

大规模信息系统构建技术导论

分布式MiniSQL系统个人设计报告

2023学年 第一学期（上）

组员信息

|  |  |
| --- | --- |
| 学号 | 姓名 |
| 3200103483 | 孙宇桐 |
|  |  |

2023年 5月 24日

**1 引言**

* 1. 系统目标

本系统为《大规模信息系统构建技术导论》的课程项目，项目全称为分布式Minsiql系统，该旨在通过课程学习知识将之前数据库课程所制作的Minisql拓展为能够支持大规模数据的**分布式关系型简易数据库系统**

该系统主要包括Zookeeper集群、Client客户端、Master Server主服务器与Region Server区域服务器四个部分，支持包括简单SQL语句的解析与处理、容错容灾、主从备份、负载均衡等功能。

该系统使用Java作为语言开发，同时使用Github进行代码管理和协作开发，由小组五名成员共同协作完成。

1.2 个人任务说明

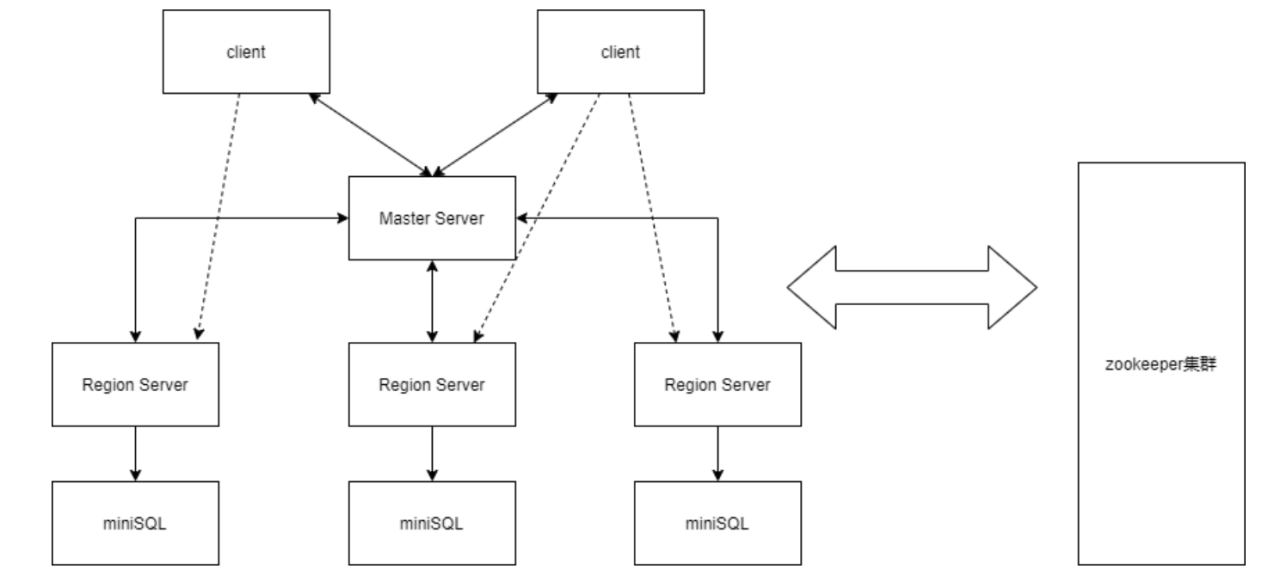
|  |  |  |
| --- | --- | --- |
| 成员姓名 | 学号 | 分工 |
| 孙宇桐 | 3200103483 | **Region Server**总体开发， **MiniSQL**中Interpreter、API开发、Catalog Manager修改与**Zookeeper**集成测试、Region相关处理与通知 |

本人在本系统开发中主要负责Region Server的总体开发以及其与Client、Master、Zookeeper三者间的通信处理，同时为了适应分布式的信息处理，本人对底层的MiniSQL的Interpreter、API、Catalog Manager等做了相关调整和代码修改，并针对对应功能进行了相关的模块代码编写。

除此之外，本人参与了Zookeeper系统的部分代码集成。为了适应本系统功能，本小组未使用Apache Zookeeper而是自己编写了Zookeeper相关代码，本人对小组另一名成员的Zookeeper代码进行了相关测试和修改使其能够支持多线程通信以及相关信息处理。

