**Vastbase G100 V2.2**

**(Build 10)**

**应用适配手册**

【版权声明】

©2007-2022 北京海量数据技术股份有限公司 版权所有

本文档著作权归 **北京海量数据技术股份有限公司**（简称“海量数据”）所有，未经海量数据事先书面许可，任何主体不得以任何形式复制、 修改、抄袭、传播全部或部分本文档内容。

北京海量数据技术股份有限公司保留所有的权利。

【服务声明】

本文档意在向客户介绍海量数据全部或部分产品、服务的当时的整体概况，部分产品、服务的内容可能有所调整。您所购买的产品、服务的种类、服务标准等应由您与海量数据之间的商业合同约定，除非双方另有约定，否则，海量数据对本文档内容不做任何明示或模式的承诺或保证。

**目录**

[1. 适配一览表 3](#_Toc16886)

[2. 概述 4](#_Toc11958)

[3. ORM框架适配 5](#_Toc23386)

[3.1. 驱动及连接池配置 5](#_Toc19957)

[3.2. Mybatis 6](#_Toc22779)

[3.2.1. 类名调整 6](#_Toc27157)

[3.2.2. 字段名称转换 7](#_Toc20533)

[3.2.3. 字段值处理 8](#_Toc442)

[3.2.3.1. Null值处理 8](#_Toc28908)

[3.2.3.2. 其他数据类型处理 8](#_Toc18883)

[3.2.4. 结果集封装 11](#_Toc30437)

[3.2.4.1. Map封装 11](#_Toc1981)

[3.2.5. 动态SQL 13](#_Toc3819)

[3.2.6. 存储过程及函数 13](#_Toc1263)

[3.2.6.1. Vastbase语法方面 13](#_Toc5677)

[3.2.6.2. 函数与Mybatis结合使用 14](#_Toc1733)

[3.3. MybatisPlus 15](#_Toc3446)

[3.3.1. MybatisPlus的数据库连接问题 15](#_Toc24489)

[3.3.2. 设置数据库表中的创建时间字段和更新时间字段的自动插入 15](#_Toc10026)

[3.3.3. 乐观锁和逻辑删除字段的配置 16](#_Toc906)

[3.3.3.1. 逻辑删除字段 16](#_Toc20017)

[3.3.3.2. 乐观锁字段 17](#_Toc9625)

[3.3.4. 分页配置（MybatisPlus分页插件的使用） 19](#_Toc13622)

[3.3.4.1. 使用分页的原因 19](#_Toc20751)

[3.3.4.2. 分页插件的本质 19](#_Toc14566)

[3.3.4.3. 分页插件的配置和使用 19](#_Toc15601)

[3.3.5. 特殊类型的字段后端传递参数 20](#_Toc29965)

[3.3.5.1. 几何类型point 20](#_Toc23345)

[3.3.5.2. Json类型、bit类型以及bytea类型 20](#_Toc11210)

[3.3.6. MybatisPlus封装的方法 21](#_Toc26016)

[3.3.6.1. Mapper crud接口 21](#_Toc22319)

[3.3.6.2. Service crud接口 21](#_Toc32640)

[3.3.7. 函数和存储过程 22](#_Toc16642)

[3.4. Hibernate 22](#_Toc30921)

[3.4.1. Hibernate配置数据库的连接 22](#_Toc6533)

[3.4.2. 实体类需要注意的事项 23](#_Toc26703)

[3.4.3. 持久层Repository需要注意的事项 23](#_Toc3429)

[3.4.4. 各种基本类型的操作 24](#_Toc15475)

[3.4.4.1. 时间类型 24](#_Toc7238)

[3.4.4.2. 整型 25](#_Toc7687)

[3.4.4.3. 字符类型 25](#_Toc15492)

[3.4.5. Hibernate操作存储过程 26](#_Toc8066)

[4. 中间件适配 28](#_Toc32284)

[4.1. Tomcat 28](#_Toc17089)

[4.2. ShardingSphere 30](#_Toc11751)

[4.2.1. ShardingSphere-JDBC 30](#_Toc29158)

[4.2.1.1. 数据分片 30](#_Toc17283)

[4.2.1.2. 分布式事务 31](#_Toc11283)

[4.2.1.3. 读写分离 32](#_Toc30618)

[4.2.2. ShardingSphere-Proxy 33](#_Toc27922)

[4.2.2.1. 数据分片 33](#_Toc3442)

[4.2.2.2. 读写分离 34](#_Toc20265)

# 适配一览表

| **适配组件** | **适配对象** | **适配版本** | **适配说明** |
| --- | --- | --- | --- |
| 连接池 | Alibaba Druid | 1.1.24 | 适配 |
| ORM框架 | Mybatis | 3.4.6 | 适配 |
| MybatisPlus | 3.2.0 | 适配 |
| Hibernate | 5.5.0 Final | 适配 |
| 中间件 | Tomcat | 7.0.109 | 适配 |
| ShardingSphere-JDBC | 4.1.1 | 适配 |
| ShardingSphere-Proxy | 4.1.1 | 适配 |

# 概述

概述

本章介绍如何数据库迁移至Vastbase后应用程序的兼容适配处理，以解决由于数据库不同特点导致原有应用运行出错的问题。

分为如下几个方面进行适配说明：

1. ORM框架
2. 中间件

读者对象

本文档是为基于Vastbase G100进行Java应用程序开发的程序员而写的，提供了必要的参考信息。

作为应用程序开发人员，至少需要了解以下知识：

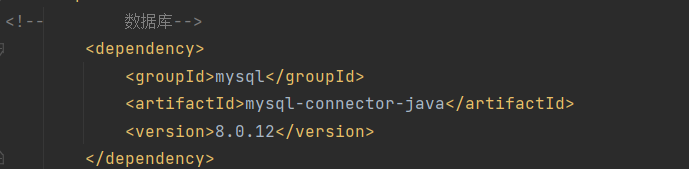
* 操作系统知识。这是一切的基础。
* Java语言。这是做应用程序开发的基础。
* 熟悉Java的一种IDE。这是高效完成开发任务的必备条件。
* SQL语法。这是操作数据库的必备能力。
* ORM框架。这是适配这一部分的基础
* 中间件Tomcat和ShardingSphere。同样是适配这一部分的必要条件

# ORM框架适配

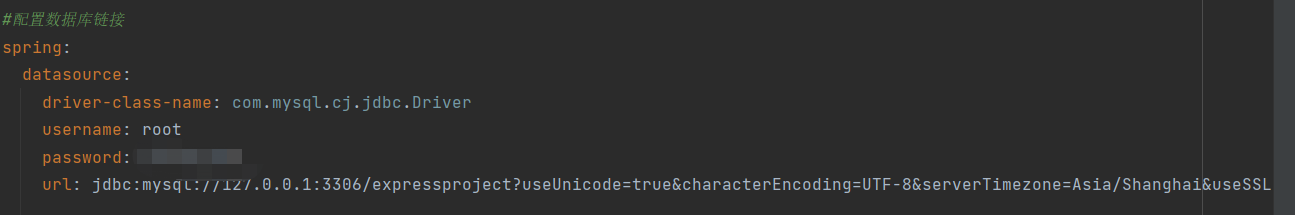
## 驱动及连接池配置

我们知道对于不同的数据库我们需要配置不同的数据库驱动，在Springboot项目中，我们可以在application.yml中进行配置。如对于MySQL8.0版本，需要进行如下操作：

Step1:在pom.xml中导入依赖：



Step2:在application.yml中进行配置



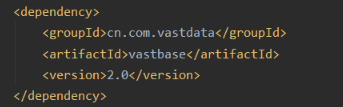
驱动四要素即：driver、username、password以及url，这里使用的数据库连接池为默认连接池。

而对于vastbase，则需要导入vastbase的驱动，以2.0版本为例：

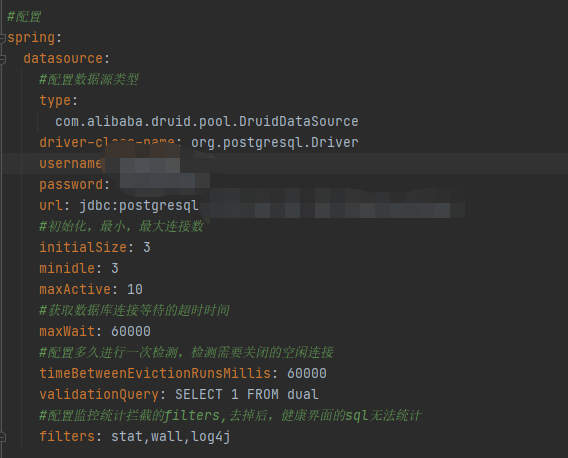
Step1:Maven本地导入Vastbase-G100-2.0\_xxxxxxxxxx.jar依赖

mvn install:install-file "-Dfile=Vastbase-G100-x.x\_xxxxxxx.jar" "-DgroupId=cn.com.vastdata" "-DartifactId=vastbase" "-Dversion=2.0" "-Dpackaging=jar"

Step2:在pom.xml中导入



Step3:配置数据源和驱动



同样只需要配置驱动四要素：driver、username、password以及url

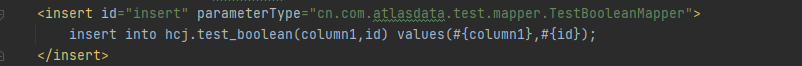
而其中的com.alibaba.druid.pool.DruidDataSource是市面上较为流行的性能也相对较高的数据库连接池druid，除此之外还有C3P0,DBCP等主流数据库连接池。不同连接池共同的地方在于需要配置驱动四要素，尽管命名可能有略微不同。我们只需要根据其官方文档来进行命名配置即可。

而如上所示的druid连接池中我们还可以配置多个属性如initialSize等，可以根据项目的需要进行相关配置。

## Mybatis

### 类名调整

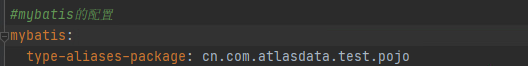
Mybatis需要用到类名时，需要写入类的全路径，这样无疑耗费时间，如下例当我们需要使用cn.com.atlasdata.test.mapper包下的TestBooleanMapper类时，在Mapper中的写法如下：



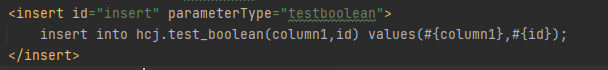
也就是当我们的resultType或parameterType为实体类时，我们需要写这个类的全路径，否则将会报错如下：



Mybatis提供了一种简便的方法解决这一问题，也就是“扫描包变小写”



我们只需要在application.yml中配置这一段，在Mapper.xml中便可以使用类名小写来替换类名的全路径，如下：



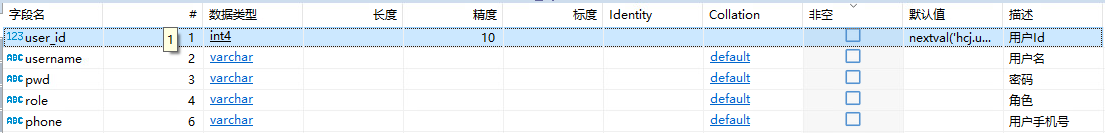
但是，在不断的尝试过程中，我发现这一方法并没有针对性得要求一定是小写，而是不区分大小写，也就是说这这里我们使用Testboolean、TESTBoolean都是可以的。

