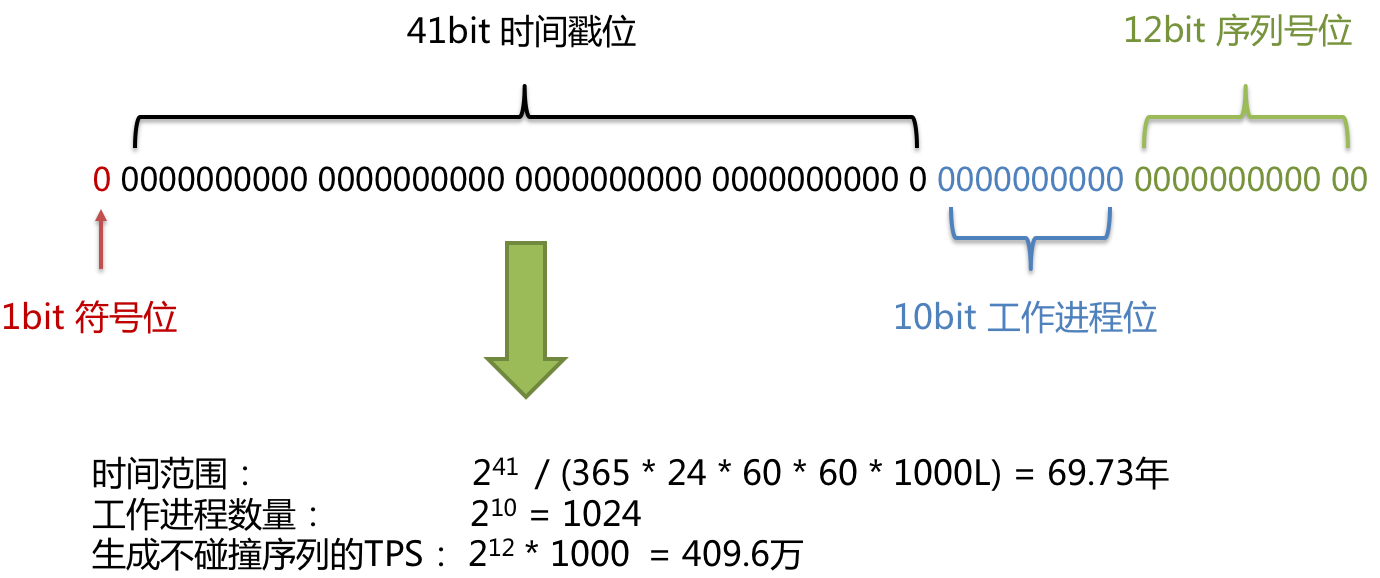
# 分布式主键

传统数据库软件开发中，主键自动生成技术是基本需求。而各个数据库对于该需求也提供了相应的支持，比如MySQL的自增键，Oracle的自增序列等。 数据分片后，不同数据节点生成全局唯一主键是非常棘手的问题。同一个逻辑表内的不同实际表之间的自增键由于无法互相感知而产生重复主键。 虽然可通过约束自增主键初始值和步长的方式避免碰撞，但需引入额外的运维规则，使解决方案缺乏完整性和可扩展性。

io.shardingsphere.core.keygen.DefaultKeyGenerator



# Sharding-proxy介绍

Sharding-Proxy是ShardingSphere的第二个产品。 它定位为透明化的数据库代理端，提供封装了数据库二进制协议的服务端版本，用于完成对异构语言的支持。 目前先提供MySQL版本，它可以使用任何兼容MySQL协议的访问客户端(如：MySQL Command Client, MySQL Workbench等)操作数据，对DBA更加友好。

# Atlas介绍：

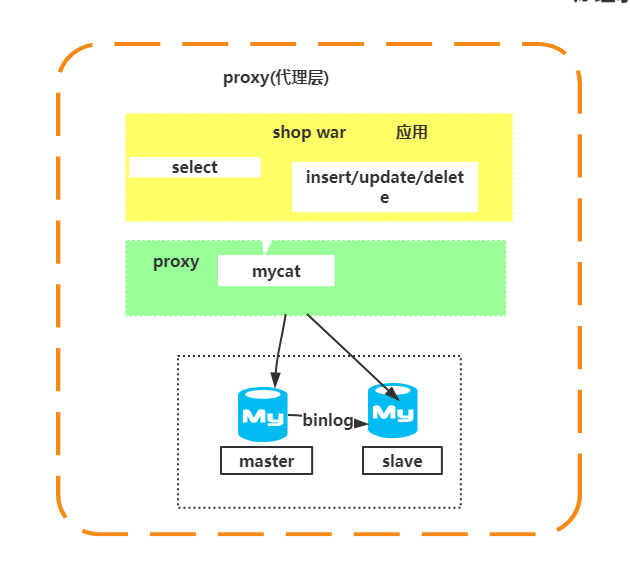
Atlas是由 Qihoo 360公司Web平台部基础架构团队开发维护的一个基于MySQL协议的数据中间层项目。它在MySQL官方推出的MySQL-Proxy 0.8.2版本的基础上，修改了大量bug，添加了很多功能特性。目前该项目在360公司内部得到了广泛应用，很多MySQL业务已经接入了Atlas平台，每天承载的读写请求数达几十亿条。同时，有超过50家公司在生产环境中部署了Atlas，超过800人已加入了我们的开发者交流群，并且这些数字还在不断增加。

