service层---DO--->服务层---VO--->客户端

Boolean类型的get方法使用isBoolean

Mybatis和数据库类型的对应

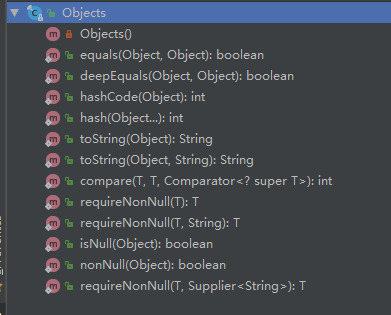


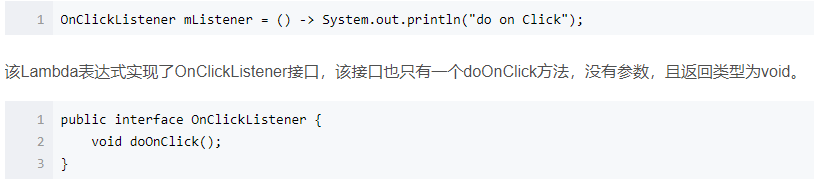
在MySQL中，可以使用TINYINT对Boolean类型和0,1进行转换

### lambda

Object和Objects的区别

Object是所有对象的父类，而Objects是Object的工具类

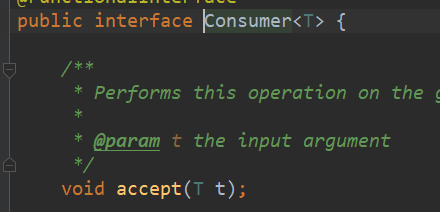
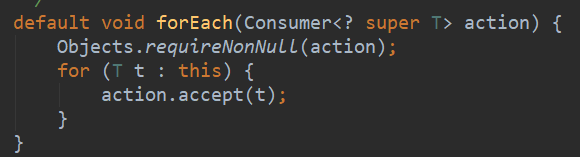




mListener相当于是OnClickListener的一个实现类，然后重写了doOnClick方法，方法体为System.out.println

List<Person> javaProgrammers = new ArrayList<Person>()；

forEach：用于遍历集合

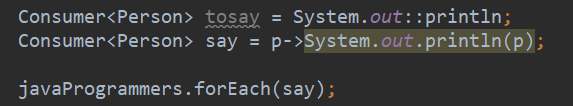


forEach的参数是一个Consumer，Consumer接口中的唯一一个未实现的方法accept，要求传入一个参数，且无返回值，所以，这里的lambda表达式只只能传入一个参数，然后无返回值的操作(person传入参数，System.out.println无返回值的操作)



