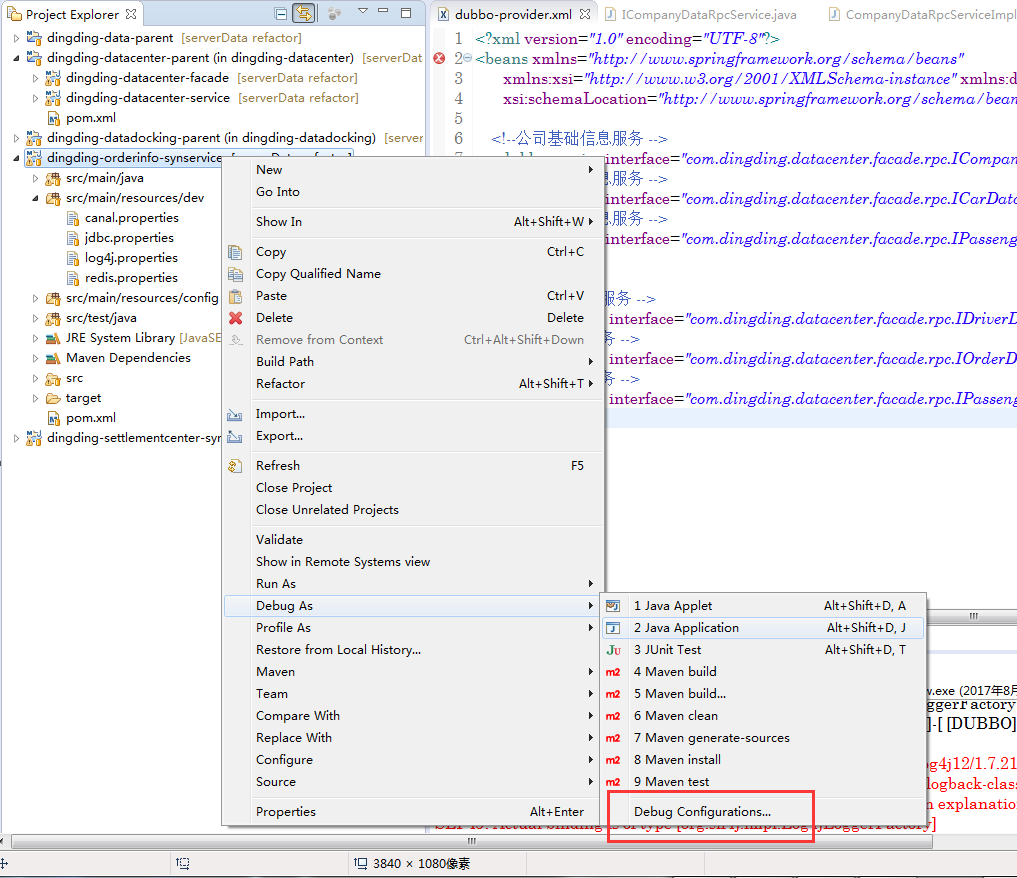
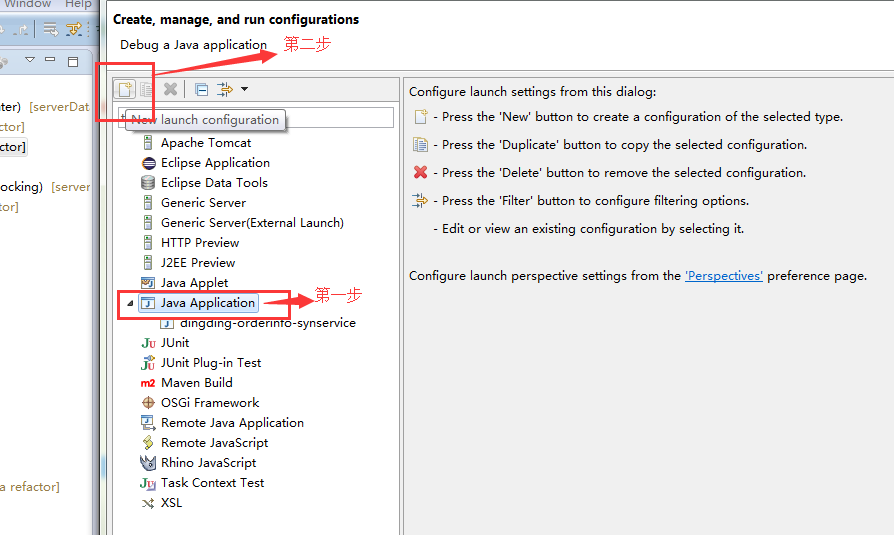
