MongoDB笔记

关系型数据库和非关系型数据库的区别：

- 1.数据储存形式与结构
  - 关系型数据库：以表为单位，行和列的形式储存数据
    - 结构：必须要先创建表结构，强结构性更有利于表与表之间的协同工作
  - 非关系型：数据以块的形式存储在本地磁盘
    - 结构：没有强约束性，直接可以将定义好的数据存入
- 2.效率
  - 关系型数据库：效率低下，安全性高
  - 非关系型：效率高，安全性低
- 3.事物
  - 关系型数据库：事物种类多，某些事物级别越高，安全性越高，效率越低
  - 非关系型：可用原则
- 4.可扩展性
  - 关系型数据库：纵向扩展，花钱堆硬件，sql优化也可以提高效率，比较有限
  - 非关系型：本身只需要扩展就是本地的磁盘与内存容量
- 5.可维护性
  - 关系型数据库：维护性强
  - 非关系型数据库：维护性差

MongoDB本身特点：

- 性能高，读写速度快，支持的并发强
- 本身支持读写分离操作和副本集的搭建
- 基于磁盘，操作性更强

- 操作性非常简单
- 基于BSON来储存数据，数据非常轻，增加了读写速度
    - BSON是JSON的一种扩展，binary json，增加了很多json没有的数据类型，byte，date等，一个非常大的json对象，要通过key来获取值，效率相对较低，BOSN在数据的头尾制，将所有的字段信息保存下来，然后分配下标，所以直接查询对应key的时候，可以直接定位到当前key的下标。传输数据的时候，效率比较高，是因为在传输当前数据时，数据的长度已经提前告诉，直接就可以分配空间，不用在把数据遍历到最后才知道当前数据的长度。

推荐Window64版本

- 下载
- 安装

命令简介：

- 开启服务 mongod –dbpath 数据文件夹 –port 27017
- 开启客户端 mongo localhost:10000
- 连接成功后，操作当前的数据库
    - 客户端中使用mongoDB命令来操作数据库
        - show dbs; 显示当前多有的数据库
        - use 数据库名称；
        - show tables(collections) ;查看当前数据下的所有数据,显示所有的表
        - 查看表下的所有数据 db.表名.find()
        - 在当前数据库下创建表 db.表名.insert() 默认创建表，而且每一条插入的数据会自动生成_id

表操作：

- 查
    - 查询所有当前表下的数据 db.表名.find()
    - 条件查询 db.表名.find({key:value})

    - 模糊查询 db.表名.find({key://}) 正则表达式
    - 范围查询 db.admin.find({age:{$gte:18}}) $gte 大于等于 $lte小于等于
    - 分页查询 skip() 从第几条开始查 limit()，一次查询几条数据
    - 排序 .sort({key:-1}) -1倒叙 1正序排列

- 增
    - db.表名.insert({})

- 删
    - db.表名.remove({}) 与find中条件匹配规则相同

- 改
    - db.表名.update({},{$set:{}},upsert,multi)
        - 参数1 匹配需要修改的数据条件
        - 参数2 需要修改的信息内容
        - upsert是否匹配不到的数据进行插入 false
        - multi匹配到多条符合条件的数据是否全部修改,如果设置成false，只修改匹配数据的第一条

Doument

- 所有插入到当前表中的每一条数据都是一个Doument，大小不能超过4M-8M-16M，一旦存储文件，当前的Doument就不在建议使用，建议使用 GridFS，专门为保存文件提出的新功能，将文件分布式储存在mongoDB，将文件拆分成若干份，保存在MongoDB中，每一份大小不超过255K
    - 使用GridFS上传文件
    - 调用 mongofiles.exe -d 数据库名称 put 文件 –port 10000
    - 文件被保存在指定数据库下的 fs.files 与fs.chunks表中
        - fs.files 记录文件基本信息
        - fs.chunks存放文件流信息，通过分布式算法，当前超过255k的文件，会被分成若干份，fs.files中的_id可以将fs.chunks所有该文件的流信息全部关联在一起，形成完整的文件。