### 字段名称转换

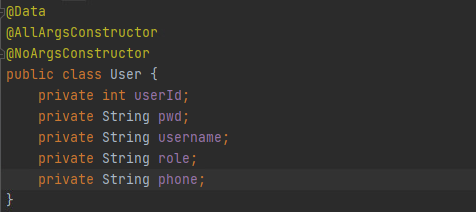
由于Windows环境下数据库不区分大小写，因此我们对表名或字段名进行命名时通常使用下划线如user\_id。但是，我们在Java开发中，为了遵守开发规范，命名一般为驼峰命名，如userId。这样,Mybatis就可能报错没有user\_id的get和set方法。

如下例：

数据库的user表

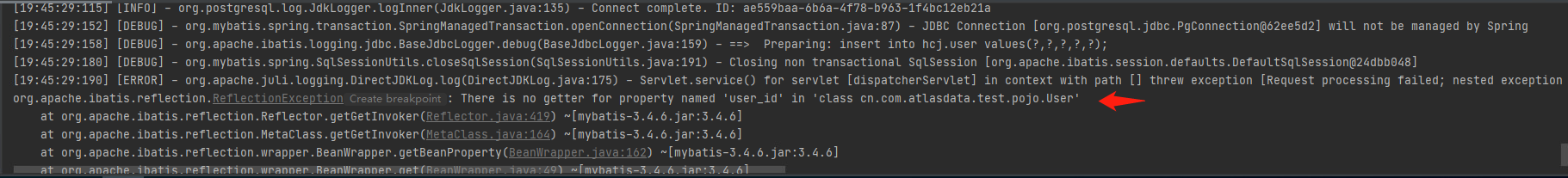


Java的User类





此时，我们搭建好Mybatis并运行，便会出现如下情况：



因此，我们需要将数据库的下划线命名和Java开发的驼峰命名进行转换。解决这一问题有多种方法，但最简便的一种便是Mybatis提供的下划线驼峰命名转换。

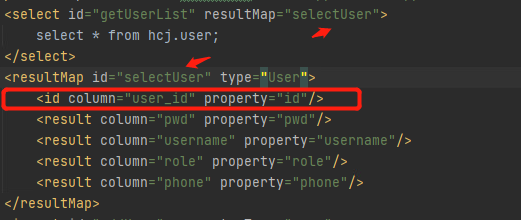
（1）Mybatis提供了十分简便的解决方法，我们只需要在yml的配置中加入这一段，也就是配置下划线转驼峰，而后Mybatis便会自动将SQL中查出来的带下划线的字段转换为驼峰命名格式，再去匹配类中的属性，问题便得已解决。



1. 也可以通过给字段起别名的方式来解决



1. 用resultMap映射



### 字段值处理

#### Null值处理

一般而言，对于数据库中的字段，其值可能为null，因此在Java类中的属性类型设置时，我们应当使用包装类即Integer代替int等，使其可与数据库字段取值范围对应。

#### 其他数据类型处理

在Vastbase中，采用的是Postgresql的语法，因此同样也支持Postgresql的相关基本类型，这些数据类型与Java数据类型的对应关系如下

（1）数值类型

|  |  |
| --- | --- |
| 数据库基本类型 | Java类型 |
| tinyint | Integer |
| smallint | Integer |
| integer | Integer |
| bigint | Long |
| numeric | BigDecimal |
| real | Float |
| smallserial | Integer |
| serial | Integer |
| bigserial | Long |
| oid | Long |

（2）货币类型

|  |  |
| --- | --- |
| 数据库基本类型 | Java类型 |
| money | BigDecimal |

注意货币类型Money对应的是Java类型中的BigDecimal而不是Float或者Double，原因在于Float和Double都是近似值，当用于金额的加减乘除时，可能会出错，这在金额的计算中显然是不被允许的，使用BigDecimal便能很好解决这一点。

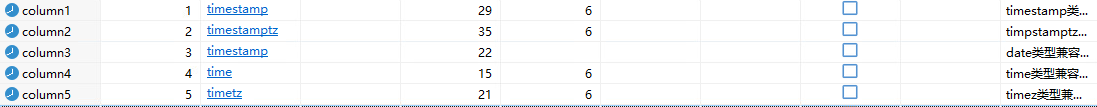
（3）字符类型

|  |  |
| --- | --- |
| 数据库基本类型 | Java类型 |
| char | Charater或String(前端传参需要注意字符数只能为1） |
| bpchar | Charater或String(前端传参需要注意字符数只能为1） |
| varchar | String |
| nvarchar2 | String |
| text | String |
| name | String |

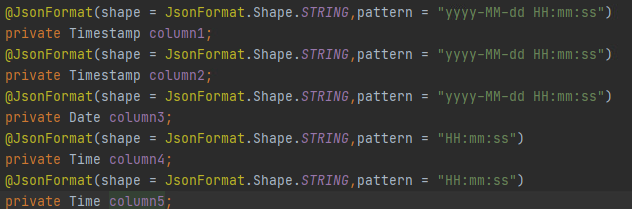
（4）时间类型

|  |  |
| --- | --- |
| 数据库基本类型 | Java类型 |
| timestamp | Timestamp |
| timestampz | Timestamp |
| date | Date |
| time | Time |
| timez | Time |
| smalldatetime | Timestamp |

注：在Mybatis操作数据库的过程中，时间类型较为特殊，如下例：

数据库中的字段定义如下:

Java类中的属性定义如下:



需要在Java属性中加入如图所示的@JsonFormat的注解

（5）位类型

|  |  |
| --- | --- |
| 数据库基本类型 | Java类型 |
| bit | Integer |
| bytea | byte[] |

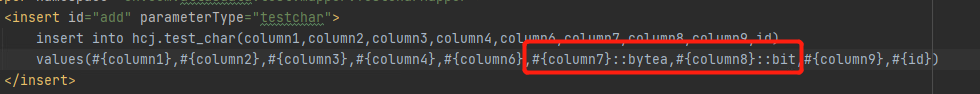
这里同样需要注意的是,百度上以及各资料都显示bit对应的Java类型应当是boolean，理由在于：boolean的true和false映射到数据库时会变化为0，1。但经过实际验证，发现此做法并不可取，无法适配。因此使用Integer,用户在前端传递参数时将参数设置为只能是0或1即可。

而bytea对应的便是byte[]，也就是byte数组。

且这两个类型存在特殊之处,由于在Postgresql语法中，对这两个类型的数据插入操作如下：



也就是强制类型转换。因此，在Mybatis增删改查时，同样需要进行一定处理，以插入数据操作为例：



（6）其它类型

|  |  |
| --- | --- |
| 数据库基本类型 | Java类型 |
| json | Object |
| point | Object |

这两个类型可以直接用Object类型进行处理。

在传入参数时，point类型注意传入的应为：（1.0，2.0)这样的“点”；

而json数据与bit以及bytea同理，需要进行特殊处理



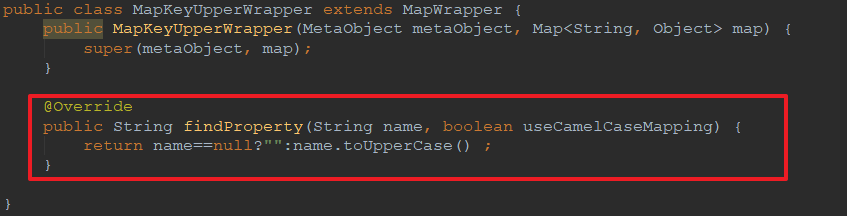
### 结果集封装

#### Map封装

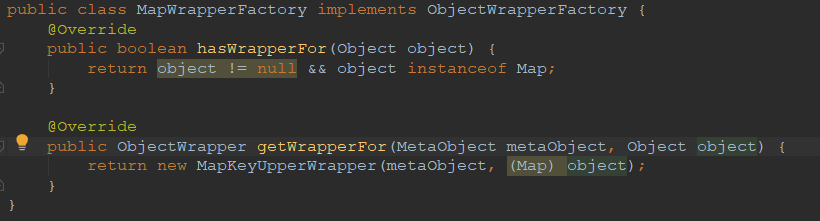
Mybatis在将结果集封装成resultType=“map”的时候，key值为小写，客户应用中在处理map中的数据时，都默认为大写。

解决方案：

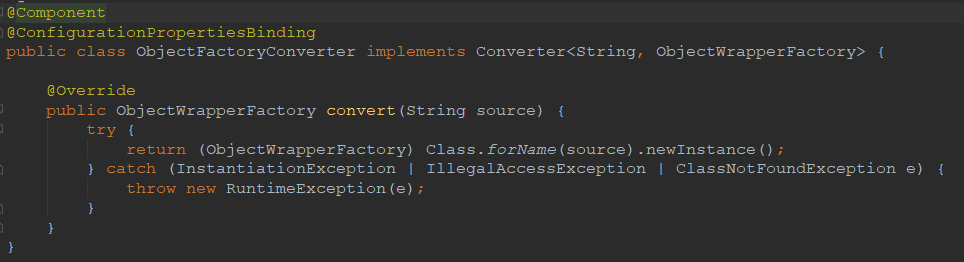
Step1：自定义Map包装器MapKeyUpperWrapper继承MapWrapper,重写findProperty()方法:



Step2：自定义Map包装器工厂类MapWrapperFactory，实现ObjectWrapperFactory接口，并实现接口方法：

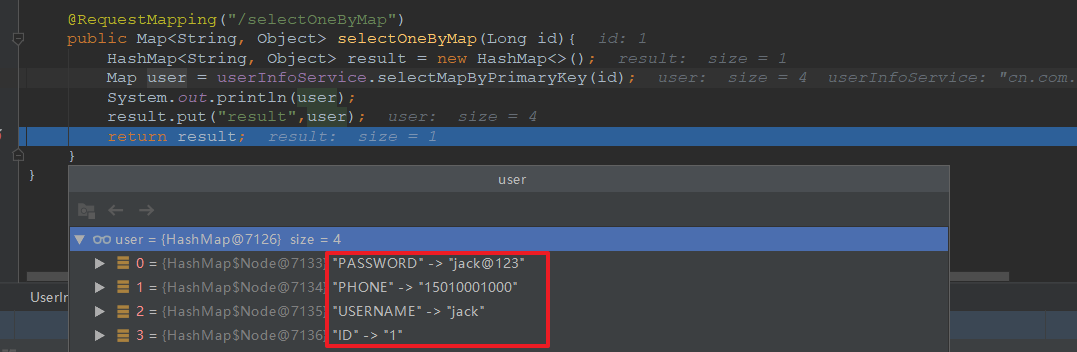


Step3：由于Mybatis与SpringBoot不提供用反射机制来构建对象的converter，但是SpringBoot提供了允许自定义的converter来进行转换，所以这里需要构建一个自定义的converter：



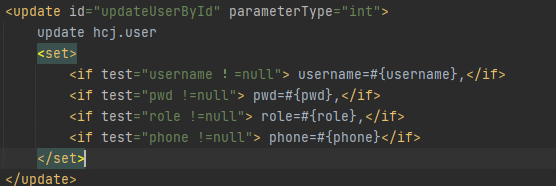
Step4：最后一步，在配置文件中，对自定义的MapWrapperFactory进行配置：





### 动态SQL

在实际应用中，可能存在传进来的参数为一个或者多个的情况，这时候动态sql是一个很好的帮手，通过动态sql，Mybatis便可以根据对应的参数进行相关条件的数据查询等操作。如下简单例子