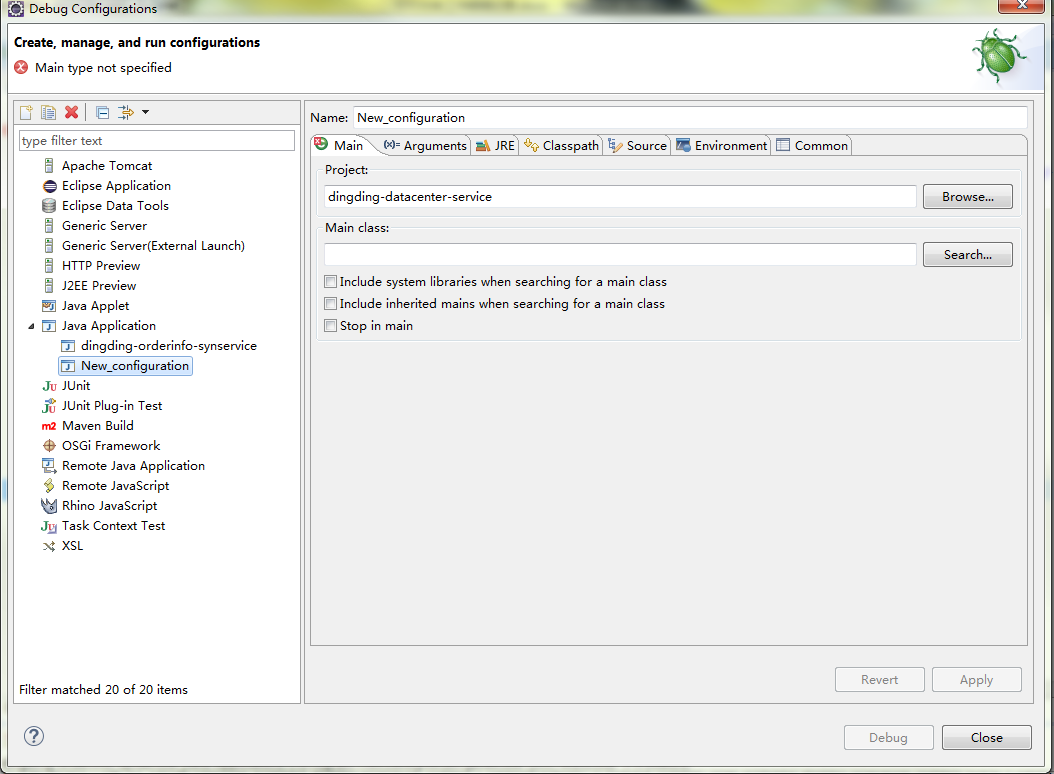
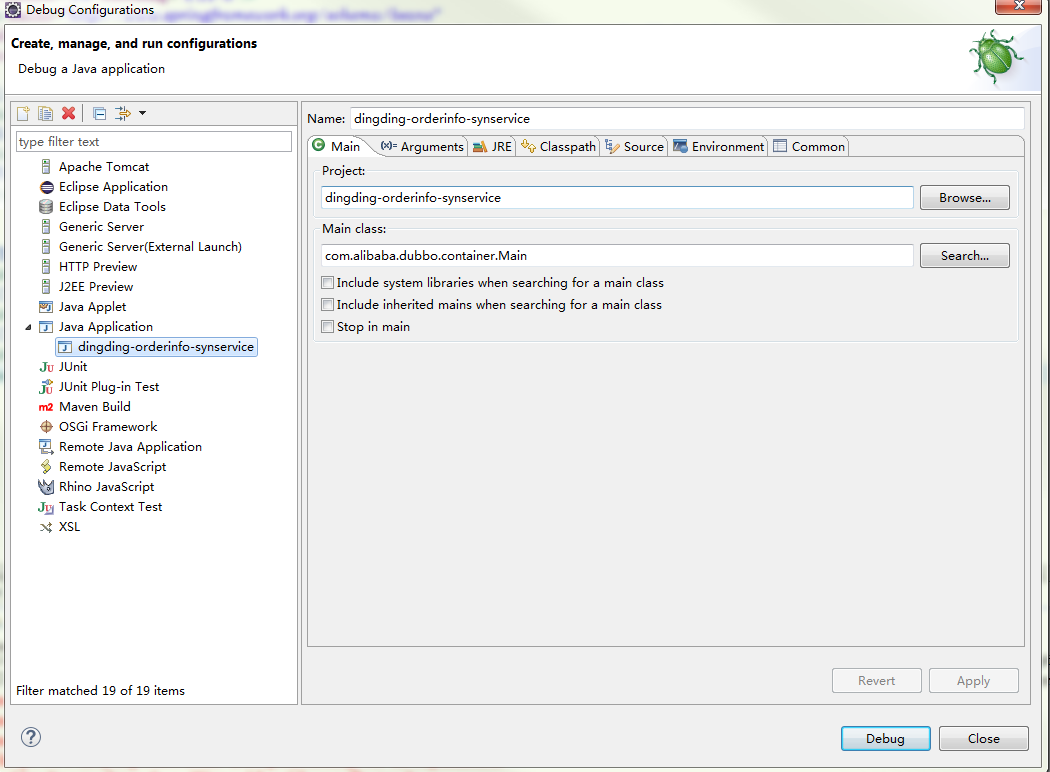
1 dubbo 项目启动方式