可以使用此方法代替person -> System.out.println(person)操作

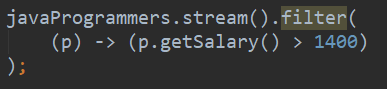
也可以将Consumer提取出来



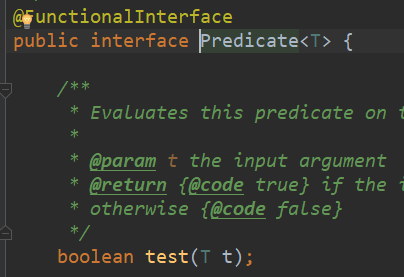


将集合转换为stream后，获得操作更多，可以对集合更多的操作

filter：对集合进行过滤操作

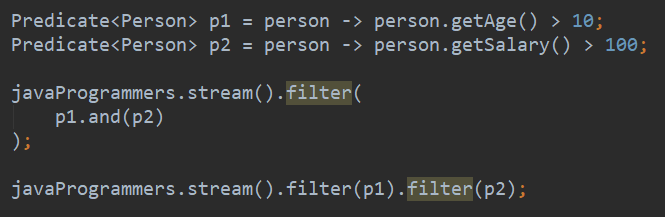






既然是过滤操作，肯定操作结果有true/false，所以filter中的lambda表达式操作必须有返回值，且返回值是boolean类型

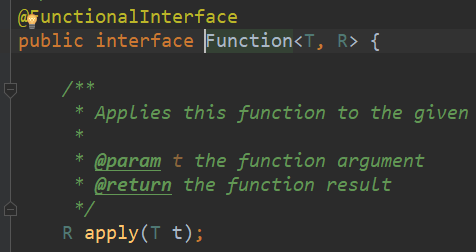
如果想使用Predict中的其他方法，就得创建Predict的事例



表示过滤条件是p1，p2都满足

map：将映射返回的结果存放到一个新的Stream中



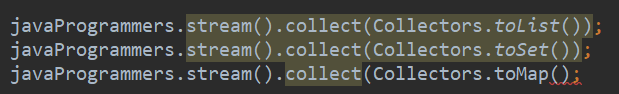


map操作是传入一个值，返回对这个值进行操作后，返回一个值，返回的值存放到一个集合中



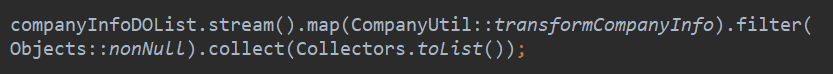
这里返回的就是一个Stream<Integer>

collect：将Stream转化为Collection集合



Stream还要其他很多操作，如获取max，min，limit，stored，count等

List<InfoDO>转List<Info>的操作

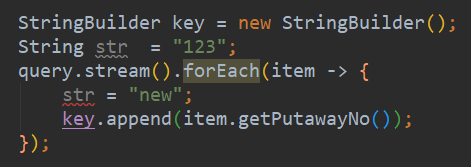
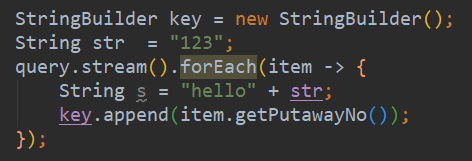


先转化stream，在通过map将结果映射到信息的Stream中，新的Stream中进行过滤操作，将Stream转化为List

Objects是对Object对象进行判断操作



Predict和IntPredict的区别：Predict是通用的模板，而IntPredict只能对int类型进行判断，但是Predict和IntPredict之间并无继承关系

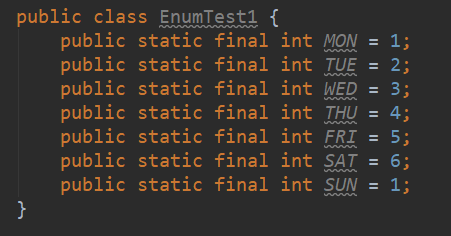


在lambda表达式中不能将外部引用值改变，str = “123”，只能在内部引用，不能在内部修改，但是key是引用类型，append调用的是对象的方法，key的引用未改变，如果在内部使用key = new StringBuilder就会报错，提示用户将key设置为final类型，这样在lamdba内部就不能修改了

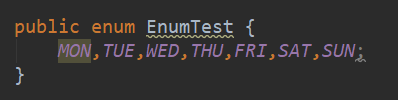
因为java对lambda表达式的实现决定的，在Java中lambda表达式是匿名类语法上的进一步简化，其本质还是调用对象的方法。 在Java中方法调用是值传递的，所以在lambda表达式中对变量的操作都是基于原变量的副本，不会影响到原变量的值。

### 枚举类型

类

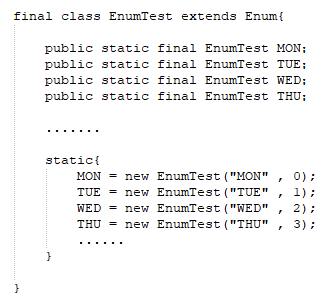


枚举

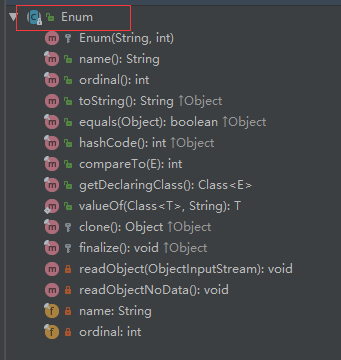


通过类定义常量的形式存在很多的不足，安全性：不能判断MON和SUN的值是否唯一，很容易混淆；

枚举类型在被编译器编译后，EnumTest会基础Enum类，而MON，TUE实际就是EnumTest



被编译后的文件

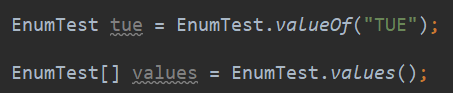


name()：获取枚举的MON，TUE

ordinal()：获取枚举的0,1(这里描述的是定义的顺序，所以存在不确定性)

name属性：枚举字符串的名称

ordinal属性：枚举的顺序



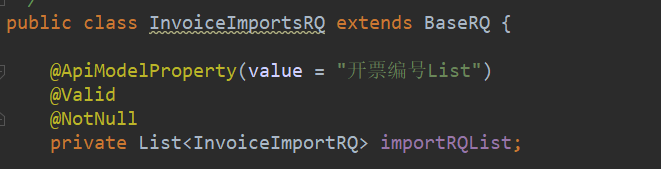
valueof和values是枚举中比较重要的方法，虽然在Enum中不能查看

values获取所有的枚举数组，valueof获取指定的枚举



### 错误：

Controller层接收List集合时的处理时，必须对List进行包装，不能直接将List暴露在Controller的方法中

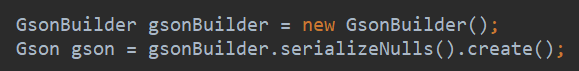


如果项目采用的Gson对Json进行转换就会报异常，因为Gson只知道importRQList是一个List而不知道List内部封装的对象是InvoiceImportRQ，Gson将Json转换后，List内部实际上是一个LinkedHashMap通过KV的方式保存json数据的，所以在调用