**2 系统设计**

2.1 系统总体架构

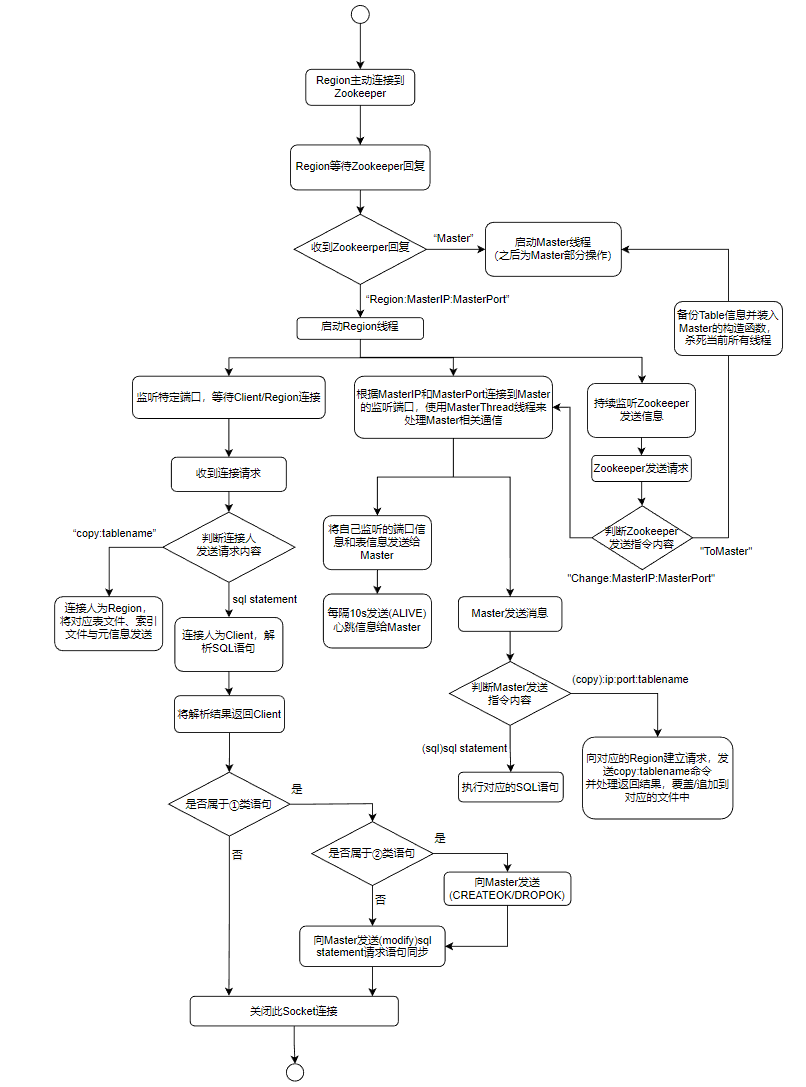


**图1 分布式MiniSQL总体架构**

该系统中Region Server主要负责处理以下多个部分的功能内容：

1. 主动连接Zookeeper集群，获取Master节点地址信息，并维持连接
   1. 如果收到master\_change信息，代表Master已经切换，则连接到新 的Master节点（容错容灾）
   2. 如果收到master信息，代表当前Region被选举为新的Master，则 关闭相关线程，启动Master相关线程并且传输当前Region所缓存 的表信息（容错容灾）
2. 主动连接Master节点，发送自己监听端口与表信息
3. 维持与Master节点的Socket长连接，以10s为间隔发送心跳，并且持 续监听来自Master的信息
   1. 如果是copy信息：获取对应要复制的Region信息以及对应的表名 称，主动建立到对应Region的Socket通信，接受以字节流传输的 表信息并将其加载到对应文件中（主从备份）
   2. 如果是sql信息：执行Master发送的对应SQL语句（主从备份）
4. 主动监听端口，等待来自Client以及Region的连接
   1. 对于Client的连接，处理其发送来的SQL语句并回复处理结果
   2. 对于Region的连接，将其索取表的Catalog元信息以及表信息、索 引信息以字节流形式发送回去。