db.fs.chunks.find({files_id:ObjectId(“”)})，查询出来的流信息，保存在本地就是我们上传的那个完整的文件。

mongoDB主从复制读写分离：

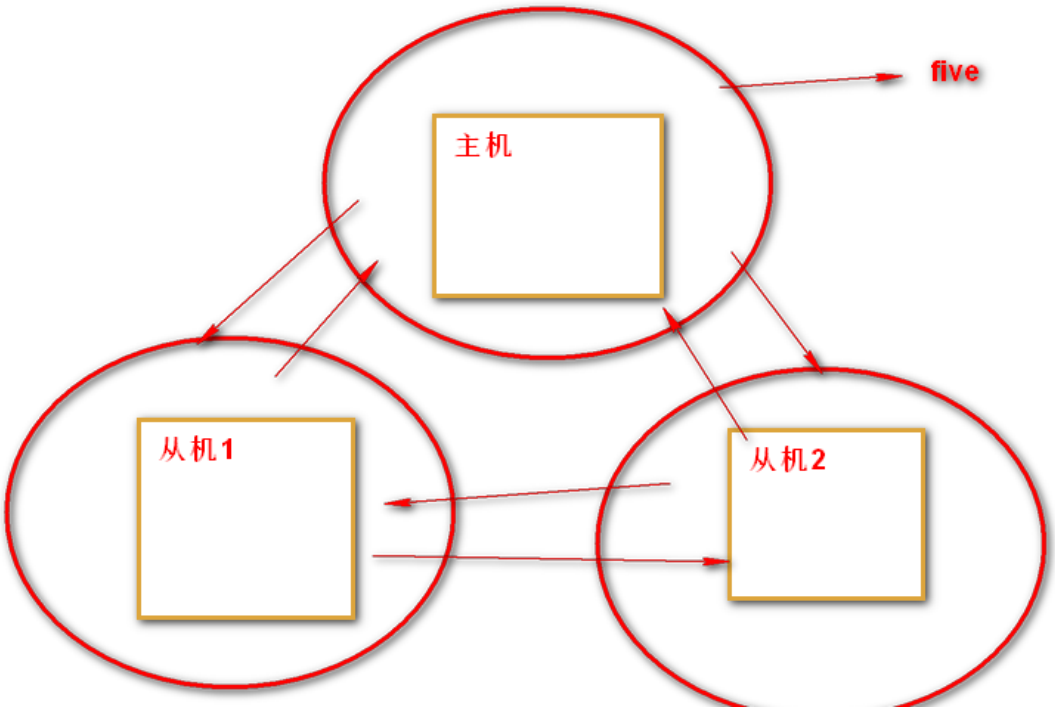
- 什么是主从复制？主机负责写操作，从机要实时监听主机，从而立即备份主机的所有新数据。
    - 开启三个服务，指定一个主服务，两个从服务，从服务实时监听主服务，一旦有数据的变更，立即更新和备份到从机的数据库中。
    - 开启主机
        - mongod –dbpath 文件夹 –port 10000 –master
    - 从机
        - mongod –dbpath 文件夹 –port 20000 –slave –source ip:端口号
- 注意默认访问从机，是拒绝的，如果要读写分离。
    - 主机只负责写操作，读操作交给从机，因为从机的数据和主机同步备份，数据是一致的，所以直接访问从机和直接访问主机效果是一样的，如果把所有读写操作都交给主机，主机压力很大，这样我们就直接把所有读操作，分配给从机，主机只负责写，减轻主机压力，提高了用户获取数据的速度。默认从机不允许访问，需要调用rs.slaveOk()

副本集：

- mongodb的副本集，拥有主从复制功能，同时也可以故障转移。
    - mongod –dbpath f:/data1 –port 10000 –replSet 节点名称
        - replSet 设置一个节点名称
    - 手动将所有当前节点下的服务关联在一起
        - 创建一个对象，该对象记录了所有指定节点下的所有服务进程

- rs.initiate(five_baizhi);调用该方法，传入创建好的对象，返回ok证明关联成功

```
{
        "_id" : "five",
        "members" : [
                {
                        "_id" : 0,
                        "host" : "localhost:10000"
                },
                {
                        "_id" : 1,
                        "host" : "localhost:20000"
                },
                {
                        "_id" : 2,
                        "host" : "localhost:30000"
                }
        ]
}
```



- 可视化工具 admin-mongo

- 解压后直接进入目录 npm i

mongoose@4.11

- 使用步骤
  - 1.下载
  - 2.引入
  - 3.创建连接信息
  - 4.连接生效
  - 5.监听连接是否成功
  - 6.创建数据骨架，指定名表，不会加s，在服务端进行查询出来的数据操作时，必须骨架中有对应的key才能获取，客户端渲染没有这个问题
  - 7.利用骨架创建model model具备直接操作表的能力
  - 8.增删改查语法

- 连接池配置—{server:{poolSize:5}}
- mongoose与express的封装，将所有表对象放在全局对象当中后，路由可以直接使用

mysql@2.15

https://www.npmjs.com/package/mysql

```
var mysql      = require('mysql');
var connection = mysql.createConnection({
  host     : 'localhost',
  user     : 'me',
  password : 'secret',
  database : 'my_db',
  port:'3306'
});
```

```
connection.connect();

connection.query('SELECT 1 + 1 AS solution', functi
on (error, results, fields) {
  if (error) throw error;
  console.log('The solution is: ', results[0].solut
ion);
});

connection.end();
```

- 连接对象调用query方法，只需关注对应的sql语句和参数的书写，参数必须与?一一对应

连接池，连接交给连接池管理，省略了连接创建关闭的步骤，提高访问速度。

```
const pool = mysql.createPool({
    host: "localhost",
    user: "root",
    password: "1234",
    database: "testdata",
    port: "3306",
    connectionLimit: 10
})
```

跨域

什么时候出现跨域??由于浏览器同源策略导致不通域的请求默认情况下不能互相访问。

协议+IP+端口必须一样才属于一个域，一个域下的访问没有跨域问题
如何解决跨域问题：

- 手动的设置响应头，让当前的跨域请求可以得到响应，但是比较麻烦
- 使用第三方中间件解决跨域问题 cors
    - npm i cors
    - app.use(cors())

- jsonp解决方案 script标签的请求 没有跨域限制
  动态添加一个script标签，而script标签的src属性是没有跨域的限制的。 和img的src一样可以不受限制从其他域加载资源。我们不能跨域请求数据，但是可以引入不同域的脚本文件。 所以我们JSONP中，src是我们请求服务器的url，一般是这样的形式

URL	说明	是否跨域
http://www.a.com/a.js http://www.a.com/b.js	同一个域名	否
http://www.a.com/lab/a.js http://www.a.com/script/b.js	同一个域名，不同文件夹	否
http://www.a.com:8080/a.js http://www.a.com/b.js	同一个域名，不同端口	是
http://www.a.com/a.js http://79.23.1.21/b.js	域名与域名对应IP	是
http://www.a.com/a.js http://script.a.com/b.js	主域和子域	是
http://www.a.com/a.js http://a.com/b.js	一级域名和二级域名	是
http://www.XXX.com/a.js http://www.a.com/b.js	不同域名	是
http://localhost/a.js http://127.0.0.1/b.js	本地和本地对应的IP	是