InvoiceImportRQ invoiceImportRQ = importRQList.get（0）的时候回报类型转化异常，因为get出来的是一个LinkedHashMap，单独使用get是不会报错的

但是如果使用SpringBoot默认的Jackson转换json就不会出现这种情况，所以最好还是对请求的List进行封装。

在使用Gson将对象转化为Json的时候，如果之间使用New Gson().toJson，如果对象中存在属性值为空，那么在转化的时候讲不会输出为null的属性名



使用serializeNulls返回的Gson，如果属性值为null，任然会输出属性名 age : null

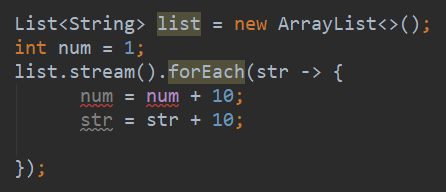
JDK1.8新特性：

接口中可以有实现方法，但是要有default关键字

Lambda表达式

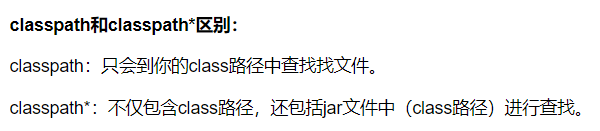
函数式接口(@FunctionalInterface)

Lambda表达式不能对外部数据进行修改，只能引用



@RequestMapping(produces="application/json; charset=utf-8")

Produces属性告知浏览器用何种方式何种编码解析返回值



Spring中读取外部properties文件和xml文件

Properties：

读取单个文件

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

读取多个文件

<bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">

<property name="locations">

<list>

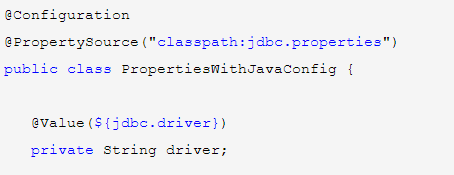
<value>classpath\*:conn.properties</value>

<value>classpath\*:configuration.properties</value>

</list>

</property>

</bean>



@PropertySource一般和@Configuration，@Value注解一起使用，@PropertySource会将properties文件中的属性存放到Spring的Environment中

Xml：

<import resource="classpath\*:applicationContext-redis.xml"/>

<context:annotation-config/>

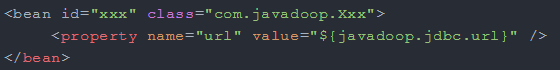
加入此标签后，会自动创建AutowiredAnnotationBeanPostProcessor，RequiredAnnotationBeanPostProcessor，CommonAnnotationBeanPostProcessor，PersistenceAnnotationBeanPostProcessor这4个BeanProcessor，分别对应@Autowired，@Resource、@PostConstruct、@PreDestroy注解，使用此标签后就可以不再添加BeanProcessor

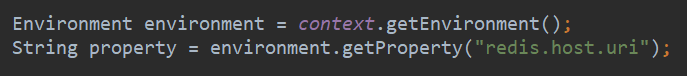
但是，一般在使用注解之前都会配置<context:component-scan base-package="pack.pack"/>

