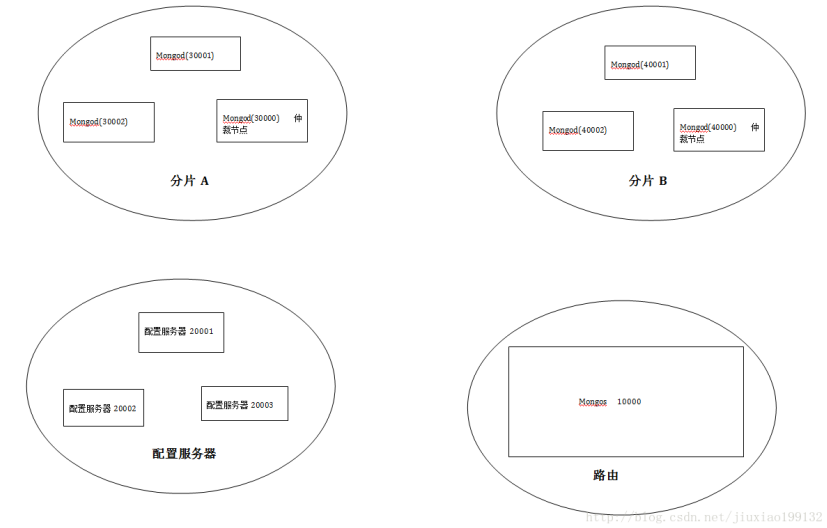
结构图



准备工作

在配置之前先说明几个概念

路由

请求的入口，所有请求都经过mongos协调和分发。通常部署多个实例，以便当一个mongos失败时，应用层驱动可以切换到其他正常的实例上。此外也可以通过一组mongos实例实现“池”的概念，在与应用层之间增加一层用于负载均衡的代理，将请求分配到“池”中的mongos实例上。mongos实例本身并不需要磁盘空间存储数据，它启动时会加载config server中的配置数据到内存，当config变化时会被通知更新

配置服务器

存储整个集群的元数据配置信息（路由、分片），mongos通过这些配置作为导向，将读写请求分发到不同的shard上

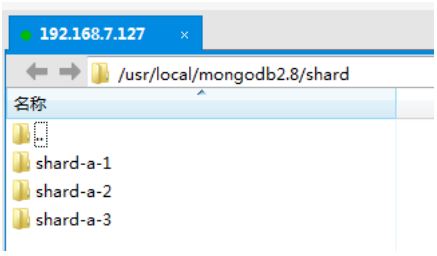
分片

数据库存储数据的组件，通过多实例达到负载均衡可伸缩目的，通过多副本主备切换达到避免单点失败的目的

配置分片

配置分片A

1 先建立好几个文件夹



2 启动分片A 这个 副本集

```
#主节点
[root@localhost bin]# ./mongod --port 30001 --shardsvr --replSet shard-a --dbpath /usr/local/mongodb
#从节点
[root@localhost bin]# ./mongod --port 30002 --shardsvr --replSet shard-a --dbpath /usr/local/mongodb
#仲裁节点
[root@localhost bin]# ./mongod --port 30000 --shardsvr --replSet shard-a --dbpath /usr/local/mongodb
```

3 配置分片A这个副本集

初始化副本集
连接mongo 30001这个节点

```
[root@localhost bin]# ./mongo --port 30001
```

执行命令

```
rs.initiate(
    {
        "_id":"shard-a",
        "members":[
            {
                "_id":0,
                "host":"192.168.7.127:30001"

            }
        ]
    }
);
```

添加从节点

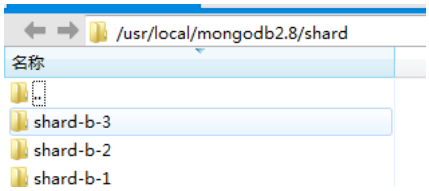
```
rs.add("192.168.7.127:30002")
```

添加仲裁节点

```
rs.add("192.168.7.127:30000",{arbiterOnly:true})
```

配置分片B

1 先建立好几个文件夹



2 启动分片B 这个 副本集

注意 目录和replSet 后面的集群名称 shard-b 不再是 shard-a

```
#主节点
[root@localhost bin]# ./mongod --port 40001 --shardsvr --replSet shard-b --dbpath /usr/local/mongodb
#从节点
[root@localhost bin]# ./mongod --port 40002 --shardsvr --replSet shard-b --dbpath /usr/local/mongodb
#仲裁节点
[root@localhost bin]# ./mongod --port 40000 --shardsvr --replSet shard-b --dbpath /usr/local/mongodb
```

3 配置分片B这个副本集

连接mongo 40001这个节点

```
[root@localhost bin]# ./mongo --port 40001
```

执行命令

```
rs.initiate(
    {
        "_id":"shard-b",
        "members":[
            {
                "_id":0,
                "host":"192.168.7.127:40001"

            }
         ]
     }
);
```