### 存储过程及函数

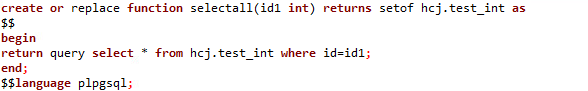
#### Vastbase语法方面

1. 带参数带返回的函数

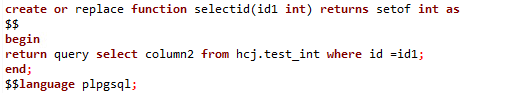
test\_int表：

函数：

返回类型为表时：

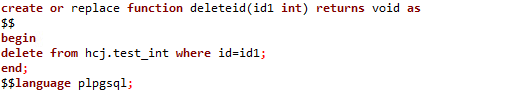


返回类型为字段时：



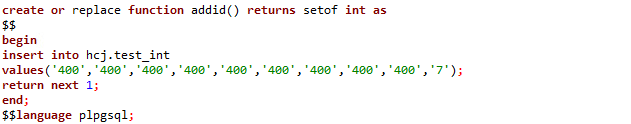
其中test\_int为表名，当我们的函数带有返回类型时，需要使用“returns setof+类型”的形式表明返回类型，且返回语句的语法为return query...，如上图所示。

1. 带参数无返回的函数



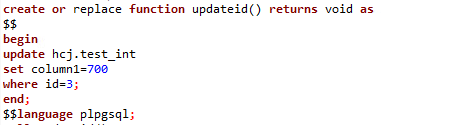
无返回时，与Java语法类似的，我们需要returns void

1. 不带参数带返回的函数



return next 1；即返回一个int类型的数据1

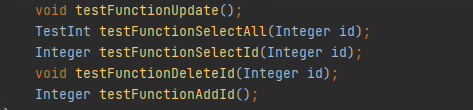
1. 不带参数无返回的函数



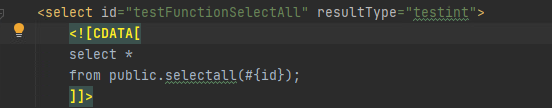
带参数时，即在函数名后加上“参数名 参数类型”,带返回值时则需要注意returns setof +返回类型以及 return query返回语句。不带返回类型时需注意returns void

#### 函数与Mybatis结合使用

首先在TestIntMapper接口中定义相关方法



而后使用即可



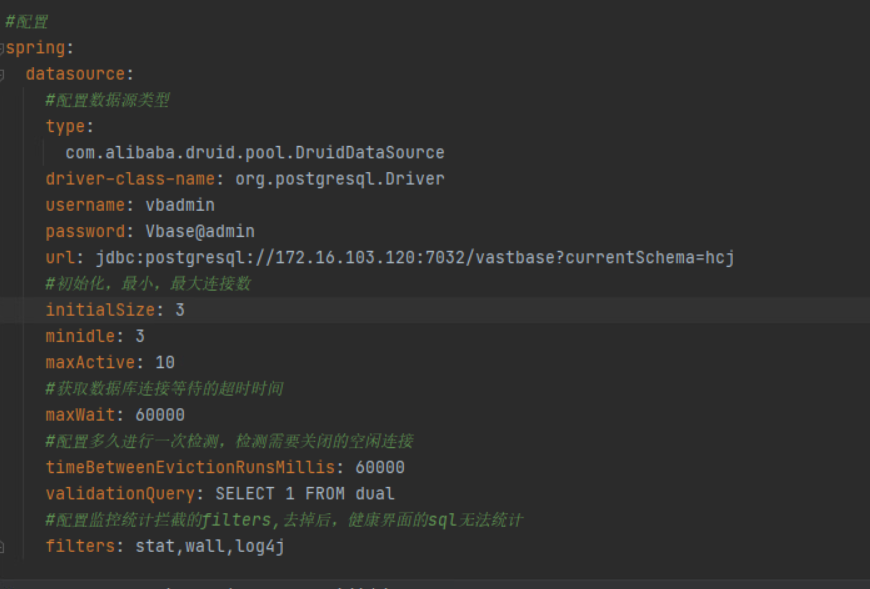
**注：这里的<![CDATA[ ]]>代表执行函数或存储过程**

## MybatisPlus

### MybatisPlus的数据库连接问题

问题产生：在使用代码自动生成器时，报错找不到表格，但实际在数据库中该表格存在

解决方法：在数据库连接的配置中加上currentSchema的配置，如下图，否则在使用代码自动生成器的过程中会默认是找public下的表，自然就会报错。

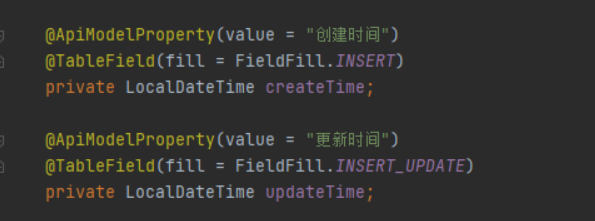


### 设置数据库表中的创建时间字段和更新时间字段的自动插入

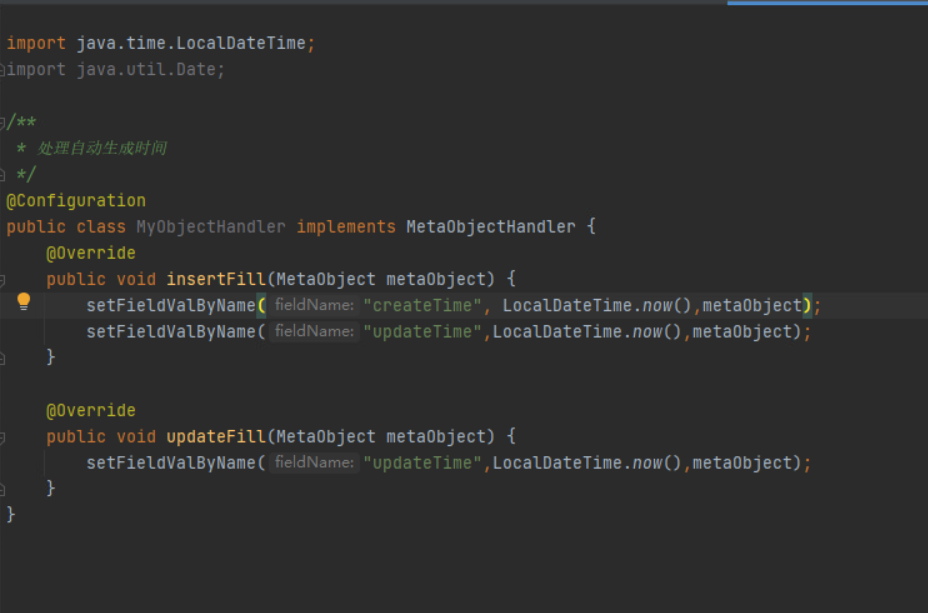
1. 数据库表的字段



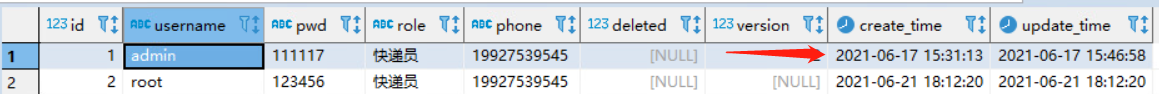
（2）在字段上使用注解



（3）配置自动插入和更新



（4）配置完毕后在对表进行数据的插入或更新时，则会对这两个字段进行进行时间的插入或更新



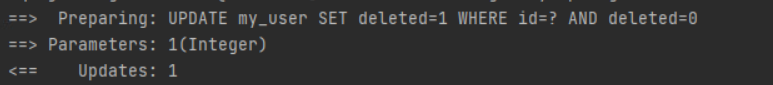
### 乐观锁和逻辑删除字段的配置

#### 逻辑删除字段

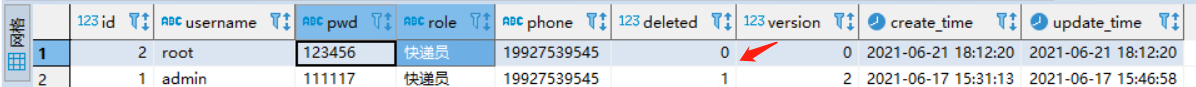
1. 设计思想

即在数据库表中新设置一个字段，这里为deleted。这一字段的设置思想为：当deleted=1时，该条记录对于用户不可见，但其在数据库中依然存在。本质上就是在查询的SQL语句中加上 and deleted=0;如下：

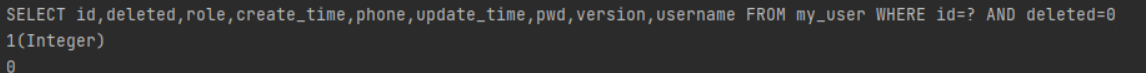
执行delete



查看数据库表，此时数据还在，但是deleted字段变为1

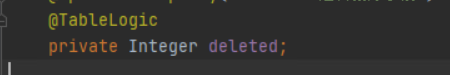


执行查询操作，可以看到SQL后面加上了and deleted=0，查询的结果为空

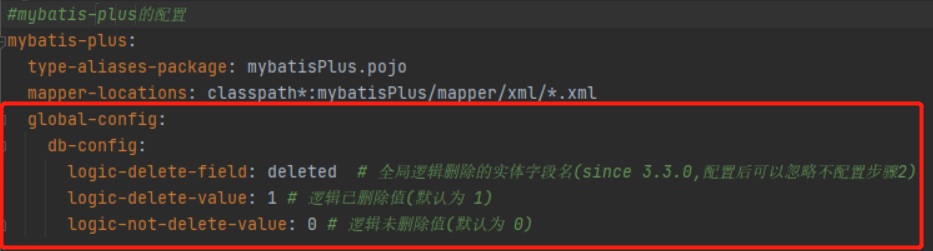


（2）逻辑删除字段的配置

首先在字段上添加注解



到application.yml中配置mybatis-plus的逻辑删除字段

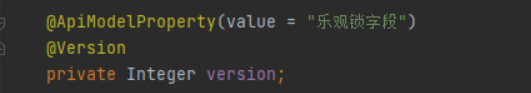


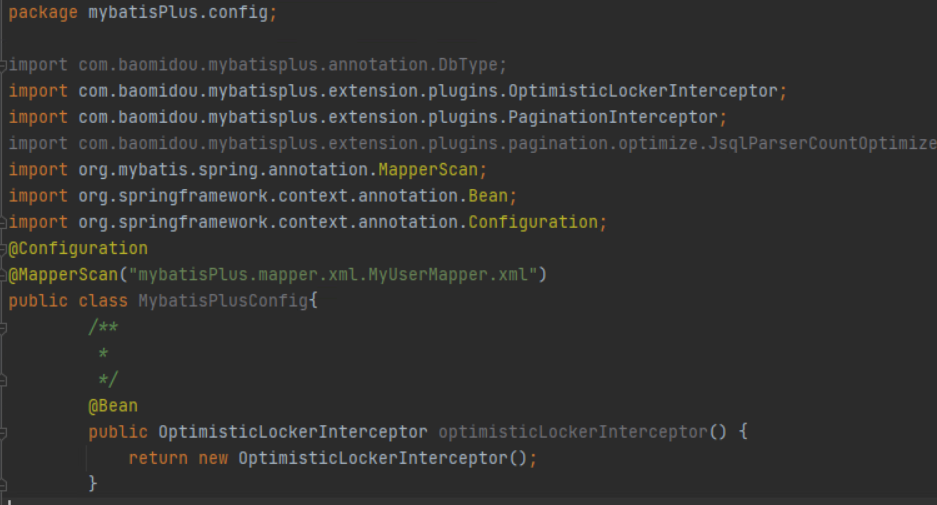
#### 乐观锁字段

乐观锁的设计思路为：新建一个字段version，取出记录时，获取当前的version,更新时带上这个vesion，执行

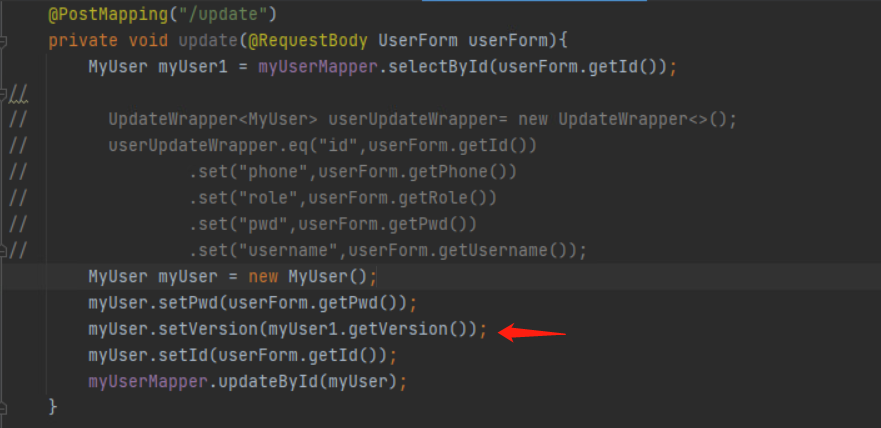
set version=newVersion where version=oldVersion