使用此标签后，就可以不用再配置<context:annotation-config/>

读取properties文件中的属性





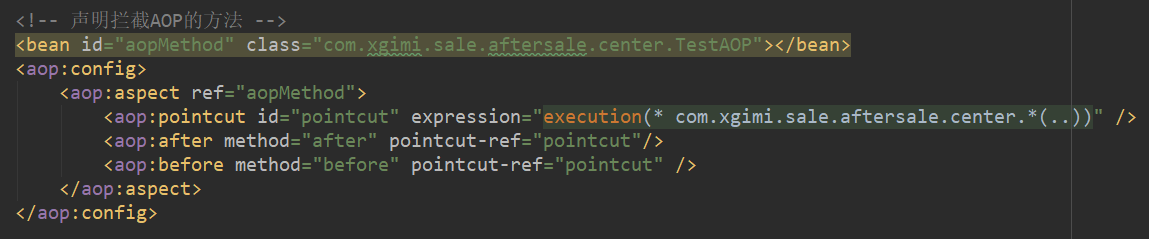


Environment中获取properties文件中的属性只能是通过@PropertySource加载的

对于application.properties文件，SpringBoot会默认加载

@ConfigurationProperties(prefix = "")会自动到properties文件为属性赋值

基于XML的方式配置AOP



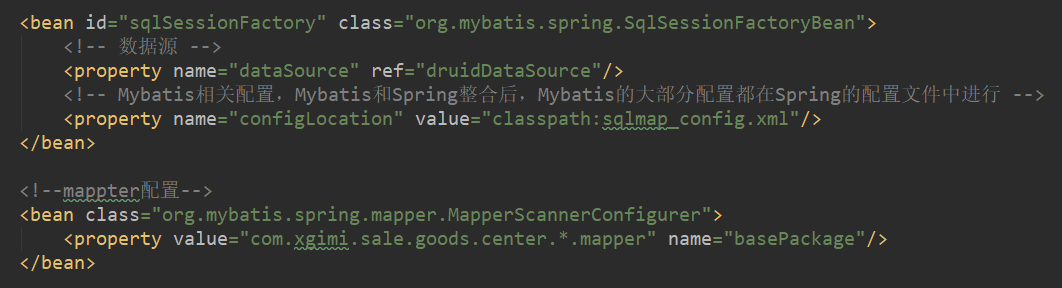
基于XML声明事务



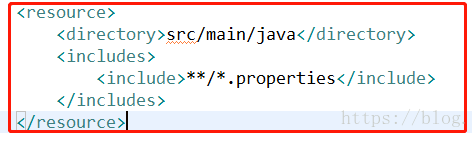
事务一般配置在service方法上

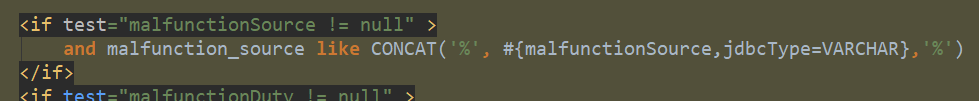
<aop:aspect>与<aop:advisor>的区别

<aop:aspect>用于定义切面，<aop:advisor>用于定义事务

mvn在打包后，会在将resource下的xml同步到和mapper.java所在的文件夹(文件夹相同)

在使用mvn打包的时候，打包后的jar/war没有properties文件，只有在pom文件中添加resource的才会将properties文件打包





Mybatis CONCAT函数

### Mybatis-Spring：

Mybatis和Spring整合后和原来的区别：

原来获取SQLSessionFactory

sqlSessionFactoryBuilder.build(配置文件)

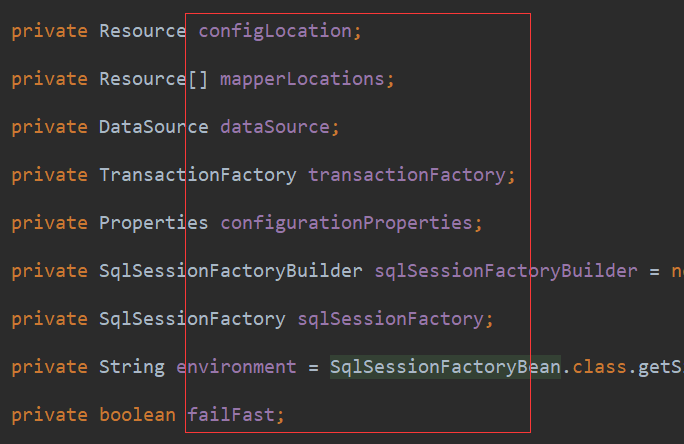
整合后：通过SqlSessionFactoryBean获取

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

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

</bean>

SqlSessionFactoryBean是Spring的一个FactoryBean，实际是调用getObject获取sqlSessionFactory

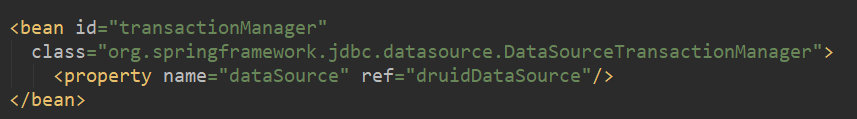


SqlSessionFactoryBean的属性，在创建SqlSessionFactoryBean的时候可以指定这些属性值，比如sqlSessionFactory，虽然SqlSessionFactoryBean可以自动创建sqlSessionFactory，也可以手动设置sqlSessionFactory的类型，可以看getObject方法，看获取sqlSessionFactory的过程