2.2 总体流程图



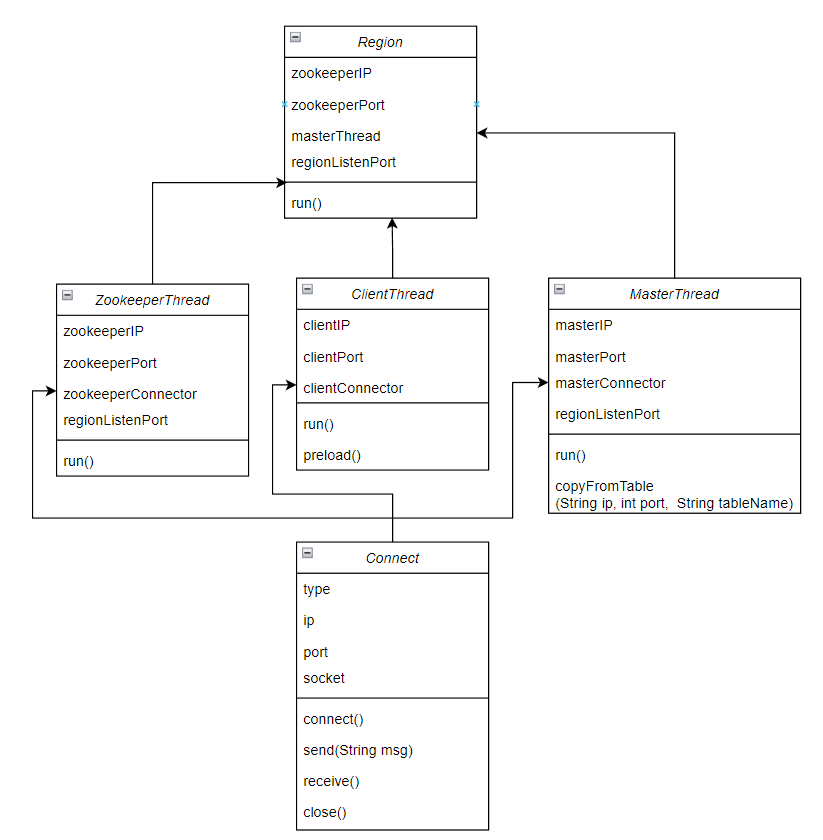
**图2 Region部分总体流程图**

注：①类语句为 create drop insert delete

②类语句为 create drop

2.3 Region设计

本系统的Region Server模块设计如下图所示：



**图3 Region总体设计类图**

Region由一个统一个Region类进行管理，该类下方还有三个子功能类分别用于与不同对象的连接，分别是：

1. ZookeeperThread：负责Region与Zookeeper的连接，在其中使用Connect类提供的接口和方法与Zookeeper建立长连接，能够根据Zookeeper发送的内容决定连接到Master、切换Master或者被选举为Master的消息。
2. MasterThread：负责Region与Master的连接，会定期向Master发送心跳来表情自己是否能够正常工作，并且会接受处理Master发送的复制表（用于主从备份）以及执行SQL语句（用于主从节点信息更新）的语句。
3. ClientThread：负责Region与Client的连接，会接受Client发送的SQL数据，将其预处理成minisql可以处理的语句格式并且交由minisql的Intepreter执行，将执行结果返回给client,同时针对create、drop语句，会通知master更新表数据，并且如果主Region接受到了create、drop、insert、delete四种会对表产生改变的语句，会通知master将sql语句同步给从Region节点。

此外，为了方便通信，还设计了一个Connect类，其内部封装了Socket通信的connect/send/receive/close四个函数，用于简化代码结构。实现代码复用。

**3 功能实现**

3.1 通信协议

Region为了实现与Zookeeper、Region、Client、Master四者的通信，需要制定针对不同节点的通信规范，在此作出以下设计：

首先，Region与Zookeeper的连接是持久的长连接，因为region要时刻监听Zookeeper关于Master节点崩溃后重新选举/节点更换的信息。

其次，Region与Master的连接是持久的长连接，因为Region需要定期向Master发送心跳（证明自己可以正常工作），并且要接受Master关于主从备份/崩溃重启相关的消息。

同时，Region与Client的连接是短连接，我们设定为Client每一次的连接只会发送一条完整的sql语句，Region在处理完该sql语句并将处理结果发送回给Client后，应当直接关闭对应的socket连接。

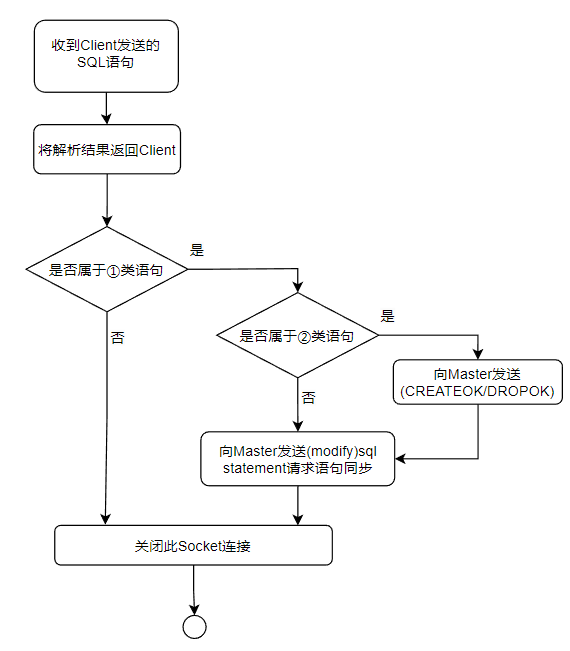
最后，Region与Region的连接是短连接，两者的通信只涉及表文件信息的传输，在传输完成后应当立刻关闭。

基于以上讨论，Region的通信协议设计如下：

**表1 通信协议**

|  |  |  |  |
| --- | --- | --- | --- |
| 协议格式 | 返回格式 | 方向 | 用途 |
| sql statement | result | client->region | 接受并处理client发送的sql语句 |
| (hello):port:table1,table2,...,tablen | null | region->master | 用于向master发送连接请求，并附上自己监听的端口以及表信息 |
| (Create/Drop) | null | region->master | 用于向master发送表create/drop成功信息 |
| (Alive) | null | region->master | 用于向master发送心跳 |
| (Showtable) | table1,table2,...,tablen | master->region | master向region询问表信息 |
| (copy)ip:port:tablename | null | master->region | master向region发送复制表命令 |
| (sql)sql statement | null | master->region | master向从region发送sql命令来更新表 |
| (modify)sqlstatement | null | region->master | 主region向master发送sql命令，请求更新从region |
| (copy)tablename | byte stream(file\_name, file\_length, file data) | region->region | 从region向主region请求表信息复制 |
| region:ip:port | null | zookeeper->region | Zookeeper宣布该节点注册为region，并通知master的ip和port |
| change:ip:port | null | zookeeper->region | Zookeeper宣布master发生更改， |
| toMaster | null | zookeeper->region | Zookeeper宣布region升级为master |