配置：



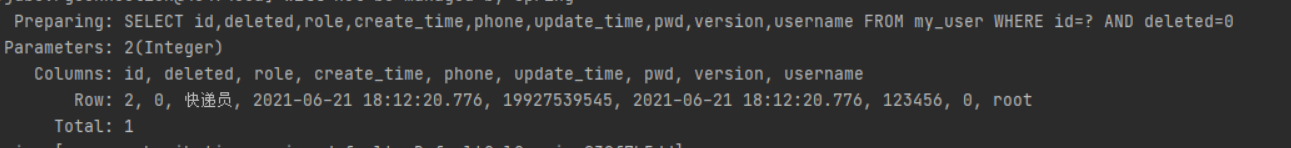


注意在乐观锁使用时，需要先查询出数据，而后更新时带上这个version

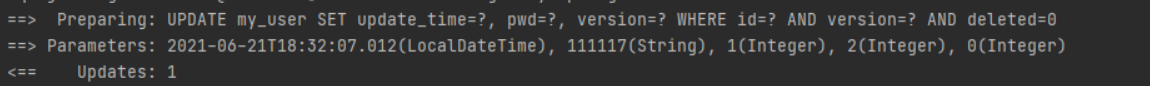


如下例：

Step1:先查询



Step2:后修改



Step3:查看数据库，可以观察到version值更新



### 分页配置（MybatisPlus分页插件的使用）

#### 使用分页的原因

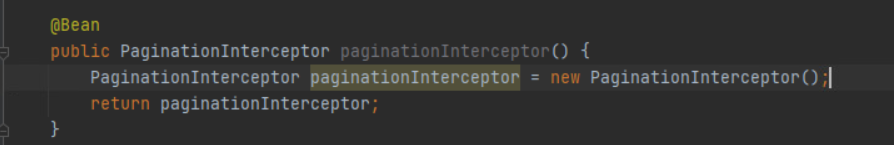
数据库表中的数据可能非常巨大，如果不进行分页处理，一次性查询全部数据可能导致许多问题如查询效率低等，MybatisPlus提供了分页插件

#### 分页插件的本质

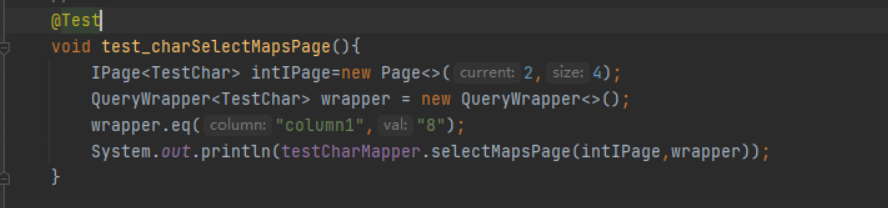
分页插件的本质为在SQL语句后加上limit ? offet ?