整合后配置Mybatis的套路：

创建数据源：DataSource

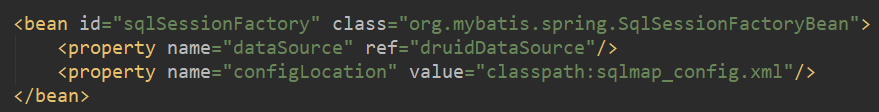
创建事务管理：DataSourceTransactionManager



为事务管理创建管理机制

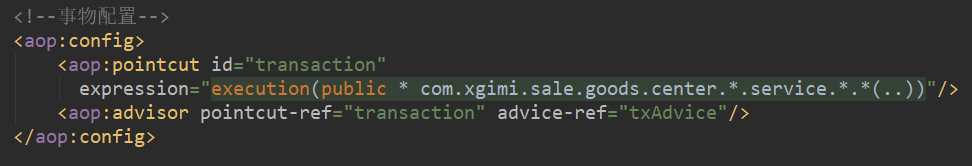


创建sqlSessionFactory：



上面的配置都可以到sqlSessionFactory中配置，看源码

为方法添加事务



DataSource

DataSourceTransactionManager

事务的行为

sqlSessionFactory

DataSource应该有事务管理，事务管理应该有事务管理的机制(行为)

sqlSessionFactory用于获取数据源，应该有DataSource

sqlSessionTemplate

有了sqlSessionFactory之后就是获取sqlSession，Mybatis中获取的是DefaultSqlSession，而整合后获取的是sqlSessionTemplate，

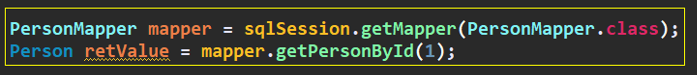


有了sqlSession就可以执行CRUD操作

MapperFactoryBean

获取Mapper

未整合

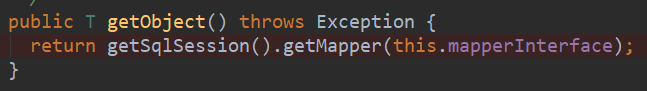


整合后

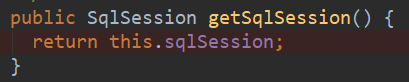
直接在service方法中使用@AutoWire注解就可以实现

如何实现的？

MapperFactoryBean的getObject方法

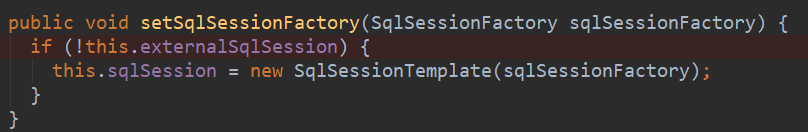


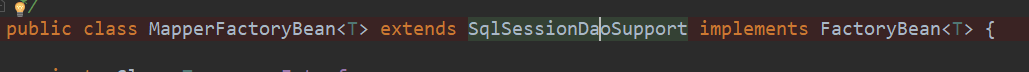
getSqlSession()：获取sqlSession，MapperFactoryBean继承了SqlSessionDaoSupport



此抽象类的getSqlSession方法获取sqlSession

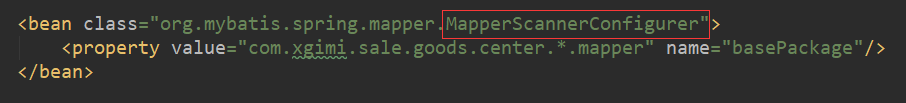
MapperFactoryBean除了为Mapper创建代理对象外，还创建了sqlSessionTemplate，在SqlSessionDaoSupport下





MapperFactoryBean继承SqlSessionDaoSupport

MapperScannerConfigurer

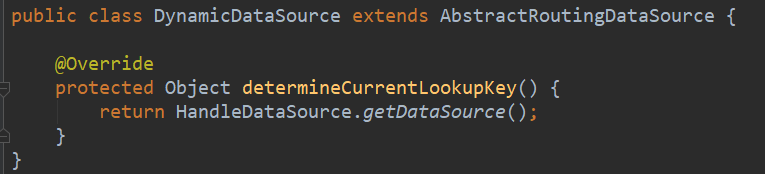


MapperScannerConfigurer的作用是加载mapper及其对应的class文件

MapperScannerConfigurer创建好后，就会去创建MapperFactoryBean(包含创建Mapper的代理对象和sqlSessionTemplate)