3.2 SQL语句查询

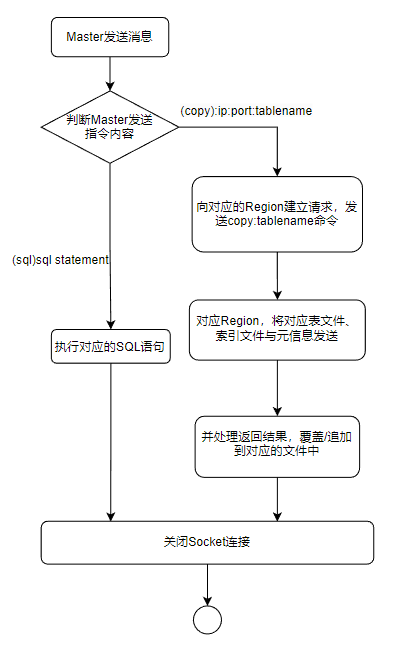


**图4 SQL语句查询流程图**

在Region接受到Client的SQL语句后，会将该SQL语句传入到preload()函数中，在该函数中，会进行以下步骤

1. 将多行语句合并为一条，并且以分号;作为语句的结束点
2. 将处理好的语句传输到minisql的Interpreter.interpret函数中，交给minisql进行语句处理，该函数会以String的形式返回执行结果
3. 将执行结果通过调用Connect.send函数发送给Client，并且对返回结果和sql语句进行判断
   1. 如果sql语句是create/drop并且执行成功，发送Create/Drop给master，这一步是为了更新Master中缓存的每个region存储表信息。
   2. 如果当前是主region并且执行成功create/drop/insert/delete类型对表有更改的语句，则向Master发送(modify)sql语句，用来告知Master应当将这个sql语句发送给从region来更新从region内容。
4. 考虑到与Client的连接是短连接，最后需要关闭client连接。

3.3 主从备份



**图5 主从备份流程图**

主从备份主要处理以下两个方面的问题：

1. 当一个Region新上线并且作为从Region时，Master会通知其去对应的主Region中复制对应的表
2. 当一个主Region出现了表信息的更新时，需要将这个信息通知给Master，并由Master来通知其对应的从Region来同步消息。

为了实现第一个功能，我在设计Region的时候刻意考虑了Region与Region间的通信，并且由于Minisql的持久化中，文件是以字节流的形式存储，因此在读取的时候需要对Minisql进行一定修改，具体实现如下：

1. Master给从Region发送 (copy)ip:port:tableName信息
2. 从Region解析信息，去建立连接到ip:port的主Region节点，并且发送copy:tableName请求复制表
3. 主Region在获取到信息后，解析出表明，进行以下操作：
   1. 模访CatalogManager中的store\_table格式，将tableName对应表的元信息填入一个用于暂时存储的ByteArrayoutputStream类tempDos中
   2. 利用DataOutputStream的writeUTF()函数向输出流写入table\_catalog, 利用writeInt()函数写入temoDos.length信息，利用write()函数写入tempDos信息
   3. 重复2过程，对index\_catalog执行相同的操作，传输索引元信息
   4. 利用FileInputstream读取对应tableName的表信息，利用上述相同的方式写入到输出流中
   5. 重复4过程，对tableName\_index.index文件执行相同的操作，传输文件索引文件信息

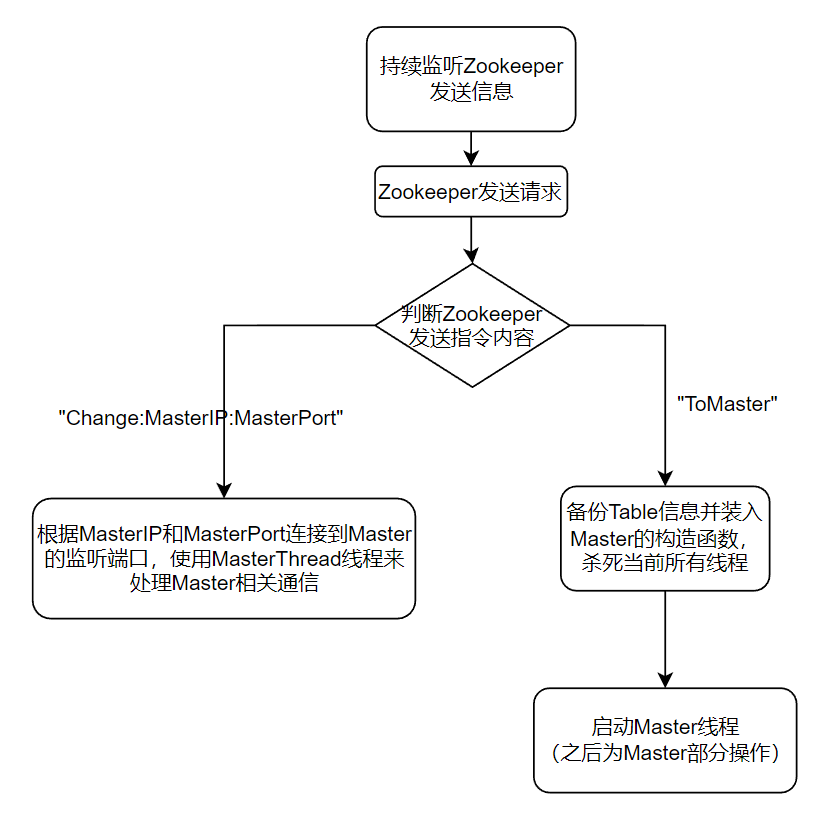
通过以上方式，主Region将全部文件相关信息以字节流的形式传输给从Region，从而实现主从备份

1. 从Region依次解析出对应的文件名、文件长度、文件内容信息，针对catalog相关信息，追加到对应文件中，针对文件信息，重写到对应文件中。

除了复制表方式的主从备份，为了方便从节点即时更新，我们决定采用sql语句同步的方式来维护主从节点的信息同步，其实现流程如下

1. 主节点完成了一条来自client的create/drop/insert/delete信息，且执行成功（即主节点中的表信息遭到了修改）
2. 主节点将使得信息修改的sql语句传输给master，由master通知从节点同步更改
3. 由于只有一条sql语句的修改，从节点不用去主节点中对整个表信息进行拷贝，而是可以通过master对sql语句的发送直接进行相同的修改，从而实现一样的效果。