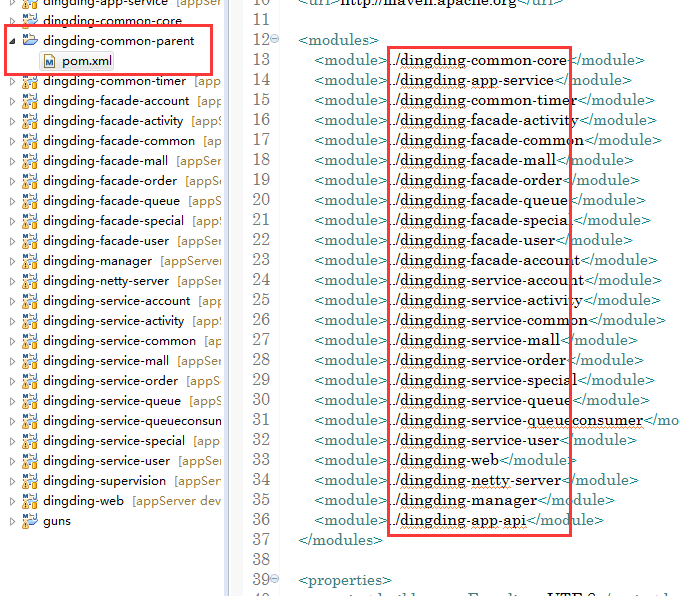


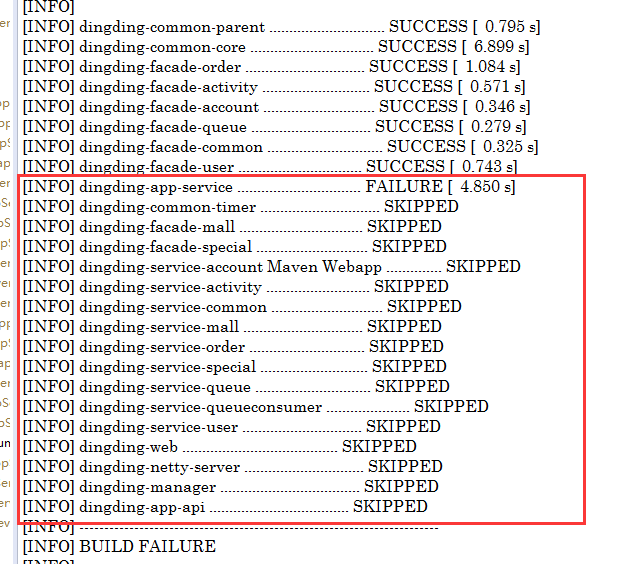


2 项目启动顺序

2.1 appService源码组构建顺序

构建此处即可全部生成项目依赖到本地仓库。



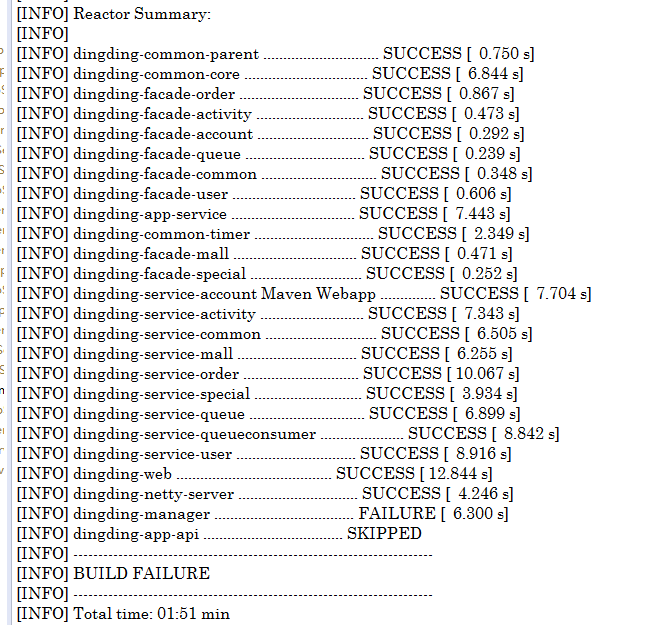
 出现FAILURE(failure失败)和SKIPPED(skipped)的情况后，需要先构建dingdingService源码组，然后再次编译。

2.2 dingdingService源码组构建顺序

A com-dingding-springboot

B dingding-parent

2.3 最终编译完成



dingding-manager 是web项目，此处构建失败。

dingding-app-api 是web项目，此处构建跳过。

3 serviceData 源码组

3.1 所监控的表

[

"dingding\_order.vip\_order",

"dingding\_order.vip\_order\_fee",

"dingding\_order.others\_call\_car",

"dingding\_settlement\_center.vip\_order\_settlement\_bill",

"dingding\_dictionary.car\_insurance\_info",

"dingding\_base\_info.activity",

"dingding\_base\_info.activity\_join\_coupon",

"dingding\_base\_info.activity\_join\_rule",

"dingding\_base\_info.activity\_rule",

"dingding\_base\_info.coupon\_templet",

"dingding\_base\_info.coupon\_grant\_record",

"dingding\_base\_info.order\_channel\_join\_city",

"dingding\_base\_info.charge\_rule\_city",

"dingding\_order.charge\_model",

"dingdingdb.userdetail",

"dingdingdb.userlogin",

"dingdingdb.dd\_equipment\_login\_info",

"dingdingdb.specialuserposition",

"dingdingdb.specialuservehicleinfo",

