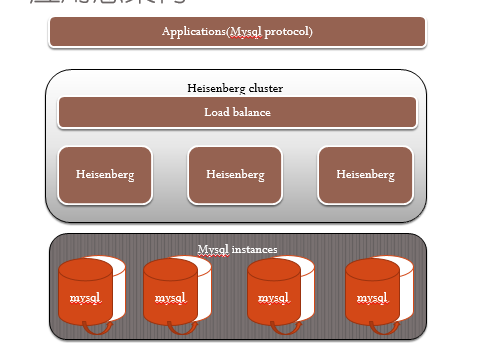
# Heisenberg分库分表开发指南

Heisenberg结构  


Heisenberg是基于mysql协议一个分库分表中间件服务器，其上层应用不需要关心相关的分库分表，而直接使用heisenberg的shard功能

## 环境搭建

Maven +JDK 部署好

svn checkout [**http**://heisenberg.googlecode.com/svn/trunk/](http://heisenberg.googlecode.com/svn/trunk/)

下载到本地后，

Mvn package 之

在本地target里会生成一个heisenberg-server-1.0.1.zip 文件

解压之

目录结构如下：



进入conf目录



进入bin目录



sh startup.sh 开启服务

## 数据库搭建

这个就自然不必说

百度有个很好的初始化工具：

<http://dba.baidu.com/r/init/list.php>

主从，建库神马的都可以

这里提供下我建的二个库（主从）：

Master:cq02-mco-lctest00.cq02.baidu.com:8701 root/st0078

Slave: cq02-mco-lctest01.cq02.baidu.com:8701 root/st0078

## 分库表配置

进入conf文件夹,见下面

### 日志配置

vi log4j.xml

将相关的日志改为info

Root也改为info

Alarm的保持为error即可。

### Server配置

Vi server.xml

#### 处理器配置

|  |
| --- |
| <system>  <property name=*"serverPort"*>8166</property>  <property name=*"managerPort"*>8266</property>  <property name=*"initExecutor"*>16</property>  <property name=*"timerExecutor"*>4</property>  <property name=*"managerExecutor"*>4</property>  <property name=*"processors"*>4</property>  <property name=*"processorHandler"*>8</property>  <property name=*"processorExecutor"*>8</property>  <property name=*"clusterHeartbeatUser"*>\_HEARTBEAT\_USER\_</property>  <property name=*"clusterHeartbeatPass"*>\_HEARTBEAT\_PASS\_</property>  </system> |

*serverPort为服务端口，即对上层应用的端口*

*managerPort为管理端口，即管理的监听端口，用于操作服务器一些配置等*

*initExecutor 为初始化的线程个数*

*timerExecutor 心跳执行线程个数*

*managerExecutor管理执行线程个数*

*processors应用接收处理器核数*

*processorHandler 应用接收处理类个数*

*processorExecutor 应用接收处理线程个数*

*clusterHeartbeatUser和clusterHeartbeatPass 不必改，用于集群的认证方式使用*

#### 应用用户密码

|  |
| --- |
| <user name=*"brucexx"*>  <property name=*"password"*>st0078</property>  <property name=*"schemas"*>trans\_shard</property>  </user> |

Brucexx为自定义应用用户名，st0078为自定义应用密码

Schemas为自定义schema,具体见schema.xml中，

这里的schemas可为多个，以逗号分隔

#### 集群设定

|  |
| --- |
| <cluster>  <node name="cobar1">  <property name="host">127.0.0.1</property>  <property name="weight">1</property>  </node>  <node name="cobar2">  <property name="host">127.0.0.1</property>  <property name="weight">1</property>  </node>  </cluster> |

这里定义了集群结点的一些结点，仅限于使用heisenberg jdbc driver做集群负载的，这里会定义一些集群信息，这里面heisenberg会自动管理踢除相应的结点

#### 白名单限制

|  |
| --- |
| <quarantine>  <host name="1.2.3.4">  <property name="user">test</property>  </host>  </quarantine> |

上面是只能test用户进行登陆

### schema配置

#### mysql数据源

|  |
| --- |
| <dataSource name=*"transDS"* type=*"mysql"*>  <property name=*"location"*>  <location>10.58.49.14:8701/db$0-9</location>  </property>  <property name=*"user"*>root</property>  <property name=*"password"*>st0078</property>  <property name=*"sqlMode"*>STRICT\_TRANS\_TABLES</property>  </dataSource> |

*这里指定的mysql的数据源，后面$0-9是一种自定义的缩略写法*

*也可以在property里面定义多个location，比如：*

|  |
| --- |
| <property name=*"location"*>  <location>10.58.49.14:8701/db0</location>  <location>10.58.49.14:8701/db1</location>  <location>10.58.49.14:8701/db2</location>  </property> |