3.4 容错容灾



**图6 容错容灾流程图**

容错容灾主要有Master以及Zookeeper来完成，其中涉及Region的部分主要有以下两个方面：

1. Region工作状态回馈：

为了实现Master能够实时监听到Region的状态，即Master需要知道Region当前是否适合处理sql语句等工作，Region需要定期向Master发送Alive心跳来表明自己的状态良好（这里其实也可以直接检测socket状态，但是我们考虑到可能有region在socket没有断开的情况下无法正常工作的情况【例如陷入死循环等】）

1. Master节点更新：

当Master节点更新时，Zookeeper可能会发送给Region两种命令：

* 1. change:masterIP:masterPort：工作流程与上流程类似，不过用于master节点崩溃后的master节点更新。收到该信息后当前会重新开启一个新的MasterThread线程，将其赋值给Region类的masterThread数据，切换到新的master连接。
  2. toMaster: 收到该信息后代表当前Region背选举为新的Master，收到该命令后Region会启动一个master部分的主线程升级为master，并且在处理完成当前命令后关闭Region相关线程。

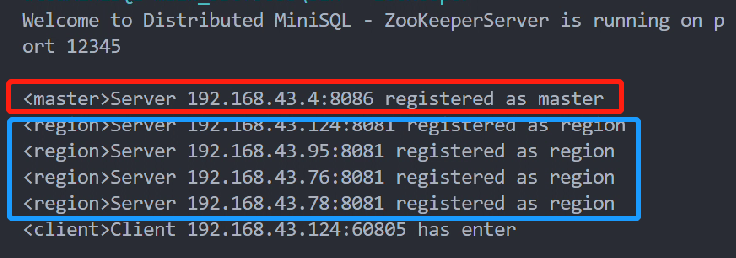
**4 测试与运行**

本项目测试中，我们一共启动了五个Server服务进程（其中一个注册为Master，四个注册为Region），一个Client服务进程以及一个Zookeeper服务进程，其中与Region相关的功能测试图片如下：

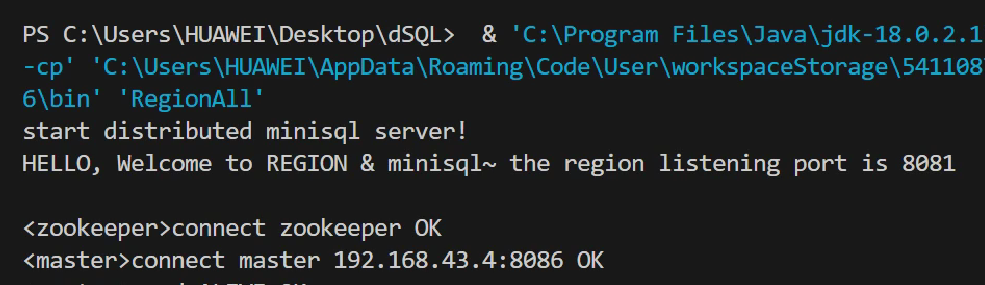
本组成员集成了Region与Master，两者共同运行在Region/RegionAll.java中

**登录上线：**

Server在上线的时候，首先会发送一个Server请求来判断自己身份，Zookeeper会选出一位Server来成为Master，其余Server将宣布为Region，因此可以看到该Server（192.168.43.76:8081）注册为Region，并且成功连接Master



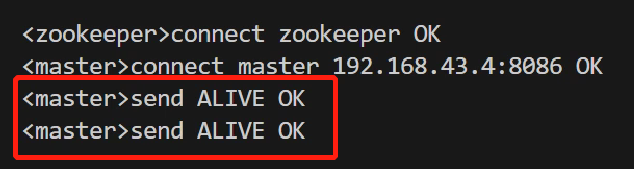
**图7 登录上线 Zookeeper 部分**



**图8 登录上线 Region 部分**

**Master心跳发送（10s间隔）：**

在成功连接Master后，Region以10s为间隔向Master播报自己的情况。Master会检测Region是否按时发送了心跳，如图10所示，如果按时发送则显示Region可以正常工作，如果超时则会判断Region无法正常工作并且通知Zookeeper。