主要功能：

<https://github.com/Qihoo360/Atlas/blob/master/README_ZH.md>

## Proxy部署流程：

## 架构图



# 1、环境

192.168.0.156（数据库地址）

192.168.0.31（proxy地址）

# 2、部署

## 2.1下载

下载地址：（最新版本还没有提供下载地址，需要下载源码，自行编译打包）

<https://github.com/sharding-sphere/sharding-sphere-doc/raw/master/dist/sharding-proxy-3.0.0.tar.gz>

目前最新版本3.1.0支持分布式事务。没找到下载地址的，我这儿发给大家。

## 2.2解压：

tar -zxvf sharding-proxy-3.0.0.tar.gz

## 2.3 配置：

解压之后进入到conf目录

修改：conf/config-xxx.yaml

修改：conf/server.yaml

不同版本配置不一样，官网文档是3.0.0的版本，如果用最新版本从源码里面copy文件。

config-sharding.yaml

|  |
| --- |
| schemaName: sharding\_db  dataSources:  ds\_0:  url: jdbc:mysql://127.0.0.1:3306/demo\_ds\_0?serverTimezone=UTC&useSSL=false  username: root  password:  connectionTimeoutMilliseconds: 30000  idleTimeoutMilliseconds: 60000  maxLifetimeMilliseconds: 1800000  maxPoolSize: 50  ds\_1:  url: jdbc:mysql://127.0.0.1:3306/demo\_ds\_1?serverTimezone=UTC&useSSL=false  username: root  password:  connectionTimeoutMilliseconds: 30000  idleTimeoutMilliseconds: 60000  maxLifetimeMilliseconds: 1800000  maxPoolSize: 50  shardingRule:  tables:  t\_order:  actualDataNodes: ds\_${0..1}.t\_order\_${0..1}  tableStrategy:  inline:  shardingColumn: order\_id  algorithmExpression: t\_order\_${order\_id % 2}  keyGeneratorColumnName: order\_id  t\_order\_item:  actualDataNodes: ds\_${0..1}.t\_order\_item\_${0..1}  tableStrategy:  inline:  shardingColumn: order\_id  algorithmExpression: t\_order\_item\_${order\_id % 2}  keyGeneratorColumnName: order\_item\_id  bindingTables:  - t\_order,t\_order\_item  defaultDatabaseStrategy:  inline:  shardingColumn: user\_id  algorithmExpression: ds\_${user\_id % 2}  defaultTableStrategy:  none:  defaultKeyGeneratorClassName: io.shardingsphere.core.keygen.DefaultKeyGenerator |

server.yaml

|  |
| --- |
| orchestration:  name: orchestration\_ds  overwrite: true  registry:  namespace: orchestration  serverLists: localhost:2181  #依赖了zk地址  authentication:  username: root  password:  props:  executor.size: 16  sql.show: true  proxy.transaction.type: XA |

## 2.4 启动服务

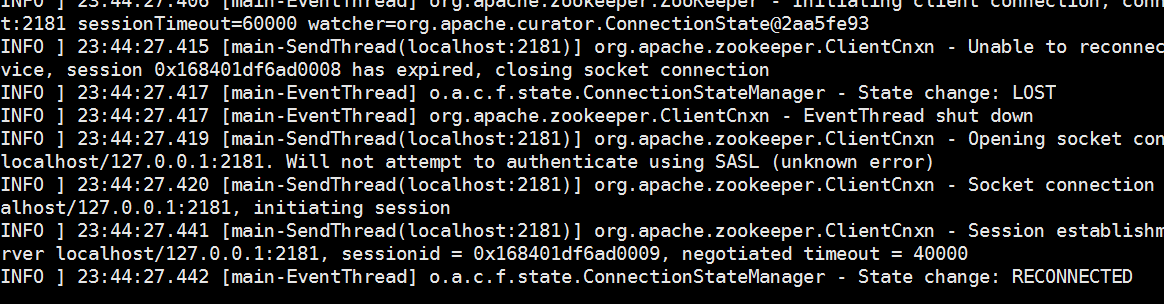
启动服务之前，需要先启动zookeeper

启动服务：sh start.sh

停止服务：sh stop.sh

## 2.5验证是否成功

tail -f ../logs/stdout.log 查看是否有错误。



用客户端测试连接下，如果可以访问说明服务成功。