#### 分页插件的配置和使用

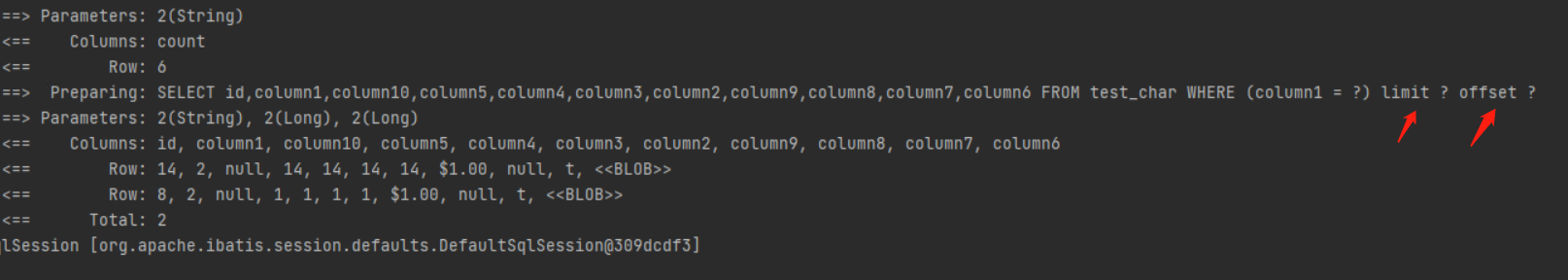
Step1:按照官网配置即可



Step2:而后使用MybatisPlus自身封装的方法即可



Step3:测试，可以看到本质便是加上限制条件

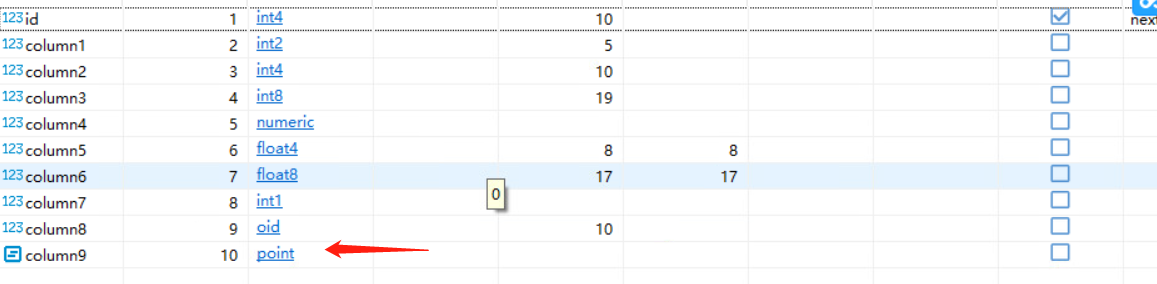


### 特殊类型的字段后端传递参数

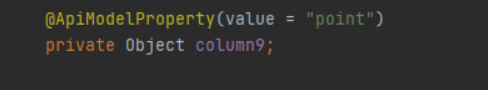
#### 几何类型point

数据库中字段类型为point，Java中使用Object作为其类型，在后端传参时，使用PGpoint进行传参。

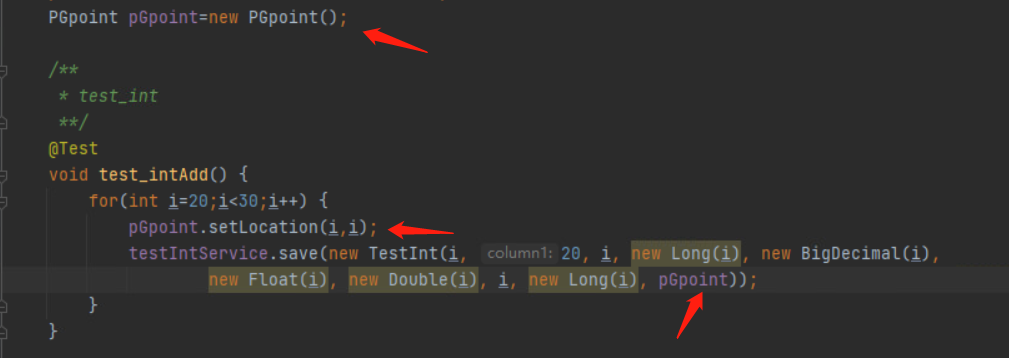
数据库中的point



Java中使用Object



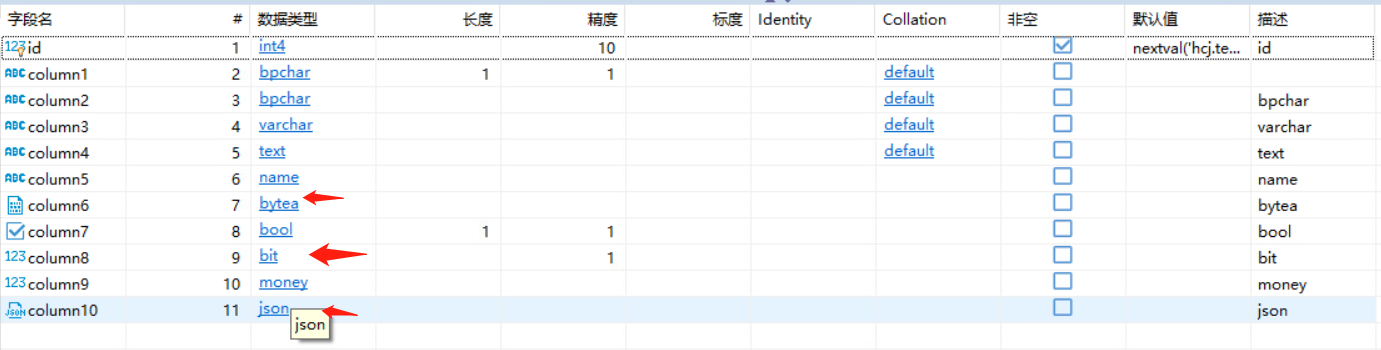
后端传递，使用PGpoint



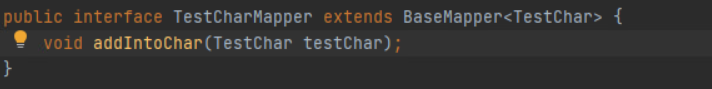
#### Json类型、bit类型以及bytea类型

这三个类型都无法使用MybatisPlus封装的方法进行插入操作，因此需要我们自定义插入的方法，编写SQL语句，如下：

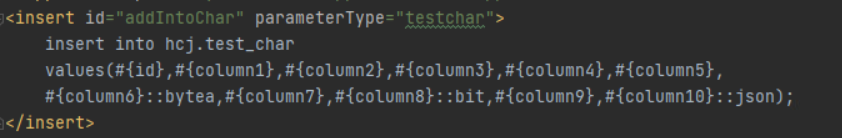
数据库表test\_char里面的字段json、bytea、bit



在Mapper中定义函数



Mapper.xml实现



即类型转换，也可通过调用函数的方式实现上述操作

其它类型与Mybatis的操作一致

### MybatisPlus封装的方法

#### Mapper crud接口

在Vastbase数据库中，各种基本数据类型对于Mapper的各种crud接口都是适配的。其中，一些特殊字段的处理如3.3.5所示。

#### Service crud接口

在Vasebase数据库中，各种基本数据类型对于Service的各种crud接口都是适配的。其中，一些特殊字段的处理如3.3.5所示。

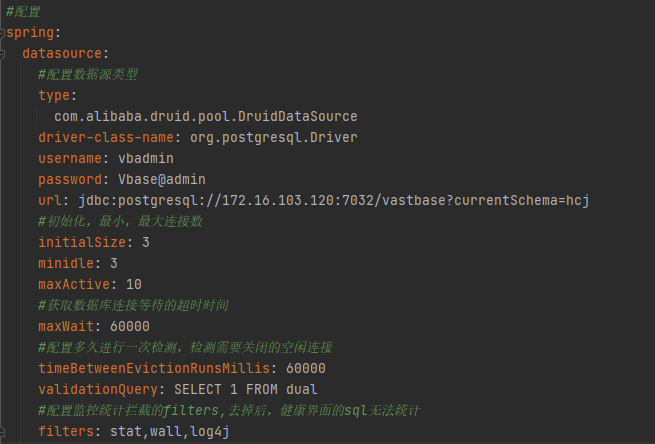
### 函数和存储过程

MybatisPlus封装的方法并无法解决函数和存储过程的问题，因此对于函数和存储过程，操作与Mybatis一致

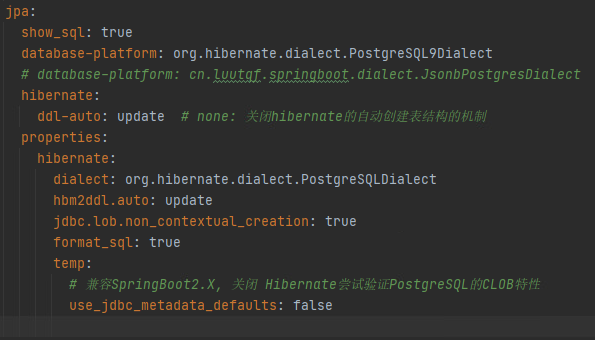
## Hibernate

### Hibernate配置数据库的连接

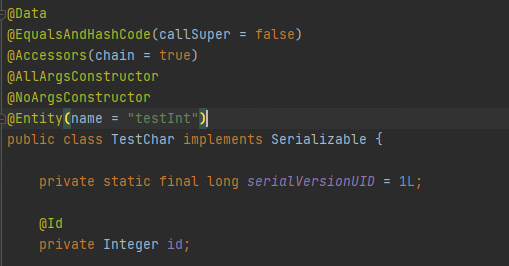
（1）配置数据库连接，这部分与Mybatis和MybatisPlus一致，同样也需要Vastbase的依赖



1. 配置jpa



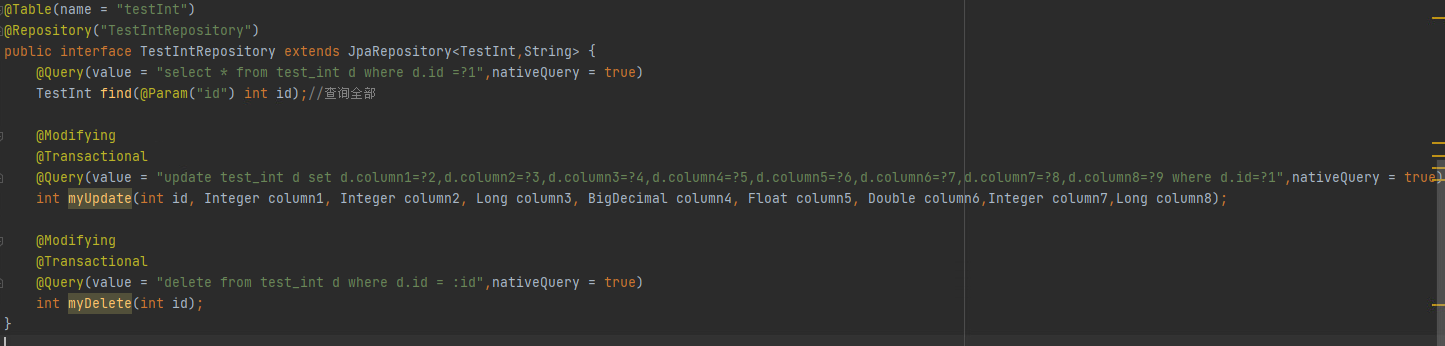
### 实体类需要注意的事项



注意点1：实体类上需要有注解@Entity

注意点2：id字段需要有注解@Id

### 持久层Repository需要注意的事项



注意点1：类上需要有对应实体类注解如@Table(name=“testInt”)

注意点2：持久层注解标志@Repository(“TestIntRepository”)

注意点3：持久类继承JpaRepository<Object,String>，将Object更改为对应实体类

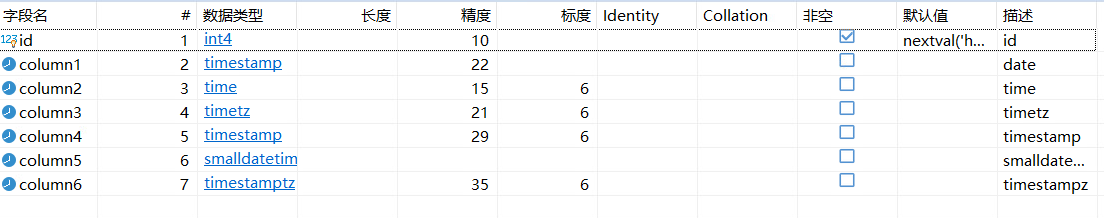
注意点4：Hibernate执行默认是HQL语句，而使用nativeQuery=true则将其转变为SQL语句语法

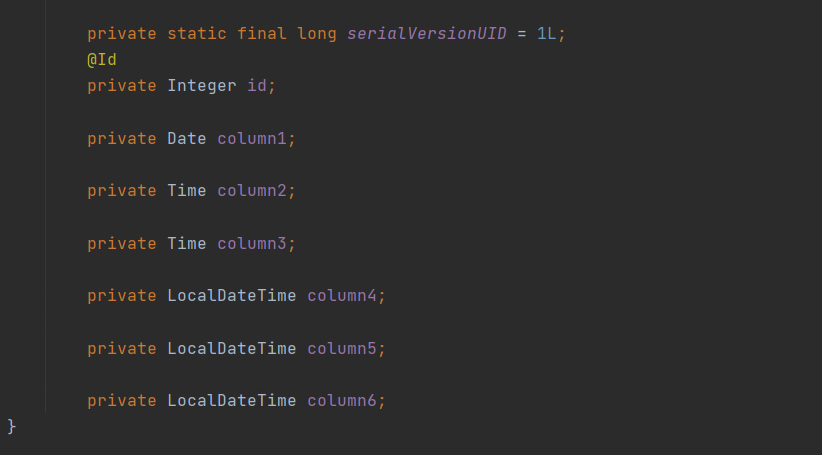
注意点5：更新删除等操作时需要加上注解@Transactional，执行事务操作