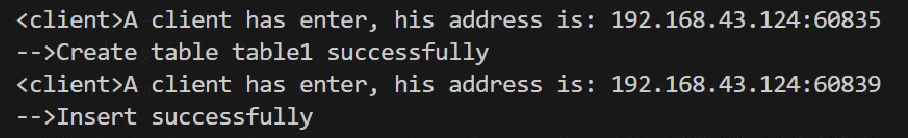
**图9 心跳测试 Region 部分**



**图10 心跳测试 Master 部分**

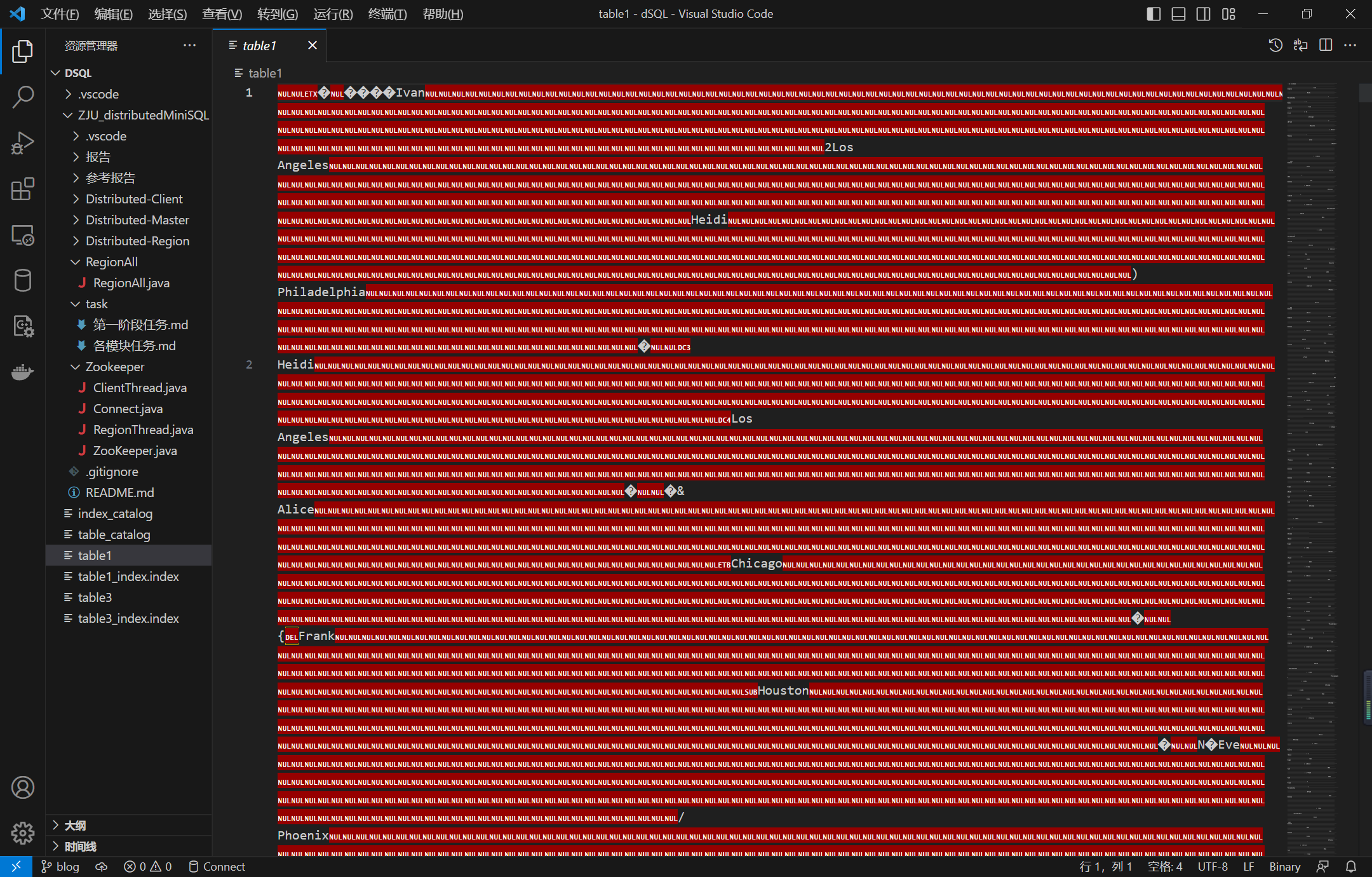
**处理SQL语句：**

Client会主动和Region建立Socket连接，并且发送对应的SQL语句，Region会使用minisql对其进行解析处理，执行相关语句，并且将返回结果打印并且回报给Client