*效果是一样的*

#### Shard结点配置

Shard结点相当于一个逻辑结点，提供给外部相关的schema，对应于数据源有

主/备/灾，

|  |
| --- |
| <dataNode name=*"transDN"*>  <property name=*"dataSource"*>  <!-- 主库 -->  <dataSourceRef>transDS$0-9</dataSourceRef>  <!-- 备库 -->  <dataSourceRef>transSlaveDS$0-9</dataSourceRef>  <!-- 灾库 -->  <dataSourceRef>transSlaveDS$0-9</dataSourceRef>  <!-- slave,暂无 -->  <!-- dataSourceRef>ds\_shard\_slave$0-3</dataSourceRef -->  </property>  <property name=*"rwRule"*><![CDATA[m:0,s:1]]></property>  <property name=*"poolSize"*>256</property>  <property name=*"heartbeatSQL"*>select user()</property>  </dataNode> |

属性dataSource 第一个是主库，第二个备库，第三个灾库，需要多少配置多少个

读写分离规则rwRule，m和s代表读取的比例，表示主库读取为0,从库读取1，这样直接读写分离，如果是1:1的话相当读取各1：1的比例

池大小poolSize为到mysqlDB的连接数和心跳sql heartbeatSQL，无特殊需求保持不变

#### Schema配置

|  |
| --- |
| <schema name=*"trans\_shard"*>    <table name=*"trans\_online, trans\_content, trans\_tb "* dataNode=*"transDN$0-9"* rule=*"rule1"* />  </schema> |

*trans\_shard 提供的schema,对应于server.xml中的名字*

*下面会有多个需要分库的表，*

<table name=*"trans\_online"* dataNode=*"transDN$0-9"* rule=*"rule1"* />

这里必须要把需要分库分表的内容写出来，当然，如果不分库表也是可以的

<table name=*”tbxxx"* dataNode=*"transDN0" ruleRequired=”false”*/>

### 分库分表rule

先上配置 vi rule.xml

|  |
| --- |
| <rule>  <tableRule name=*"rule1"*>  <columns>TRANS\_ID</columns>  <dbRuleList>  <dbRule><![CDATA[#set($start=$TRANS\_ID.length() - 2)##  #set($end=$TRANS\_ID.length() - 1)##  $stringUtil.substring($TRANS\_ID,$start,$end)]]>  </dbRule>  </dbRuleList>  <tbRuleList>  <tbRule><![CDATA[#set($start=$TRANS\_ID.length() - 2)##  $stringUtil.substring($TRANS\_ID,$start)]]></tbRule>  </tbRuleList>  <!-- 00-99 100个表,每个表属于哪个结点 Map<Integer,Set<String>> -->  <tbPrefix>  <![CDATA[  def map = [:];  for (int i=0; i<10; i++) {  def list = [];  for (int j=0; j<10; j++) {  list.add(i+""+j);  }  map.put(i,list);  };  return map;  ]]>  </tbPrefix>  </tableRule>  </rule> |

其中<columns>TRANS\_ID</columns>可以有多个，以逗号分隔

这个列名必须要大写

分库分表使用velocity语法渲染，表前缀采用groovy进行初始化

#### 分库规则配置

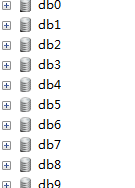
|  |
| --- |
| <dbRuleList>  <dbRule><![CDATA[#set($start=$TRANS\_ID.length() - 2)##  #set($end=$TRANS\_ID.length() - 1)##  $stringUtil.substring($TRANS\_ID,$start,$end)]]>  </dbRule>  </dbRuleList> |

分库规则dbRuleList可以有多个dbRule，当第一个不满足时，可以用第二个，当然这个效率不好，如果有规则区分，尽量再写一个rule，

dbRule 最后的结果是表的前缀

比如分库分表 库名为db0-db9,那么这个dbRule渲染时

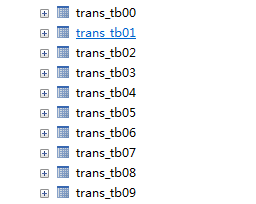
取到TRANS\_ID 这个为后，在脚本里计算出取倒数第2位为库后缀



#### 分表规则配置

|  |
| --- |
| <tbRuleList>  <tbRule><![CDATA[#set($start=$TRANS\_ID.length() - 2)##  $stringUtil.substring($TRANS\_ID,$start)]]></tbRule>  </tbRuleList> |

这个和上面分库一样了，以倒数1，2位为库的后缀



有个潜规则就是

**需要保证全局的表名不能重复**

**比如db0有个trans\_tb00,db1就不能有叫trans\_tb00的表**

#### 表初始化

|  |
| --- |
| <!-- 00-99 100个表,每个表属于哪个结点 Map<Integer,Set<String>> -->  <tbPrefix>  <![CDATA[  def map = [:];  for (int i=0; i<10; i++) {  def list = [];  for (int j=0; j<10; j++) {  list.add(i+""+j);  }  map.put(i,list);  };  return map;  ]]>  </tbPrefix> |

需要初始化个表，其中key为db的下标索引，比如db0 的下标为0,

list为每个库里的表后缀名

目录是为了初始化定义这些库表

## 服务器部署注意事项

1.使用bin/startup.sh 脚本启动即可

2.集群负载使用bvs

3.必须要通过heisenberg直接连接mysqlDB，如果通过dbproxy又隔了一层，效率会低很多

## 性能初测结果

年前测试约2300TPS，和客户端连接数有关，heisenberg服务器未到瓶颈,load 在0.0-0.2之间，二台物理mysql load 已经到0.9-2.0，由于时间原因未达到极限