

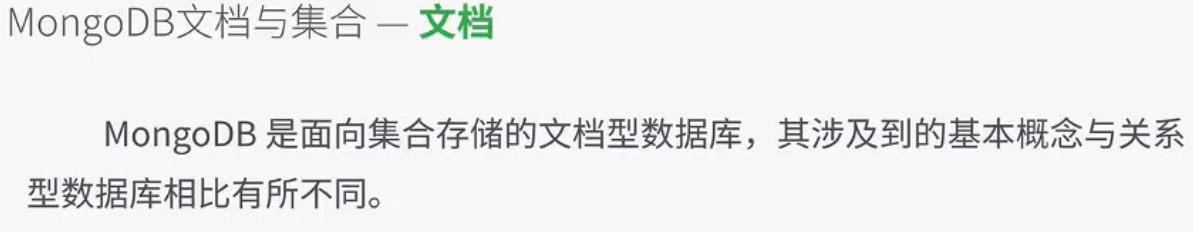
MongoDB的数据模型



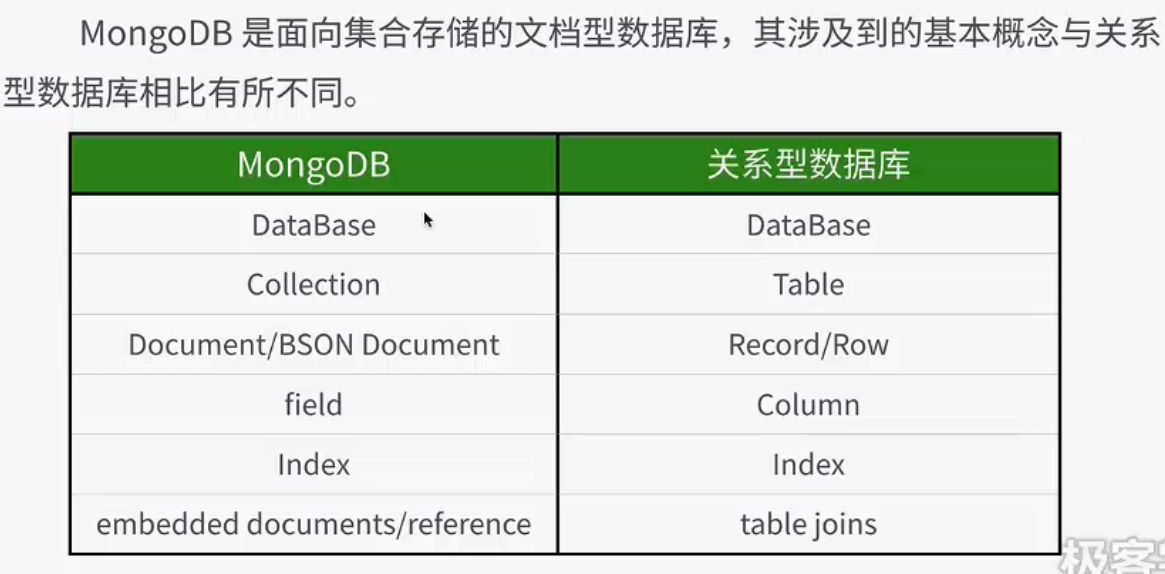
MongoDB的文档和集合







MongoDB是基于文档的非关系型数据库



集合和数据库表等价

横向的document和row等价

纵向的field和Column等价

Mongo中的索引和关系型数据库的索引的作用是一样的 都是为了加快表的查询

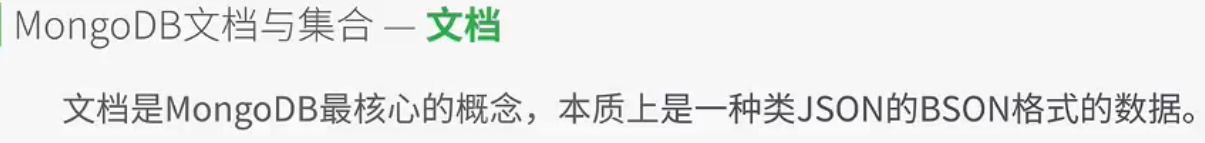
由于关系型数据库之间满足范式的要求 所以 表之间的查询存在表关联

在Mongo中 使用内嵌文档或者表引用 表引用和关系型数据库相类型

---- 关系型数据库中的表关联 必须首先创建相应的表结构 并且指明两个表之间关联的字段

不同的是 mongo集合 不需要定义表结构

---- 如果存在被参照表的话 如果进行了更新 那么会对参照的表进行参照观察性检查 但是 mongo中不会 数据表之间的关系 完全靠客户端进行维持



MongoDB的存储是文档格式 文档是mongoDB最小的集合单位

MongoDB的CRUD操作 索引管理 集群管理 分片集群管理都会使用文档

文档是

Bson全称是binary serialized document format 是一种类json格式的数据

简称 binary json 支持内嵌的文档对象 和 数组对象



Bson常被用作数据存储 和 网络数据交换的格式

类似于GOOgle的 protocolbuffer