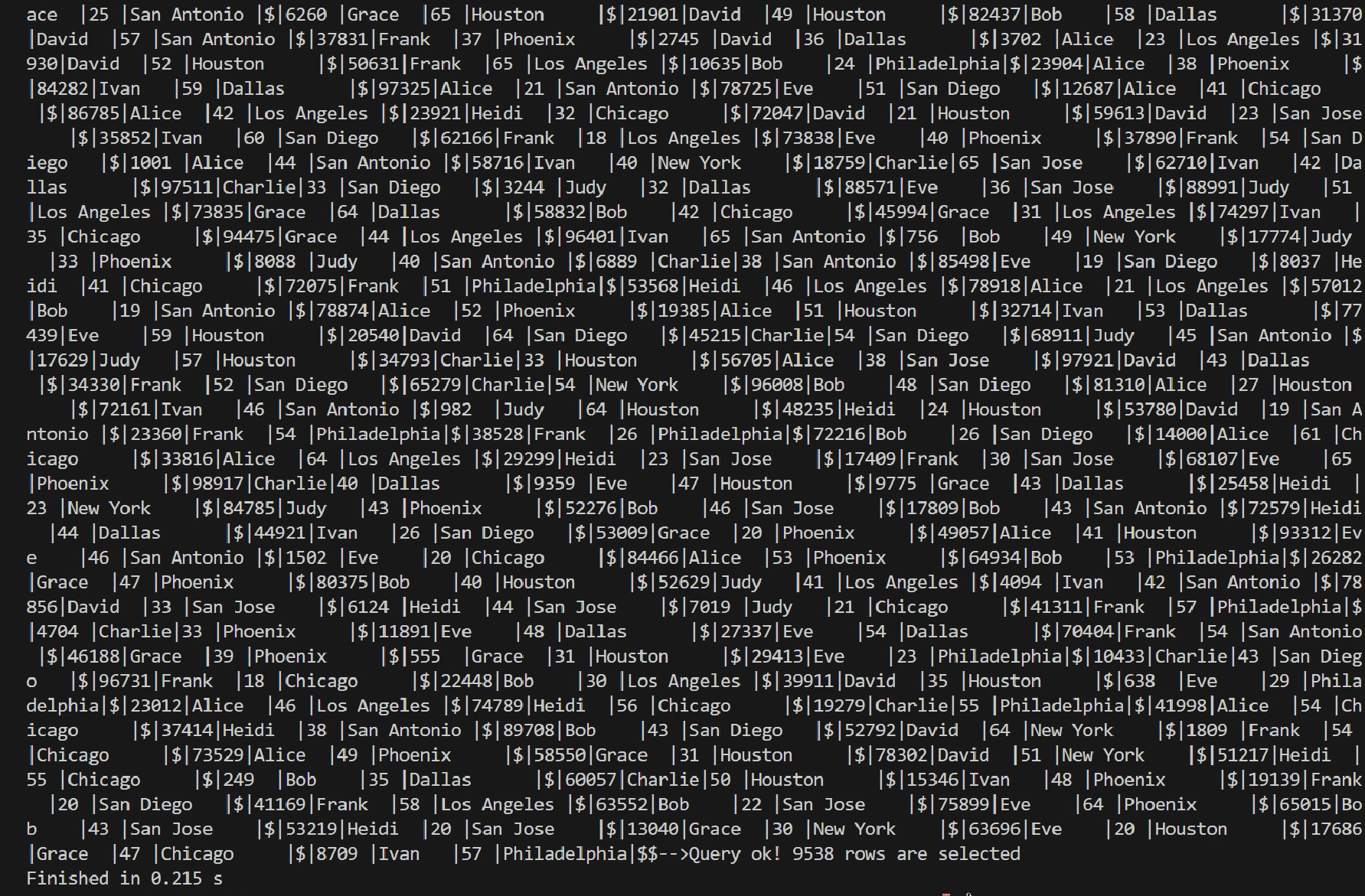
**图11 SQL测试 Region部分**

同时，Region中的数据以bytes文件进行存储

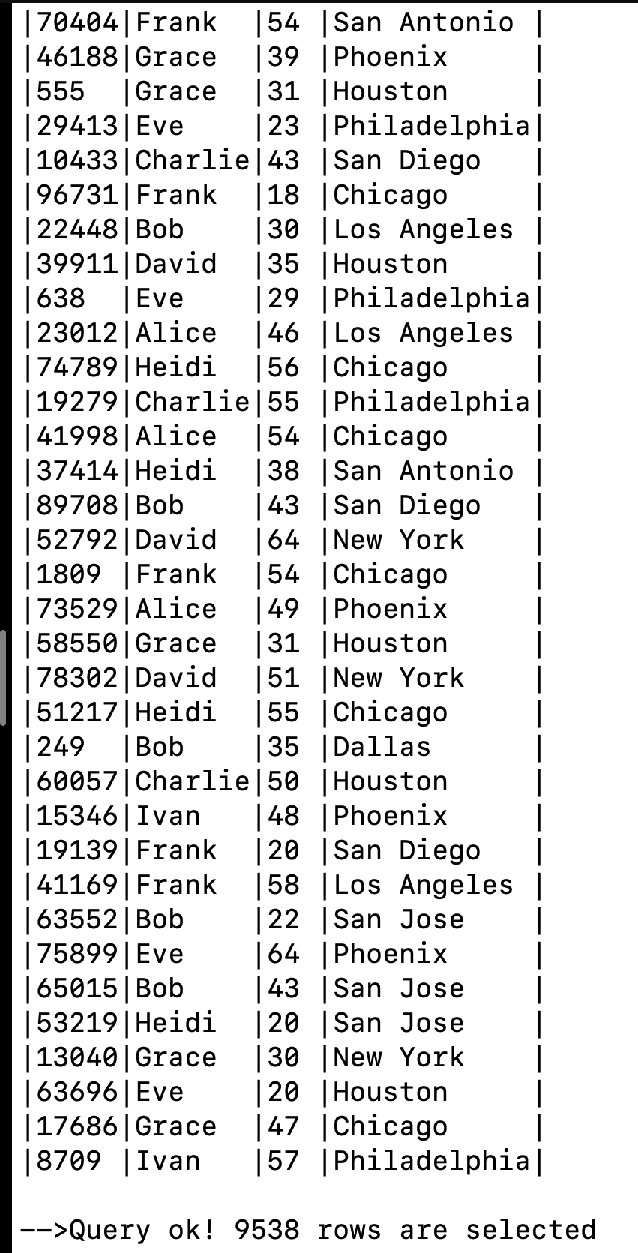


**图12 Region端文件存储**

针对select等语句，考虑到write无法正常传输换行符，我们约定使用$代替\n进行传输。下图是对10000条插入语句（存在重复）的select结果，Client会对其进行解析。