"dingdingdb.specialuserlogin",

"dingdingdb.specialuserinfo",

"dingdingdb.dd\_special\_user\_auth\_info",

"dingdingdb.specialuserloginrecord",

"dingdingdb.dd\_organization",

"dingdingdb.driver\_punish",

"dingdingdb.driver\_train",

"dingdingdb.specialordercomment",

"dingdingdb.passenger\_complaint",

"dingdingtestdb.userdetail",

"dingdingtestdb.userlogin",

"dingdingtestdb.dd\_equipment\_login\_info",

"dingdingtestdb.specialuserposition",

"dingdingtestdb.specialuservehicleinfo",

"dingdingtestdb.specialuserlogin",

"dingdingtestdb.specialuserinfo",

"dingdingtestdb.dd\_special\_user\_auth\_info",

"dingdingtestdb.specialuserloginrecord",

"dingdingtestdb.dd\_organization",

"dingdingtestdb.driver\_punish",

"dingdingtestdb.driver\_train",

"dingdingtestdb.specialordercomment",

"dingdingtestdb.passenger\_complaint",

"dingdingtestdb.dd\_city\_dict",

"dingdingdb.dd\_city\_dict",

"dingding\_base\_info.dd\_busi\_lice\_info",

"dingding\_base\_info.dd\_comp\_base\_info",

"dingding\_base\_info.dd\_comp\_operate\_scale",

"dingding\_base\_info.dd\_comp\_pay\_info\_inte",

"dingding\_base\_info.dd\_serv\_inst\_info",

"dingding\_dictionary.dictionary\_city\_info"