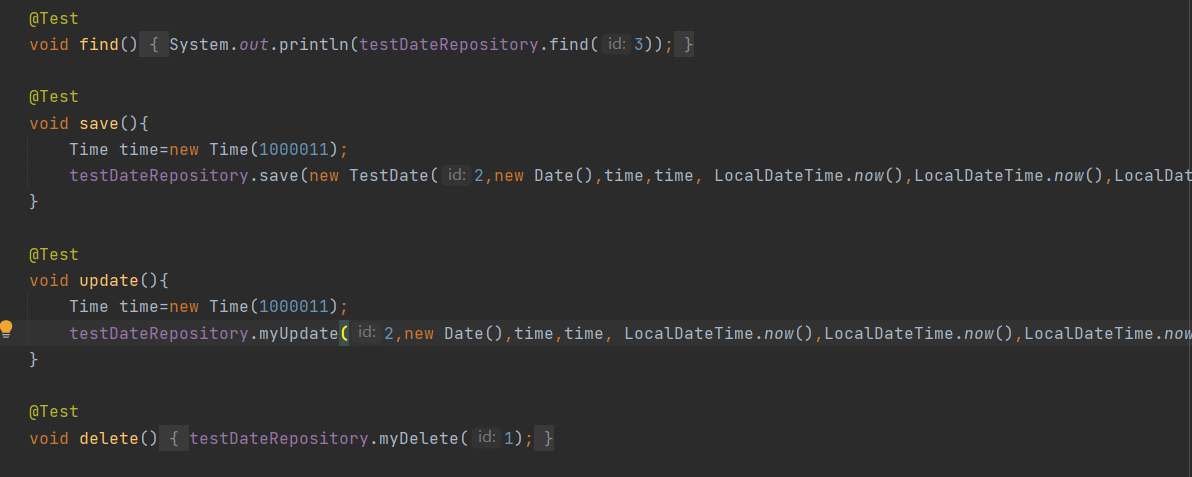
注意点6：@Modifying+@Query 定义个性化更新操作

### 各种基本类型的操作

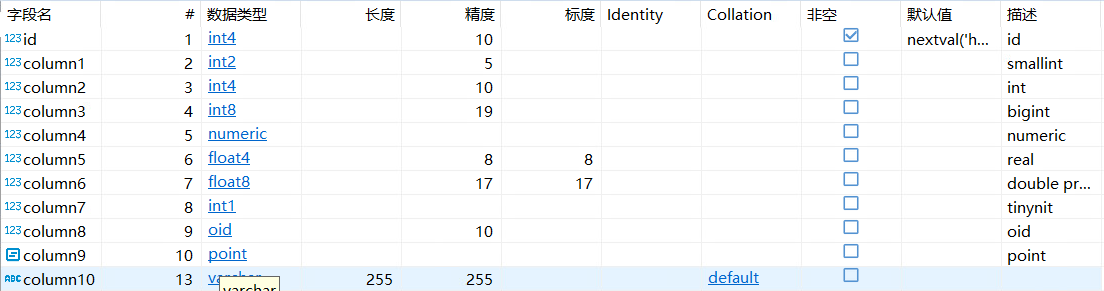
#### 时间类型

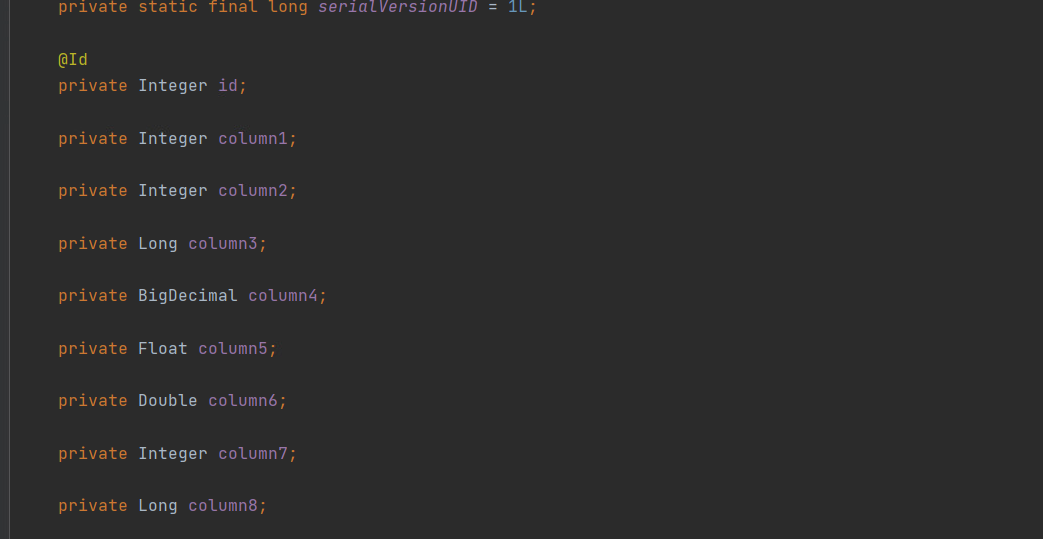




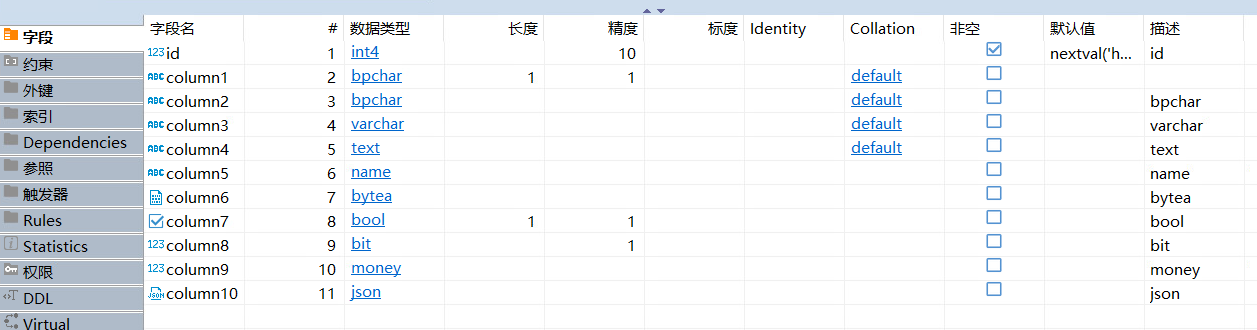


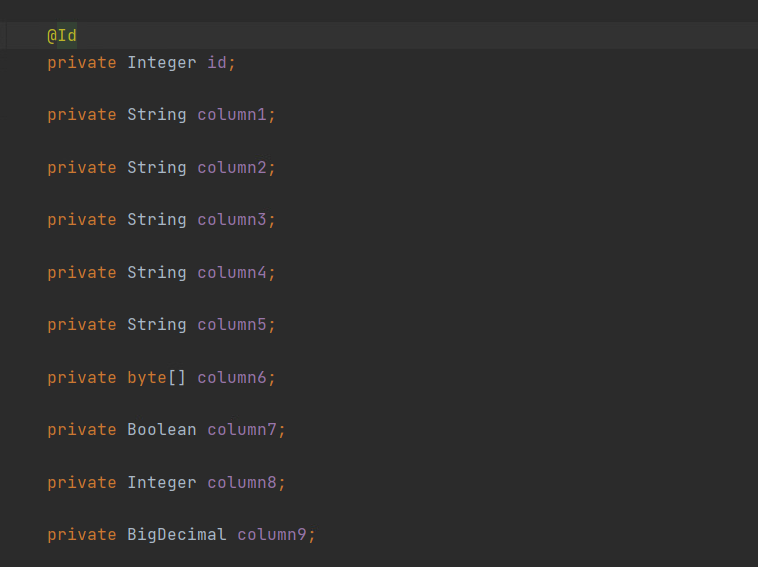
#### 整型





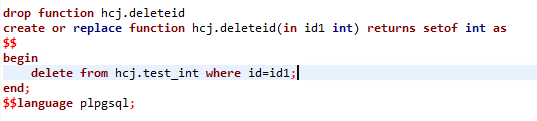
#### 字符类型





### Hibernate操作存储过程

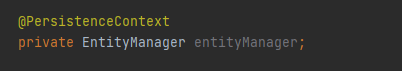
（1）首先我们需要在数据库定义一个函数

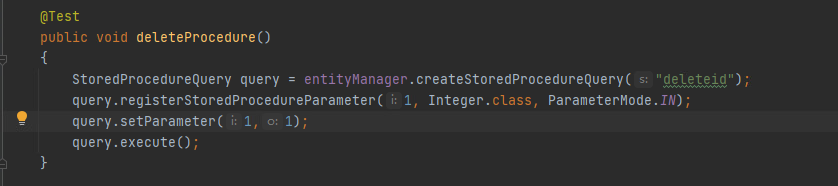


注意点1：将该函数定义到对应的模式下，否则后续会报错找不到函数

注意点2：在函数的参数中，应该加上in,如 in id1 int，表明这是一个徐娅传入的参数

（2）而后在程序中调用





步骤1：定义EntityManager

步骤2：由EntityManager创建查询

步骤3：查询的参数传递应先“注册”，即如图所示，第一个参数为Integer类型且是一个传入的参数

步骤4：设置参数，而后执行即可

注：若需要查看参数返回值等，需要在query.execute()之前，如下：

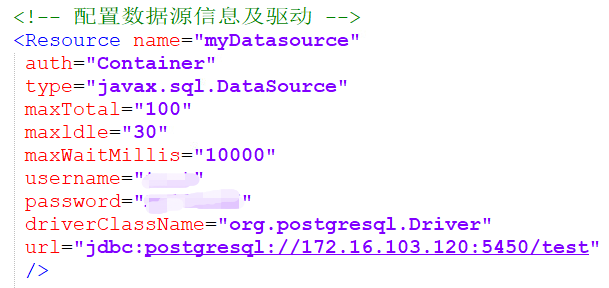
# 中间件适配

## Tomcat

Step1:在本地安装Tomcat 7.0.109

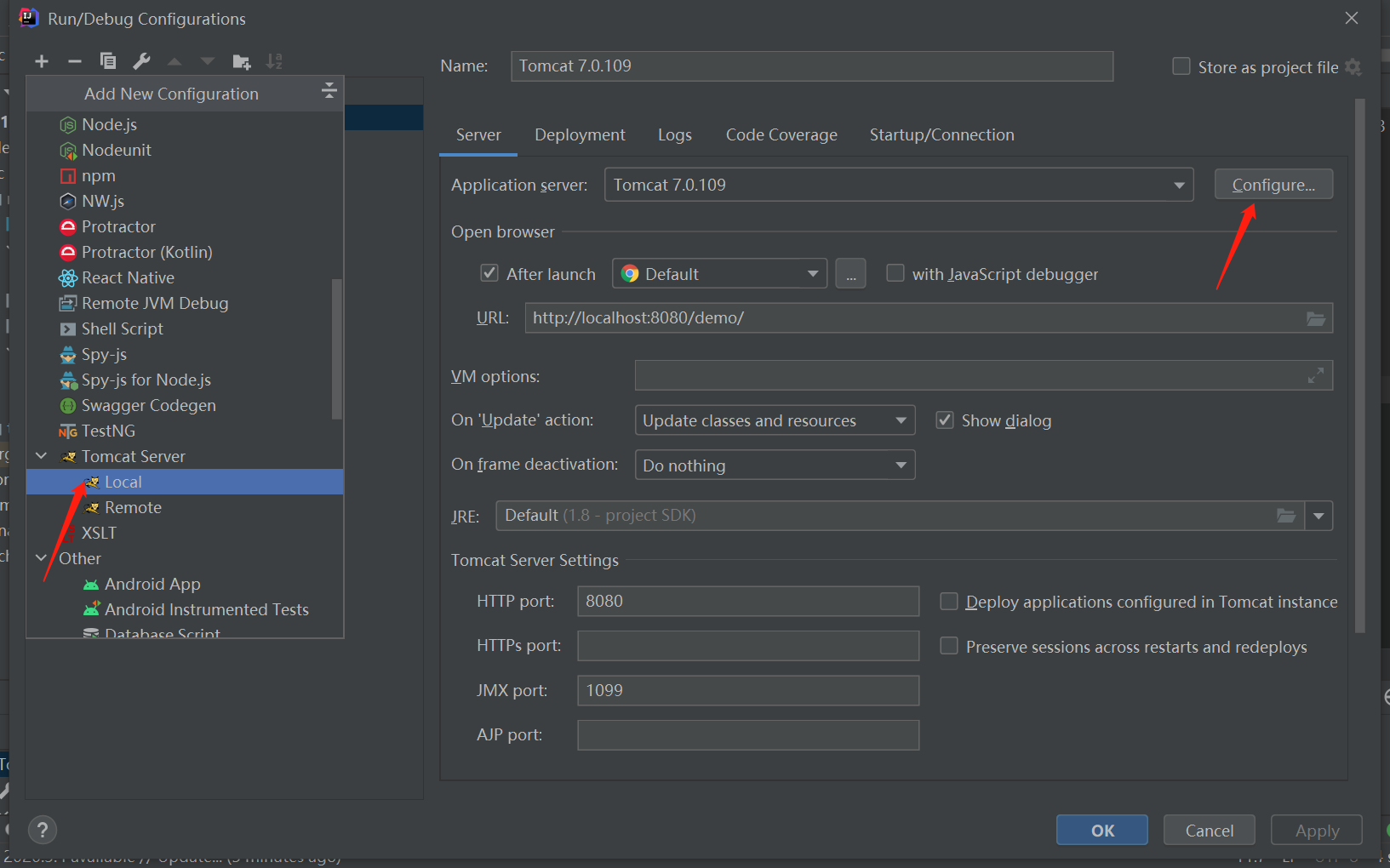
Step2:将vastbase驱动包vastbase-2.0.jar拷贝进apache-tomcat-7.0.109\lib

Step3:在Tomcat中配置数据源和驱动

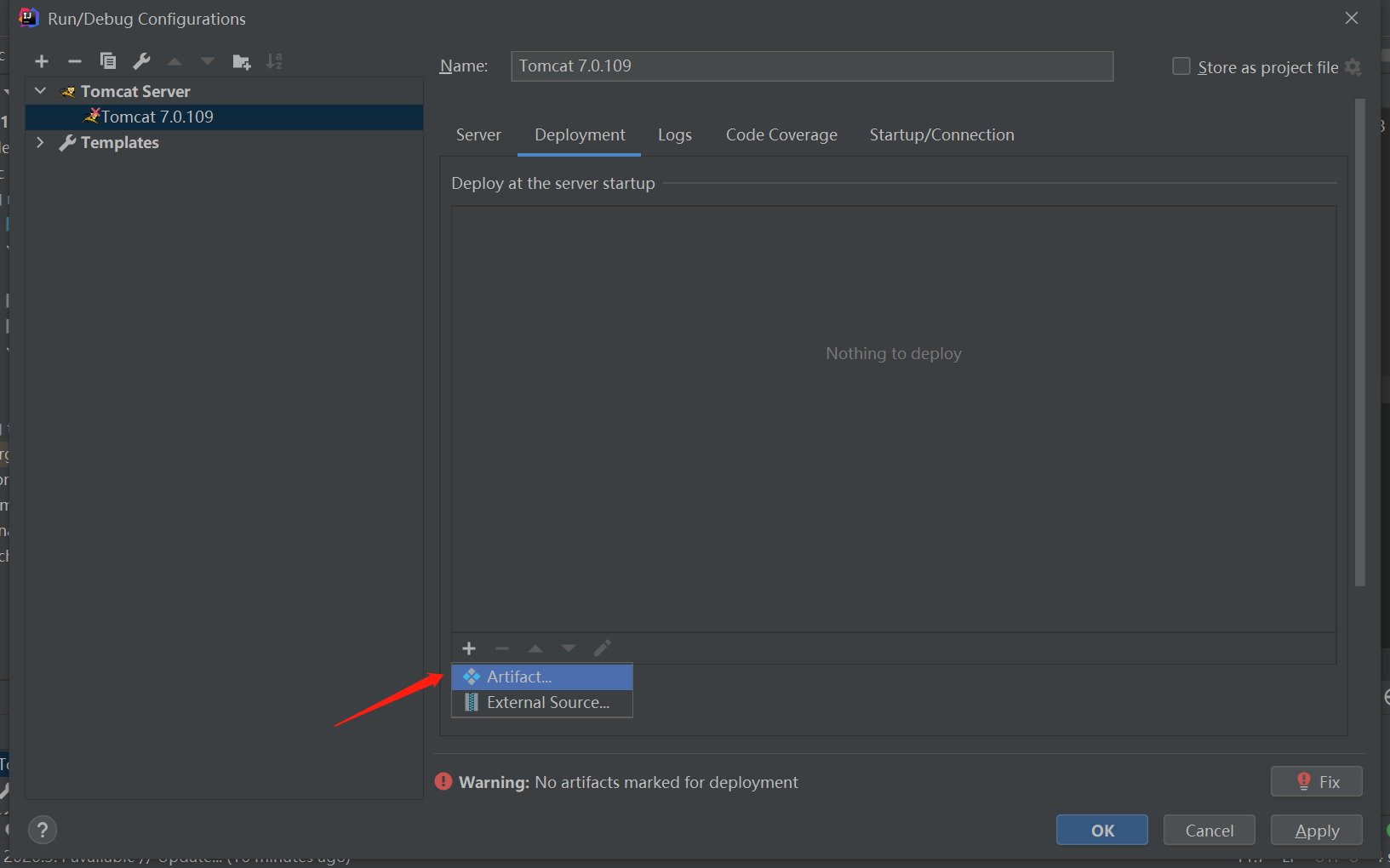
打开apache-tomcat-7.0.109\conf\context.xml做如下配置：