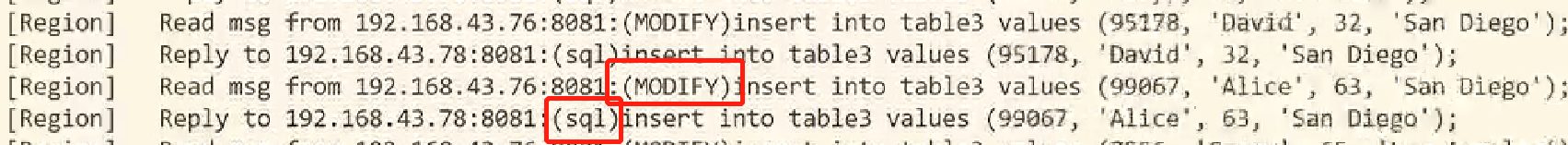
**图12 Select测试Region部分**



**图13 Select测试Client部分**

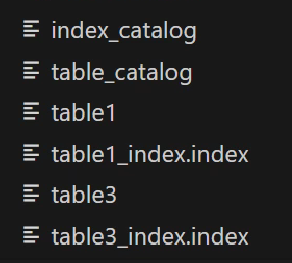
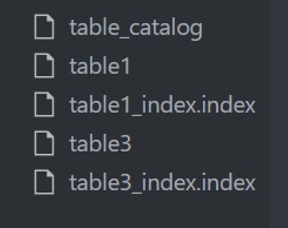
**主从备份：**

在主节点执行SQL语句的同时，master会将信息同步给从节点。当主Region收到一条client发送的SQL①类语句，其会向Master发送一个MODIFY请求，同时Master会将该sql语句以SQL请求的方式同步给从节点



**图14 主从备份，master页面**

可以看到从Region同步了主Region1的表信息

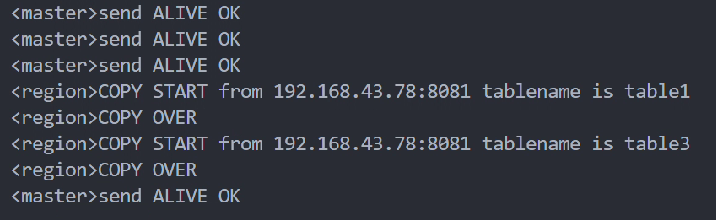
**图15 主从备份，主从Region表信息界面**

**容错容灾（Region死亡部分）：**

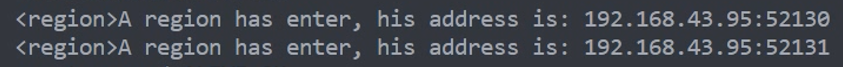
region死亡后，master会通知Zookeeper region死亡信息，并启动相对应的主从备份。在这里，我们杀死了Region1（其从节点为Region2），则Master会启动一台新的Region3来代替Region1，Region3会从Region2处拷贝信息。

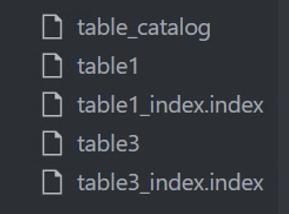


**图16 Region1死亡，Zookeeper页面**



**图17 Region1死亡，Region3页面**



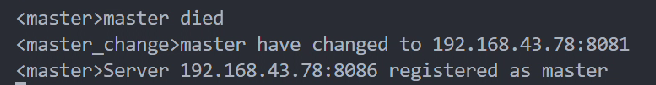


**图18 Region1死亡，Region2页面**

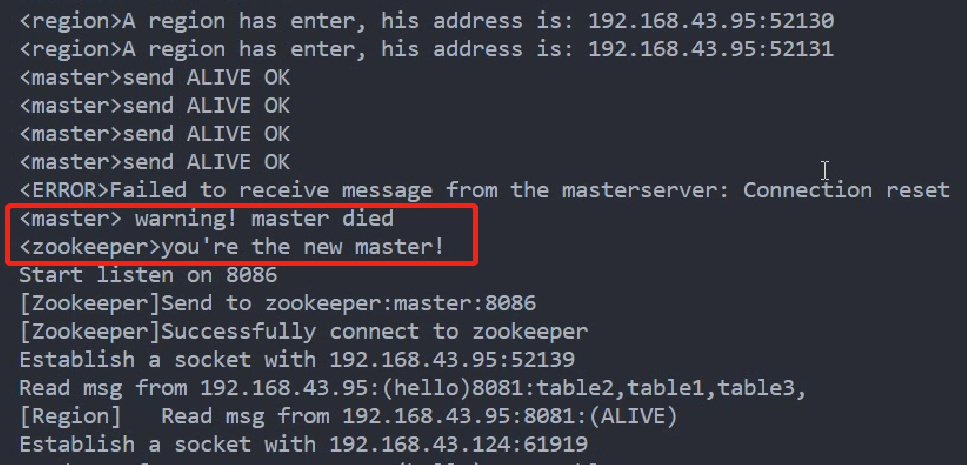
可以看到Region3向Region2拷贝了table1、table3两张表。

**容错容灾（Master死亡部分）：**

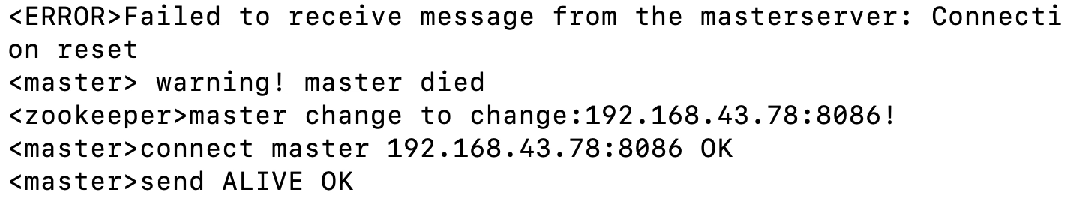
master死亡后，zookeeper会宣布一台可以正常工作的Region升变为Master，关闭其所有的region相关线程并启动master相关线程。



**图19 Master死亡Zookeeper选择**



**图20 Master死亡Region被选中**



**图21 Master死亡其它Region重新连接**