]

3.2 canal部署与配置信息

线上

开发

测试

3.3 otter部署与配置信息

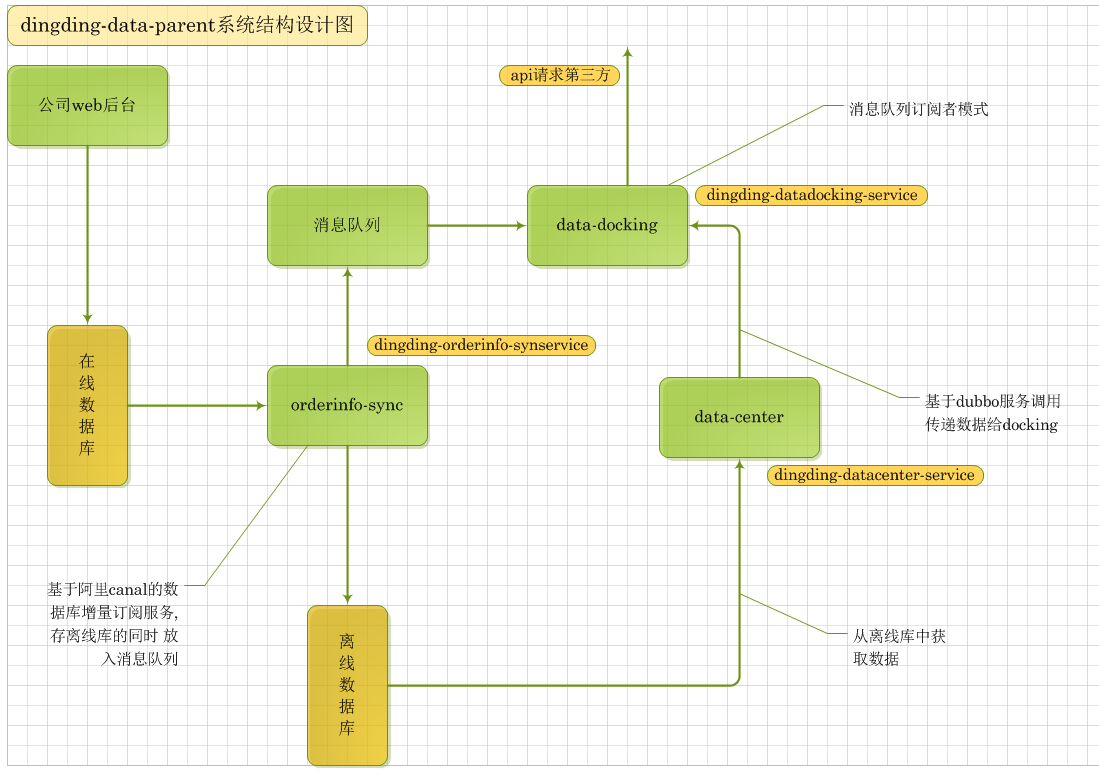
线上

开发

测试

3.4 项目技术与业务逻辑

整体项目结构



serviceData源码组包括如下3个项目：

dingding-orderinfo-synservice 对应图中的【orderinfo-sync】

dingding-datadocking-parent对应图中的【data-docking】(maven父子结构)

---- dingding-datadocking-façade

---- dingding-datadocking-service

dingding-datacenter-parent对应图中的【data-center】(maven父子结构)

---- dingding-datacenter-facade

---- dingding-datacenter-service

原项目组中dingding-settlementcenter-synservice 废弃。

3个java应用服务器

A 消息队列服务器

B 阿里巴巴canal 服务器

canal服务器会将所监控的表(见3.1)中的数据，全部同步到dingding\_data库中，这是一个特别大的离线数据库，为后期做数据分析、大数据统计等等使用。