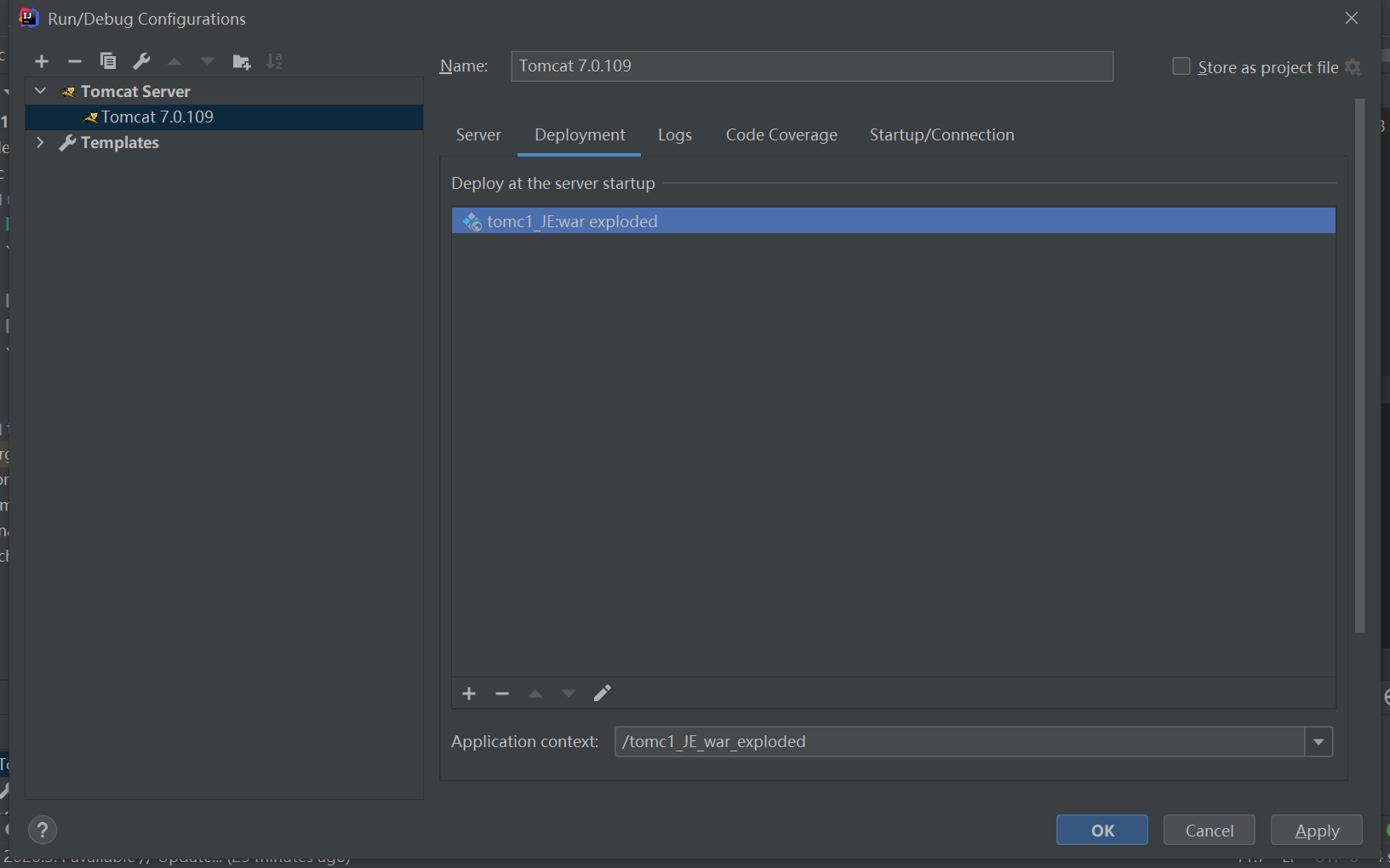
配置Tomcat的数据源信息及驱动，其中name为自定义的数据源名称

Step4:创建web项目，指定本地Tomcat7.0.109



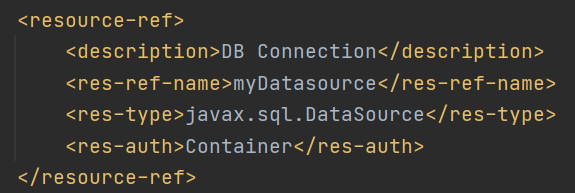
在Configuration中添加Local Tomcat Server，Configure选择本地安装的Tomcat7.0.109