添加从节点

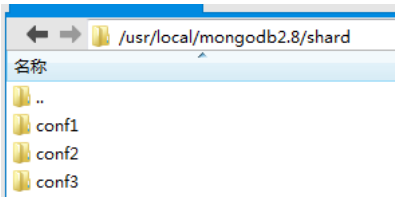
```
rs.add("192.168.7.127:40002")
```

添加仲裁节点

```
rs.add("192.168.7.127:40000",{arbiterOnly:true})
```

启动配置服务器

1 先建立如下文件夹



2 分别启动配置服务器

```
[root@localhost bin]# ./mongod --port 20001 --configsvr --dbpath /usr/local/mongodb2.8/shard/conf1/
[root@localhost bin]# ./mongod --port 20002 --configsvr --dbpath /usr/local/mongodb2.8/shard/conf2/
[root@localhost bin]# ./mongod --port 20003 --configsvr --dbpath /usr/local/mongodb2.8/shard/conf3/
```

启动路由

1 先建立如下文件夹

```
/usr/local/mongodb2.8/shard/route
```

2 启动路由

```
[root@localhost bin]# ./mongos --port 10000 --configdb 192.168.7.127:20001,192.168.7.127:20002,192.1
```

配置集群

1 连接路由节点

```
[root@localhost bin]# ./mongo --port 10000
```

2 添加分片

添加分片A

```
mongos> sh.addShard("shard-a/192.168.7.127:30001,192.168.7.127:30002")
```

添加分片B

```
mongos> sh.addShard("shard-b/192.168.7.127:40001,192.168.7.127:40002")
```

查看数据库

```
mongos> show dbs
```

输出结果

```
admin   (empty)
config  0.016GB
```

会发现有两个数据库 查看一下config数据库中的表

```
mongos> use config
mongos> show collections
```

输出结果

```
actionlog
changelog
chunks
databases
lockpings
locks
mongos
settings
shards
 system.indexes
 version
```

可以发现config数据库就存放了分片的信息

测试分片

目标：通过Java 驱动的方式写入到 数据库 cloud-users的表 user一千条数据

1 开启 cloud-uses 数据库分片，不管这个数据库是否存在

```
mongos> sh.enableSharding("cloud-users")
```

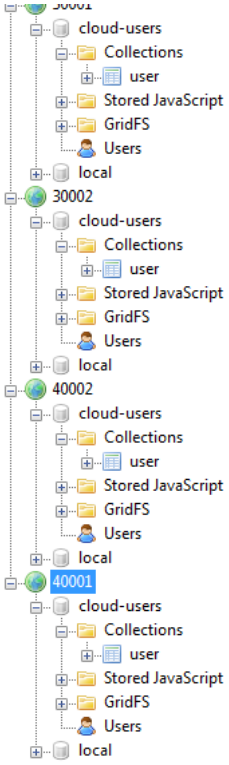
2 开启表 user的 分片，不管表是否存在直接执行命令即可， 分片键 由 age 和 id组合而成

```
mongos> sh.shardCollection("cloud-users.user",{"age":1,"_id":1})
```

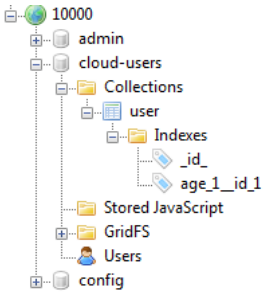
3 Java驱动执行代码

```
coll = new MongoClient( "192.168.7.127", 10000).getDatabase("cloud-users").getCollection("user");
for(int i = 0 ; i < 1000 ; i ++){
    Document doc = new Document();
    User u = User.initUser();
    doc.put("address", u.getAddress());
    doc.put("age", u.getAge());
    doc.put("email", u.getEmail());
    doc.put("height", u.getHeight());
    doc.put("job", u.getJob());
     doc.put("nickname", u.getNickname());
     doc.put("phone", u.getPhone());
     doc.put("school", u.getSchool());
     doc.put("sex", u.getSex());
     doc.put("hoby", u.getHoby());
     Document dog = new Document();
     dog.put("name", u.getDog().getName());
     dog.put("age", u.getDog().getAge());
     doc.put("dog", dog);
     coll.insertOne(doc);
}
```

4 查看数据分布



可以看到 30001 和 30002 数据完全一样， 40001 和 40002 也是完全一样 这是副本集的复制效果
30001 的数据 + 40001的数据 就是一个完整的一千条数据



路由的数据 正好 等于完整的 1000条

查看一下 路由 节点中 config数据库的 chunks表

```
/* 0 */
{
  "_id" : "cloud-users.user-age_MinKey_id_MinKey",
  "lastmod" : {
    "$timestamp" : NumberLong("8589934593")
  },
  "lastmodEpoch" : ObjectId("591279ffefc48e1467398a04"),
  "ns" : "cloud-users.user",
  "min" : {
    "age" : {
      "$minkey" : 1
```