类型为多对一服务，多个表数据插入到一个表中。

C 阿里巴巴 otter 服务器

otter服务器是直接使用页面来配置所要监控的表的。无需代码开发，为一对一服务，他所同步的表数据会通过otter服务器直接进入离线库。首先你需要启动otter服务器。

dingding-orderinfo-synservice

该项目的入口为com.dingding.canal.common.main.CoreMain.java。这个项目不是以dubbo的方式来启动，作者模仿了dubbo的方式写的主类，运行main方法即可启动。

这个类是canal服务器的客户端。为多对一服务，多个表数据插入到一个表中。

dingding-datadocking-service

dingding-datacenter-service

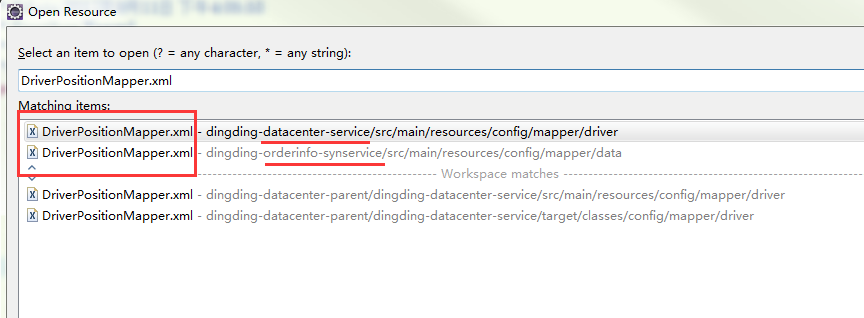
4 数据库表用途

4.1 dingding\_data

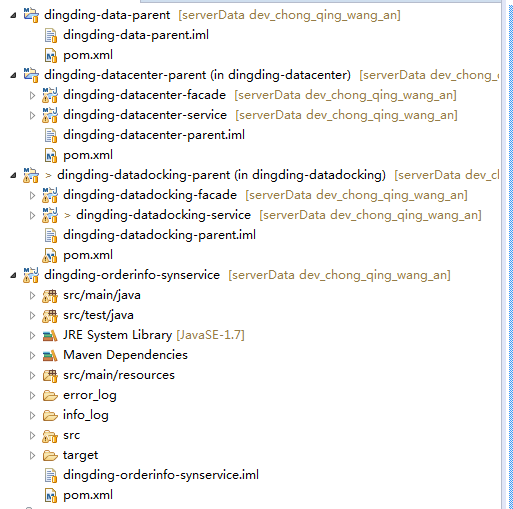
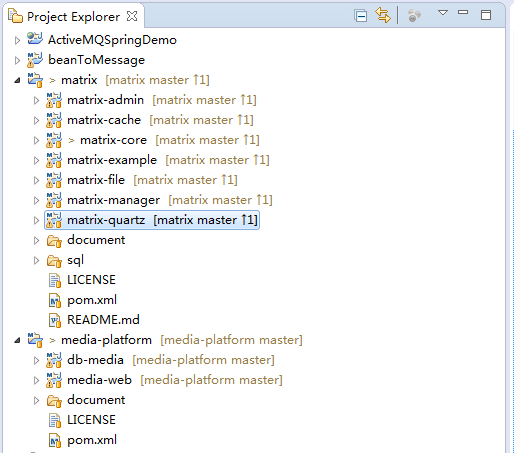
dictionary\_city\_info 城市信息，包含城市的id，如：北京-10，重庆-23