### Mybatis双数据源的实现：

先创建AbstractRoutingDataSource的实现类

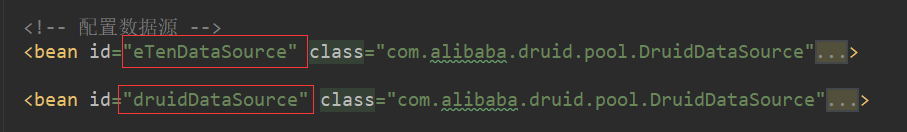


AbstractRoutingDataSource是spring-jdbc包下的一个类，在和数据库操作时，会调用determineCurrentLookupKey方法获取DataSource

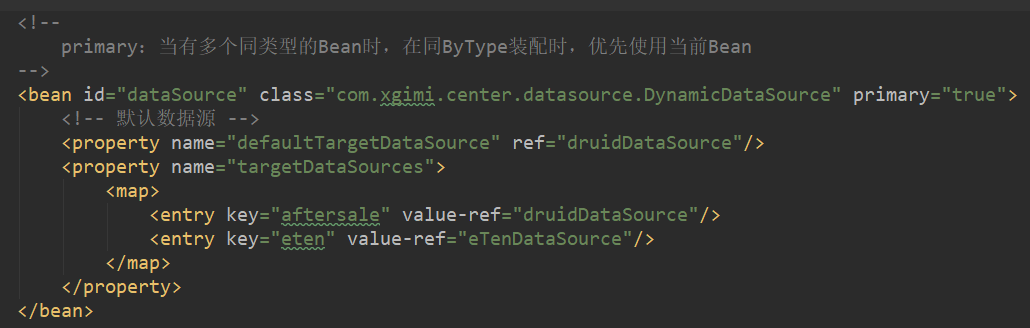
而这里的HandleDataSource主要是一个工具类，对DataSource的处理



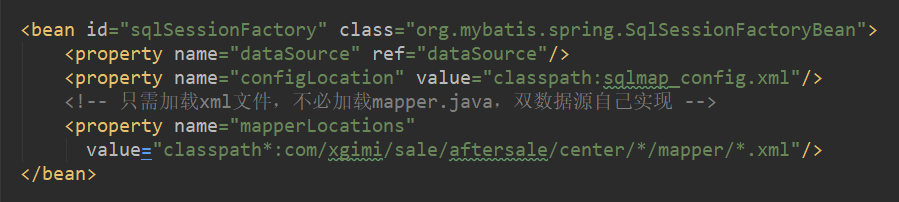
创建数据源(DataSource)



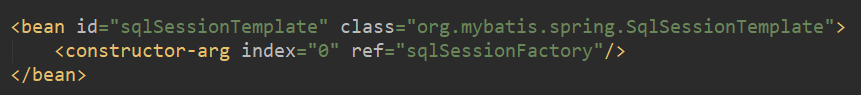
实例化AbstractRoutingDataSource



创建sqlSessionFactory



创建sqlSessionTemplate



这里为什么要单独创建sqlSessionTemplate和mapperLocations

一般加载mapper及xml都是通过MapperScannerConfigurer(自动创建mapper的代理对象，sqlSessionTemplate)，由于是多数据源，执行哪个数据库应该在执行，mapper中的方法时就得确定，所以我们必须自己手动实现mapper接口的方法，不能使用默认的，所以就不能使用MapperScannerConfigurer，就得自己创建sqlSessionTemplate和加载mapper.xml

