**Morning：**

1、MongoDB副本集

**-What’s 副本集**

相当于高可用集群+主从同步；

提高数据可用性、安全性，方便数据故障恢复。

**-工作原理**

至少2节点，一个主节点(处理客户端请求)，其余是从节点(复制数据)；

主节点记录所有oplog(操作日志)，从节点定期轮询主节点获取操作，对本机数据副本执行这些操作。

**-副本集实现方式**

Master-Slave（不带故障自动切换，已被淘汰）

Replica Sets（支持故障自动切换、自动修复成员节点）

2、配置Replica Sets

**-指定副本集名称**

[root@mongo51..53 ~]#

echo replSet=rs1 >> /usr/local/mongodb/etc/mongodb.conf

[root@mongo51..53 ~]# mstop && mstart

#配置文件指定副本集名称，重启服务

**-配置节点信息**

在任意一台主机连接mongod服务，执行以下操作：

> config = {

... \_id:"rs1",

... members:[

... {\_id:0,host:"192.168.4.51:27051"},

... {\_id:1,host:"192.168.4.52:27052"},

... {\_id:2,host:"192.168.4.55:27055"}

... ]

... };

Tips：默认第一台为主节点，或添加priority字段指定优先级（越小越优先）

**-初始化、查看状态**

> rs.initiate(config)

rs1:PRIMARY> rs.status() #查看状态信息

"\_id" : 0,

"name" : "192.168.4.51:27051",

"health" : 1, #健康状态

"state" : 1, #主1/从2

"stateStr" : "PRIMARY"

rs1:PRIMARY> rs.isMaster() #查看是否是master库

"ismaster" : true,

"secondary" : false,

"primary" : "192.168.4.51:27051",

"me" : "192.168.4.51:27051"

**-验证配置**

客户端连接主库 --> 修改数据 --> 客户端连接从库验证

rs1:SECONDARY> db.getMongo().setSlaveOk()

#允许从库查询数据（默认从库不可读写）

Tips：集群的配置信息存放在local库；

若配置错误，将所有主机的local库中集合删除，再重新配置。

**-Master选举**

1）master宕机，一段时间后把数据最接近的从机自动提升为master；

2）宕机恢复，自动作为slave加入到集群，把宕机期间的数据同步到本机。

**Afternoon：**

1、MongoDB文档管理

**-插入文档**

|  |  |
| --- | --- |
| 插入语法 | 例子 |
| **db.集合名.save({key:”值”, ...})**  #\_id字段值已存在时，修改文档值 | > db.tb1.save({\_id:3,name:"tom"}) |
| **db.集合名.insert({key:”值”, ...})**  #\_id字段值已存在时，放弃插入 | > db.tb1.insert({\_id:3,name:"tom"}) |
| **db.集合名insertMany(**  **[**  **{key:”值”, ...},**  **{key:”值”, ...},**  **]**  **)** | > db.tb1.insertMany(  ... [  ... {name:"bob",age:21},  ... {name:"gjq",job:"teacher"}  ... ]  ... ) |

Tips：集合不存在时自动创建集合

**-匹配条件**

|  |  |
| --- | --- |
| 匹配条件 | 例子 |
| 简单条件/与**{key:”值”, ...}** | >db.passwd.find({shell:"/bin/bash"})  >db.passwd.find({uid:0,gid:0}) |
| 在...里**{key:{$in:[v1,v2...]}}** | > db.passwd.find({uid:{$in:[0,1,2]}} |
| 不在...里**{key:{$nin:[v1,v2...]}}** | > db.passwd.find({uid:{$nin:[0,1,2]}} |
| 或**{$or:[{key:”值”},{key:”值”}]}** | >db.passwd.find({$or:[{name:"root"},{uid:1}]}) |
| 正则匹配**{key:/正则/}** | >db.passwd.find({name:/^ro/}) |
| 数值比较**{key:{$gt:数字,$lt:数字}}**  $lt $lte $gt $gte $ne  < <= > >= != | >db.passwd.find({uid:{$gt:0,$lt:10}},{\_id:0,uid:1,name:1}) |
| 匹配**null** | db.passwd.find({name:null}) |

Tips：匹配条件适用于find()，update()，remove()

**-查询文档**

|  |  |
| --- | --- |
| 查询语法 | 例子 |
| **db.集合名.find()** | 显示所有行(默认显示20行，输入it换页) |
| **db.集合名.findOne()** | 显示第1行 |
| **db.集合名find({条件},{定义显示的字段})** #0不显示，1显示 | >db.passwd.find({shell:"/bin/bash"},{\_id:0,name:1,gid:1}) |
| **db.集合名.find().limit(数字)** | 显示前几行 |
| **db.集合名.find().skip(数字)** | 跳过前几行 |
| **db.集合名.find().sort({key:1|-1})**  #1升序，-1降序 | > db.passwd.find().sort({uid:1}) |

**-更新文档**

|  |  |
| --- | --- |
| 更新语法 | 例子 |
| **db.集合名.update({条件},{修改的字段})** | #默认只修改与条件匹配的第1行  #文档其他字段删除，留下修改的字段  比较少用 |
| **db.集合名.update({条件},**  **{$set/$unset:{修改的字段}}**  **,false,true)**  #,false,true修改与文件匹配的所有行  #$set保留其他字段，修改字段值  #$unset保留其他字段，删除字段  #也可在原文档添加字段 | >db.passwd.update({uid:{$lt:2}},{$set:{pass:"y"}},false,true)  >db.passwd.update({uid:0},{$unset:{pass:"x"}}) |
| **db.集合名.update({条件},**  **{$inc:{字段名:数字}})**  #数字+/-可实现自增/减 | >db.passwd.update({uid:{$lt:2}},{$inc:{gid:10}},false,true) |
| **db.集合名.update({条件},**  **{$push:{数组名:”值”}})**  #向数组尾部添加新元素，允许重复 | >db.t1.update({\_id:0},{$push:{hobby:"eat"}}) |
| **db.集合名.update({条件},**  **{$addToSet:{数组名:”值”}})**  #向数组尾部添加新元素，避免重复 | >db.t1.update({\_id:0},{$addToSet:{hobby:"eat"}}) |
| **db.集合名.update({条件},**  **{$pop:{数组名:1 | -1}})**  #1删除尾部元素 -1删除头部元素 | >db.t1.update({\_id:0},{$pop:{hobby:-1}}) |
| **db.集合名.update({条件},**  **{$pull:{数组名:值}})**  #删除指定元素(全部) | >db.t1.update({\_id:0},{$pull:{hobby:"test1"}}) |

**-删除文档**

|  |  |
| --- | --- |
| 删除语法 | 例子 |
| **db.集合名.drop()**  #删除集合(表)，同时删除索引 | > db.passwd.drop() |
| **db.集合名.remove({})**  **db.集合名.remove({条件})**  #删除文档(行)，不删除索引 | > db.passwd.remove({uid:0}) |

[拓展] 用户授权、分片等（官网了解）