项目2233

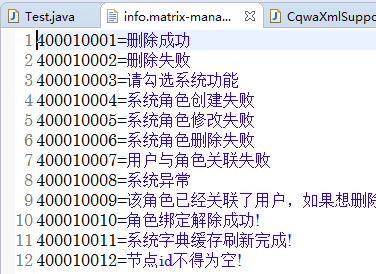
1 数据库业务表没有实现分离，各个项目中都可能有



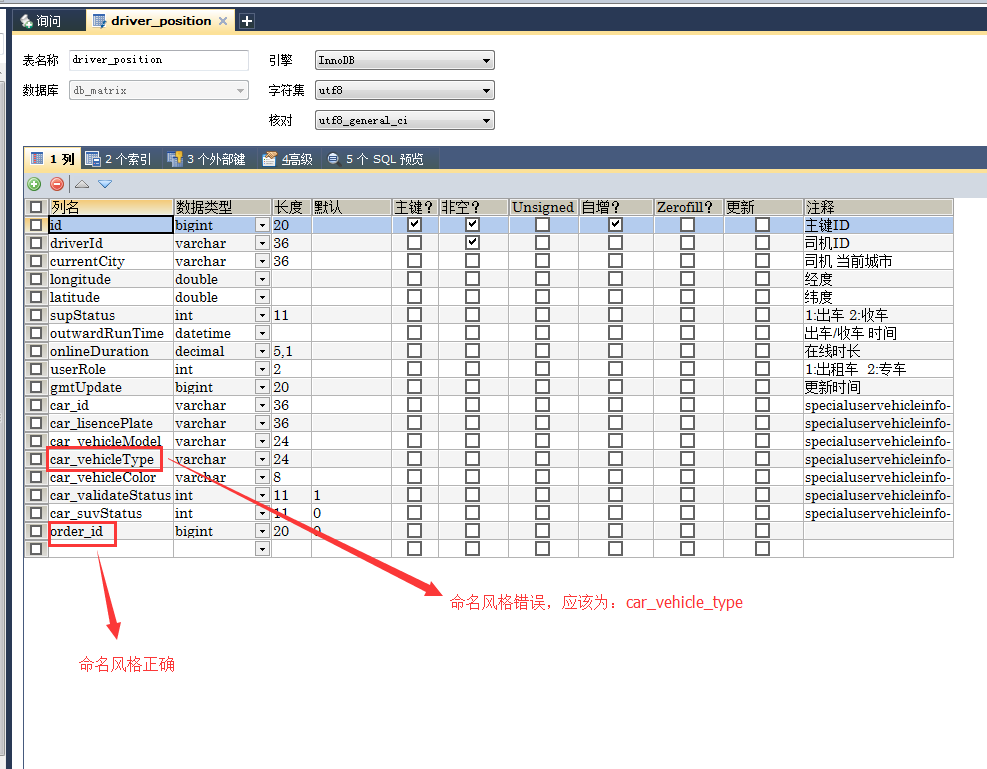
2 项目结构稍乱

3 大量使用枚举，系统可配置性差，软件国际化程度低

 对比配置文件形式 -> 

4 数据库命名风格有问题，数据表命名风格有问题，数据表字段命名风格有问题



5 各个数据库严重缺乏数据库字典文档

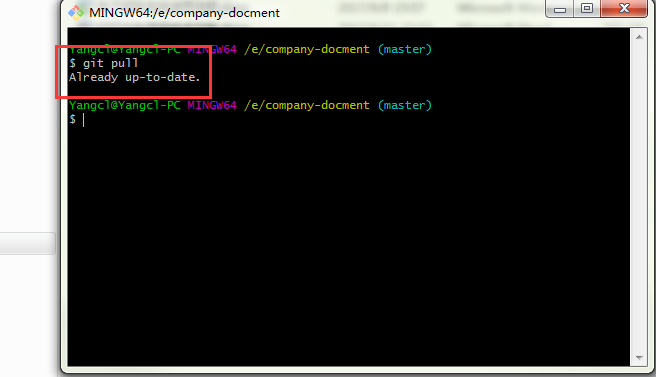
6 源代码管理存在比较大的问题

正常来讲，一家公司Git服务器上的代码应该以master为主分支，代码是最新的，最稳定的；但我们的是以dev(开发分支)为最新代码，而主分支仅仅作为历史依据。

大部分同事依然在使用git命令行的方式来维护自己的源代码分支，而不是使用简洁、明了、易于维护的图形化界面来操作。

这样做相当于在DOS下敲命令，不适用windows窗口一样，效率低，速度慢。

如下图：



Git图形化界面如下图：