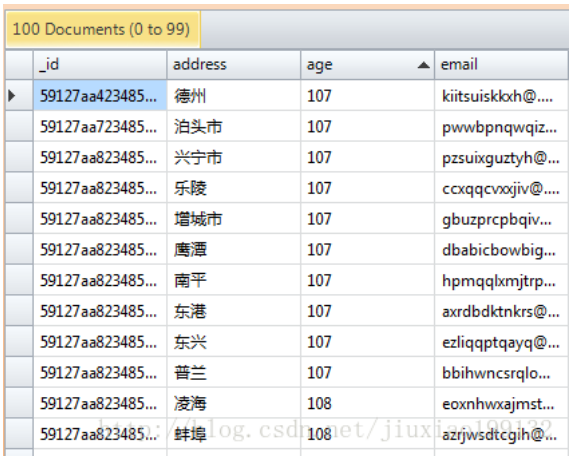
```
    },
    "_id" : {
      "$minkey" : 1
    }
  },
  "max" : {
    "age" : 46,
    "_id" : ObjectId("59127aa423485e1f80fdce60")
  },
  "shard" : "shard-a"
}

/* 1 */
{
  "_id" : "cloud-users.user-age_46_id_ObjectId('59127aa423485e1f80fdce60')",
  "lastmod" : {
    "$timestamp" : NumberLong("4294967298")
  },
  "lastmodEpoch" : ObjectId("591279ffefc48e1467398a04"),
  "ns" : "cloud-users.user",
  "min" : {
    "age" : 46,
    "_id" : ObjectId("59127aa423485e1f80fdce60")
  },
  "max" : {
    "age" : 107,
    "_id" : ObjectId("59127aa423485e1f80fdce5e")
  },
  "shard" : "shard-a"
}

/* 2 */
{
  "_id" : "cloud-users.user-age_107_id_ObjectId('59127aa423485e1f80fdce5e')",
  "lastmod" : {
    "$timestamp" : NumberLong("8589934592")
  },
  "lastmodEpoch" : ObjectId("591279ffefc48e1467398a04"),
  "ns" : "cloud-users.user",
  "min" : {
    "age" : 107,
    "_id" : ObjectId("59127aa423485e1f80fdce5e")
  },
  "max" : {
    "age" : {
      "$maxkey" : 1
    },
    "_id" : {
      "$maxkey" : 1
    }
  },
  "shard" : "shard-b"
}
```

可以看到 age的分布存储情况，然后验证一下结果

打开 40001 也就是 分片B 最小年龄 107



100 Documents (0 to 99)

_id	address	age	email
59127aa423485...	德州	107	kiitsuiskkxh@....
59127aa723485...	泊头市	107	pwwbpnqwqiz...
59127aa823485...	兴宁市	107	pzsuixguztyh@...
59127aa823485...	乐陵	107	ccxqqcvxxjiv@....
59127aa823485...	增城市	107	gbuzprcpbqiv...
59127aa823485...	鹰潭	107	dbabicbowbig...
59127aa823485...	南平	107	hpmqqlxmjtrp...
59127aa823485...	东港	107	axrdbdktnkrs@...
59127aa823485...	东兴	107	ezliqqptqayq@...
59127aa823485...	普兰	107	bbihwncsrqlo...
59127aa823485...	凌海	108	eoxnhwxajmst...
59127aa823485...	蚌埠	108	azrjwsdtcgih@...

选择分片键

选择分片键需要注意三点
1 分布性差: 如果 一千万条数据 80%以上都分布在同一个分片上，那么分片基本上没有任何意义，还不如取消分片.
2 缺乏局部性:
场景：假设用户要上传 100张图片
分片键采取的图片名称的md5值，那么 这个值 完全是随机的，那么写入到数据库完全就是随机写入到不同的分片中，查询的时候也是从各个不同的分片去查找
分片键采取 是用户ID，那么分片键就非常集中，写入和查询也会非常快.
3 无法拆分的块:
场景：假设用户要上传 100w张图片
分片键采取用户ID，那么就会导致 100w条数据完全写入到同一个分片中，会造成单个分片非常巨大.

理想的分片键:
1 将数据均匀分片在每个分片
2 保存 写入，查询操作能够利用局部性
3 有足够的粒度能够拆分块

满足这些要求的分片键 通常由两个字段组成，第一个是粗粒度，第二个是细粒度的.
例如 {username:1,_id:1}

生成环境中的配置

上面例子用了 很多mongodb实例，但是不需要所有的实例都放一个单独的机器上，只需要满足几点即可

1 副本集的每个成员，无论是主节点，从节点，还是仲裁节点，都需要放在不同的机器上
2 副本集的仲裁节点是很轻量级的，可以和其他节点放在一起，不会消耗太多性能
3 配置节点也很轻量，可以和其他节点放在同一个机器上，不会消耗太多性能，但是需要注意 每个配置节点都要分布在不同的机器上
4 每个用于复制的副本集成员都需要有独立的机器

配置注意事项

少数大分片比多数小分片要好 , 比如设定数据量超过50G之后再进行分片