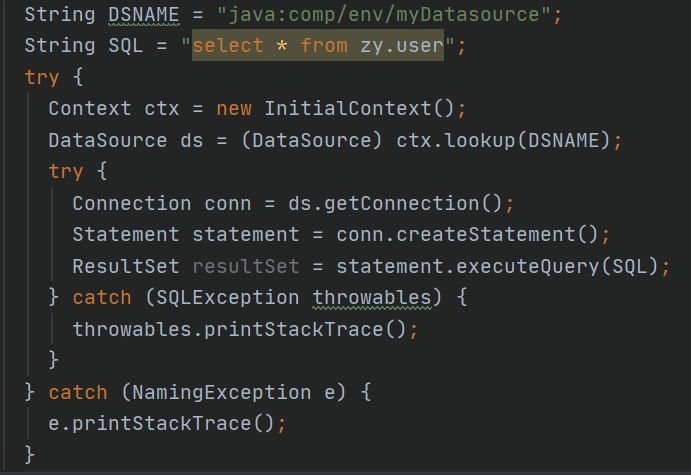
部署项目

Step5:在web.xml中配置数据源



其中<res-ref-name>标签对应的是在context.xml中配置的数据源名称

Step6:在JSP中使用数据库连接池



其中DSNAME中“java:comp/env/myDatasource”是固定写法，表示得到配置环境

## ShardingSphere

### ShardingSphere-JDBC

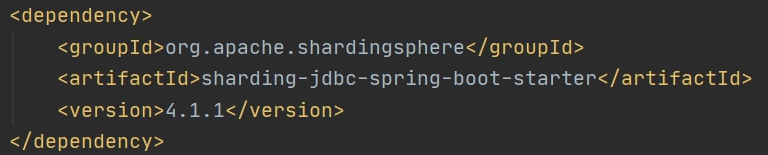
引入vastbase的maven依赖



#### 数据分片

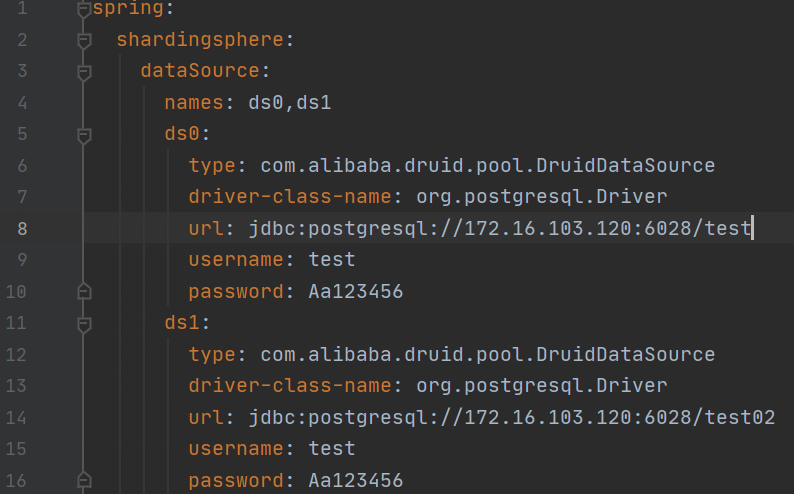
Step1:准备好分片后的数据库

Step2:引入ShardingSphere的maven依赖

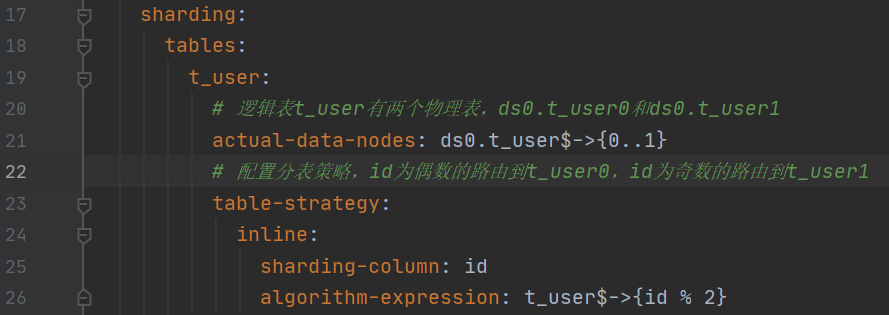


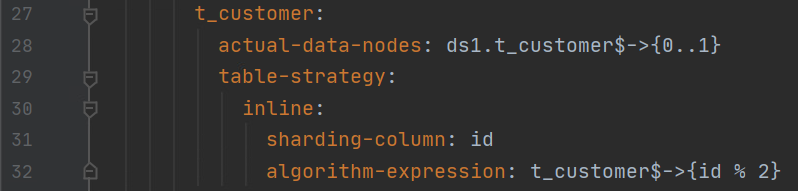
Step3:配置shardingsphere

1.配置多数据源信息



2.配置分片策略



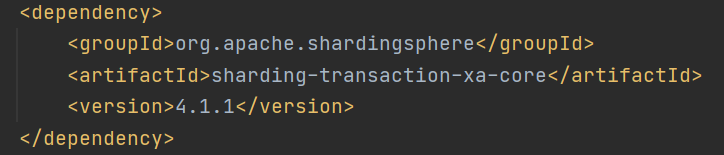


相关数据的SQL操作便会被路由到所配置的数据源分片节点上

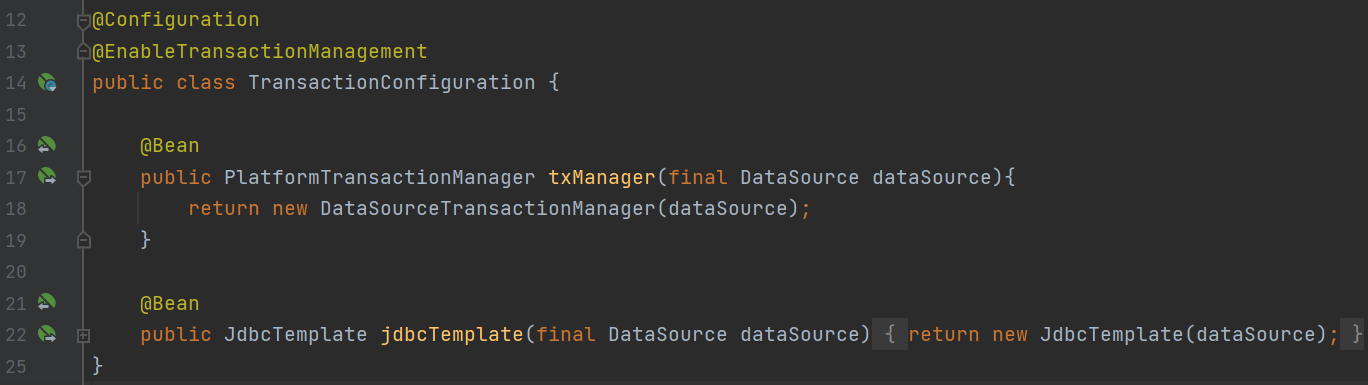
#### 分布式事务

以XA事务为例

Step1:导入maven依赖



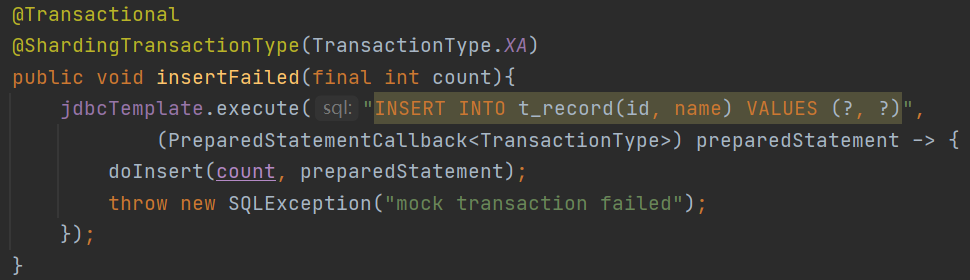
Step2:编写配置类，向Spring容器中注入PlatformTransactionManager和JdbcTemplate



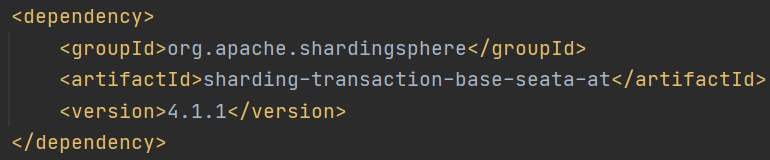
Step3:编写业务类，实现数据库操作并使用业务

ShardingSphere分布式事务注解@ShardingTransactionType要和Spring事务注解@Transactional一并使用才会生效；

doInsert方法向两个数据库中插入数据，之后手动抛出异常，在此之前对两个数据插入的数据都会回滚。

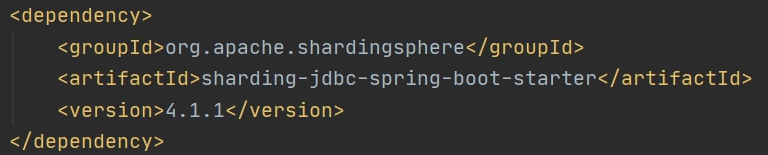


若要改成使用BASE事务，需要先启动seata服务器并导入Sharding-BASE依赖



#### 读写分离

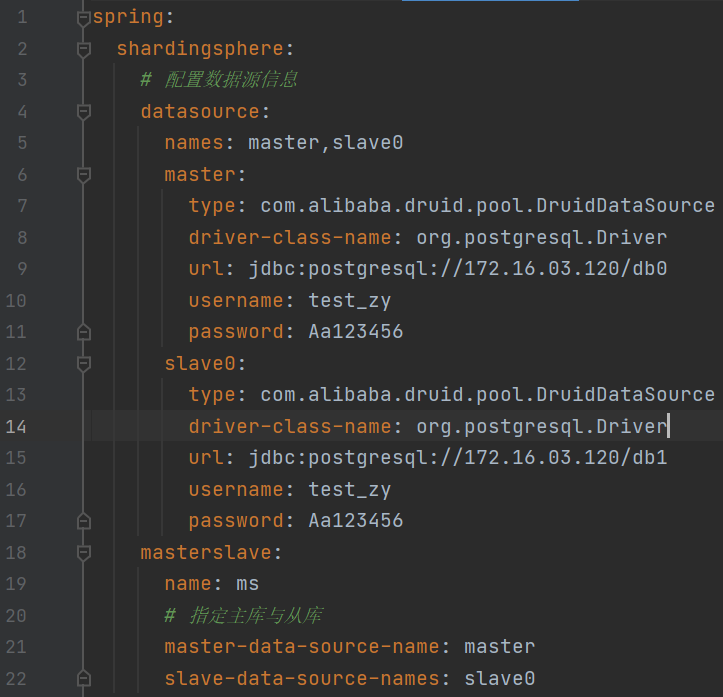
Step1:导入maven依赖



Step2:配置数据源

配置多数据源，指定主库与从库。4.1.1版本支持单主库多从库，不支持主从库之间的数据同步。

当一个线程中没有进行主库的修改，读操作会被路由到从库进行查询；若一个线程中主库进行了修改，为了保持数据一致性，读操作也会被路由到主库查询。



### ShardingSphere-Proxy

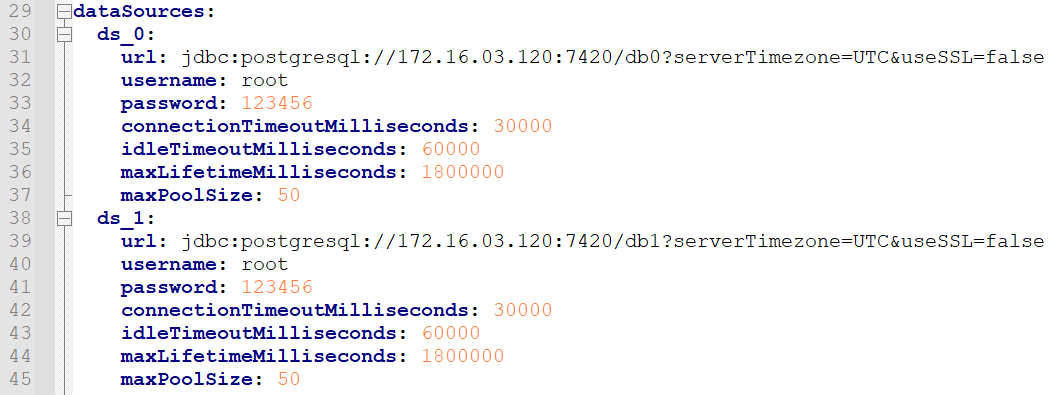
首先需要下载并解压apache-shardingsphere-4.1.1-sharding-proxy-bin.tar

由于ShardingSphere-Proxy默认只支持postgresql，使用其他数据库时要先将对应java驱动包加入lib目录，因此将vastbase-2.0驱动包加入lib目录。

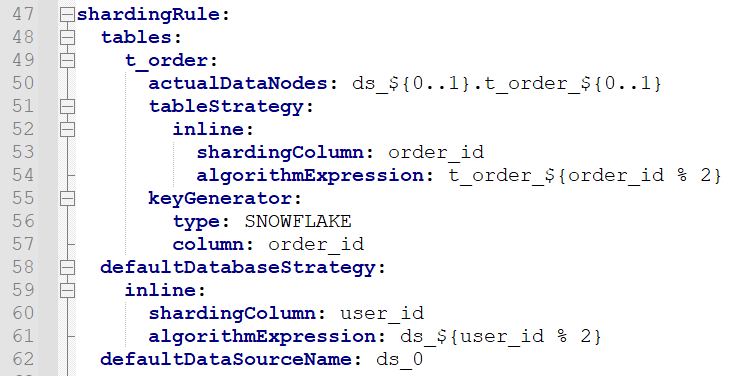
#### 数据分片

在/conf/config-sharding.yaml配置数据源及数据分片策略

Step1:配置数据源



Step2:配置分片策略



Step3:配置server.yaml

在/conf/server.yaml配置代理数据库的用户、密码及逻辑数据库权限

Step4:启动shardingsphere-proxy服务

windows运行：/bin/start.bat

linux运行：/bin/start.sh

Step5:使用代理数据库

使用任何PostgreSQL客户端，与代理数据库建立连接，即可当作普通数据库来使用

用户及密码：server.yaml中的配置

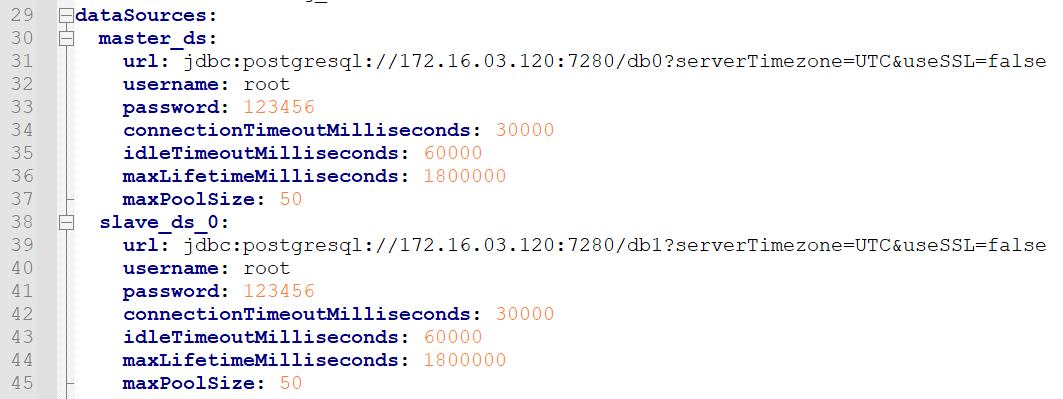
Host：127.0.0.1

Port：3307

#### 读写分离

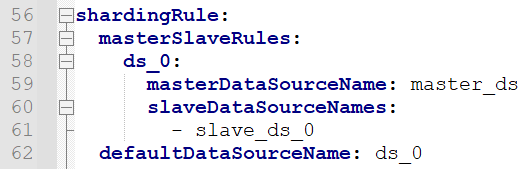
在/conf/config-master\_slave.yaml配置数据源及读写分离主从库

Step1:配置数据源



Step2:配置主从库

指定主库与从库，并将主-从数据库指定为默认数据库



Step3:配置server.yaml

在/conf/server.yaml配置代理数据库的用户、密码及逻辑数据库权限

Step4:启动shardingsphere-proxy服务

windows运行：/bin/start.bat

linux运行：/bin/start.sh

Step5:使用代理数据库

使用任何PostgreSQL客户端，与代理数据库建立连接，即可当作普通数据库来使用

用户及密码：server.yaml中的配置

Host：127.0.0.1

Port：3307



电话：010-82838118

地址：北京市海淀区学院路30号科大天工大厦B座6层

官网：www.vastdata.com.cn