既然数据库的确定在mapper中，那么事务也应该在mapper的实现类中声明

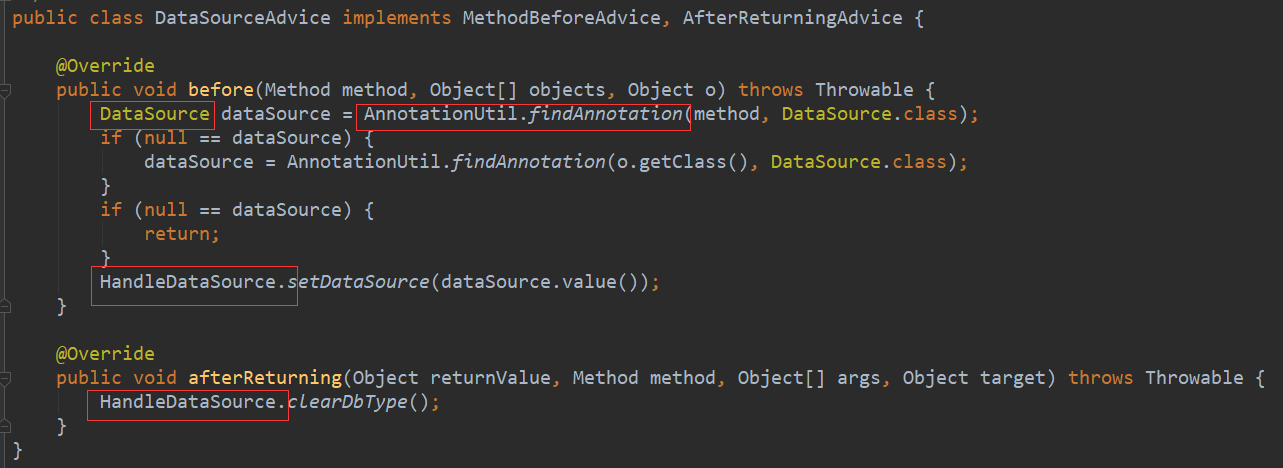


实现mapper类



SQLSessionDaoSupport的作用是获取sqlSessionTemplate

@DataSource是自定义的注解，用于判断数据源

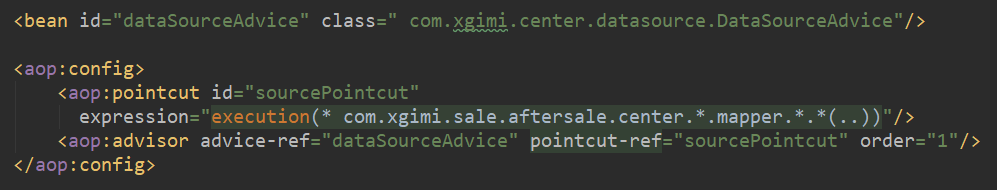


DataSourceAdvice类的作用就是用来对@DataSource注解进行拦截，当执行mapper中的方法时就会调用DataSourceAdvice中的方法

AnnotationUtil.findAnnotation(method, DataSource.class)：就是

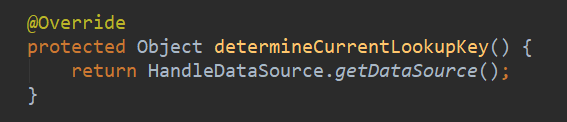
method.getAnnotation(DataSource.class);

这里DataSource是注解



MethodBeforeAdvice就是前置通知

方法执行之前将数据源的名称存放到HandleDataS的ThreadLocal<String>中



然后在执行的时候获取，执行完毕之后释放

基本大致是这样，唯一的缺点就是都要是实现mapper接口，不能使用默认的

file.delete和file.deleteOnExit的区别：

都是删除文件操作，只是删除的时机不同，file.delete是直接删除文件，而file.deleteOnExit是在整个程序运行完再删除，即JVM在关闭的时候才删除。

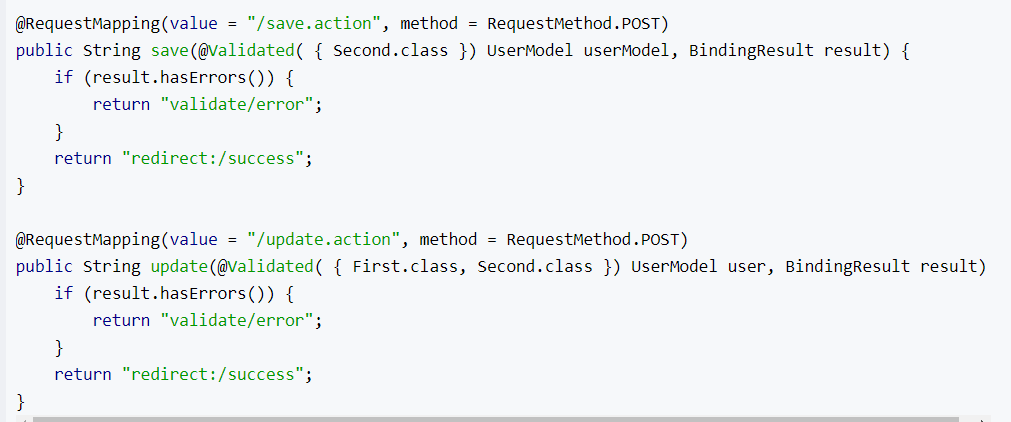
### 异常

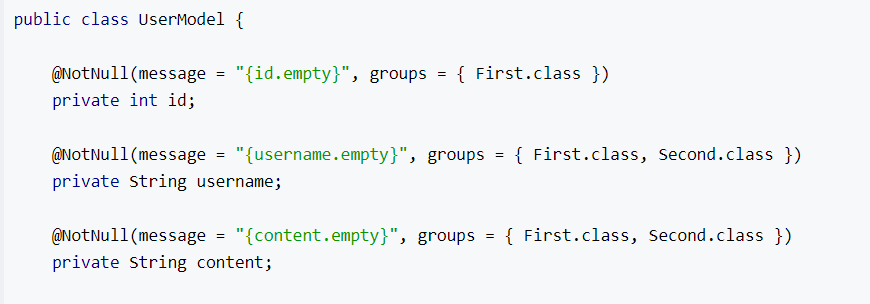
@Valid @Validated的区别：

@Valid是java的注解JSR303

@Validated是SpringMVC的注解

@Validated支持分组校验

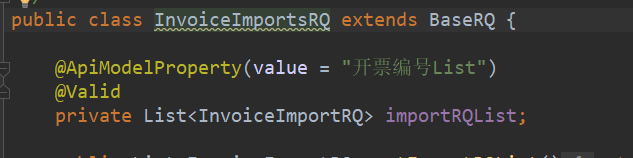


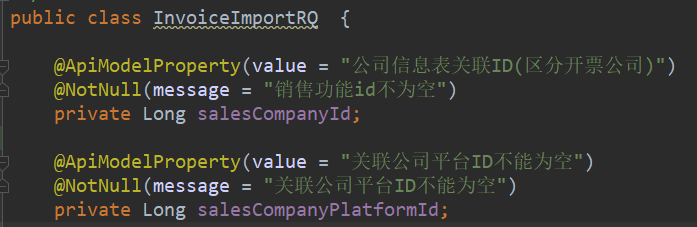


@Validated：可以用在类型、方法和方法参数上。但是不能用在成员属性（字段）上

@Valid：可以用在方法、构造函数、方法参数和成员属性（字段）上







@Valid支持嵌套校验，而且也不要在Controller的请求参数中写@Valid，@Validated不能作用在成员属性(Bean的属性)，如果使用@Validated校验，在Controller的方法参数中必须写，而@Valid作用在成员属性中，嵌套校验



将Service中的异常往Controller层抛，在Controller中读取异常信息，返回给用户

@ControllerAdvice+@ExceptionHandler可以对Controller层抛出的异常进行处理



当Controller中抛异常时，会执行@ControllerAdvice中的方法，如果异常类型和@ExceptionHandler中声明的异常类型一致，则执行指定的方法，这样就可以获取到整个程序的异常，并可以将异常结果返回用户

关键：在Service层或其他层中的异常必须throw，往上层抛，不能将异常打印了，这样就会导致Controller层获取不异常

### 日志

MessageFormat.format(“票号：{0}，流水号：{1}，代码：{2}”, “11” , “22” , “33” )；

JDK自带的类可以将{}中的内容自动转化到{}中

Bean实例化初始化操作：@PostConstruct，init-method，Initl什么接口

分布要求，如果一方挂了，对调用方法的影响要降到最低

Dubbo可计算生产者和消费者之间调用的次数，方便后期性能优化

面向对象特征：

抽象：将类的共同进行总结，包含数据抽象以及行为抽象

继承：得到父类的数据和行为

多态：允许不同子类型对同一消息做出不同相应，多态分为编译是多态和运行时多态，方法的重载和重写等

封装：把数据进行封装，提供操作数据的方法，使得类封闭(将信息隐藏)

short s1 = 1; s1 = s1 + 1 -----错误，s1+1是int，需强制类型转换

short s1 = 1; s1 += 1; -----正确，s1+=1;其实执行的是s1 = (short) (s1 + 1)，强制转换了

数组没有length()方法，有length属性，String有length()方法

重载对返回类型没要求，方法名，参数个数，参数顺序

怎样将GB2312编码的字符串转换为ISO-8859-1编码的字符串

String s2 = new String(s1.getBytes("GB2312"), "ISO-8859-1");

序列化时，只对对象的状态进行保存，而不管对象的方法

IO是面向流的，NIO是面向缓冲区的

serialVersionUID：它决定着是否能够成功反序列化

ClassLoader：类加载器有哪些，双亲委派

WebService和http：

WebService基于soap协议，soap是http协议的基础上，基于xml的协议

http的请求和响应封装在请求头里，而soap的请求和响应封装到xml中

rpc和http：

rpc效率高，传输的是二进制字节，没有http请求头之类无用信息

http迭代更快，更加灵活

SOA：面向服务编程

将服务拆分为多个模块后，会涉及服务与服务之间的相互调用，松耦合

微服务和soa在含义上类似

获取Class对象：

类名.getClass

对象.class

Class.forName()

获取实例对象：

new

clone

反射

序列化

AOP动态代理实现，如果代理是接口实现，可以通过proxy-target-class=false/true指定是使用JDK(false)还是Cglib(true)[默认false]，但是当代理不是接口实现，那么spring会默认使用cglib

Spring会动态匹配代理模式