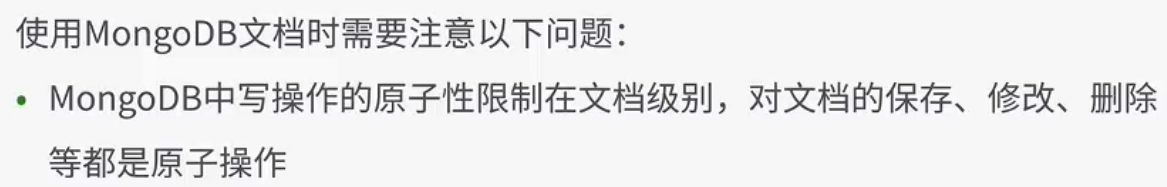
Bson是无模式的数据存储形式 protocolbuffer是由protocal来描述文件的限制

所以 bson的灵活性更高

缺点是 存在KV名字的冗余信息







也就是一个单独的写操作 不可能影响多个文档 除此以外提供了很多原子性的操作 对文档的保存修改 删除

MongoDB不支持事务 所以项目中 不要要求mongodb 有事务的特性

但是 由于文档中可以内嵌文档 而一个写操作 就限制在文档级别 所以 我们的可以使用这个特点对文档和内嵌文档达到类似于事务的特点

如果使用引用方式的话 如果进行了更新 那么 另一个引用的文档就没有办法保持数据的原子性



和其他的数据库一样 存在单个文档 新版本是16MB 以前是限制在4MB以内

这样避免单个文档过大 导致占用的网络带宽过大

如果文档大小超过16M 很可能是你的表结构设计有问题 ---- 这样 可以使用gird fs来存储文件

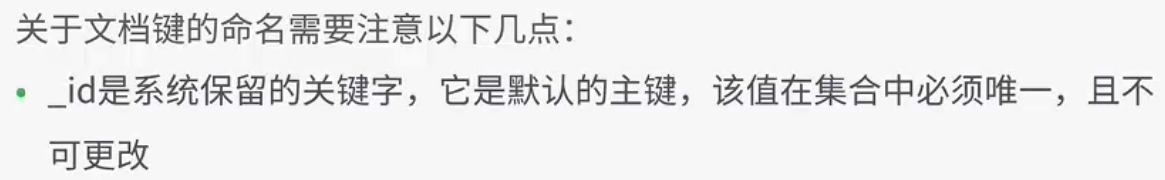
从2.6开始



更新操作会改变的字段的顺序 因为更新操作会改变文档内存的重新分配

\_id标示的字段 永远文档第一的位置

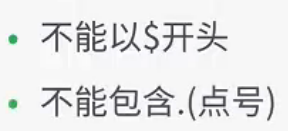
可以忽略mongo对字段顺序的保持 我们写的程序不要依赖于数据库中键值对的顺序



MongoDB的\_id类型的值 是ObjectId类型

Mongo可以保证 在分布式环境下 生成的ObjectId类型的值具有唯一性

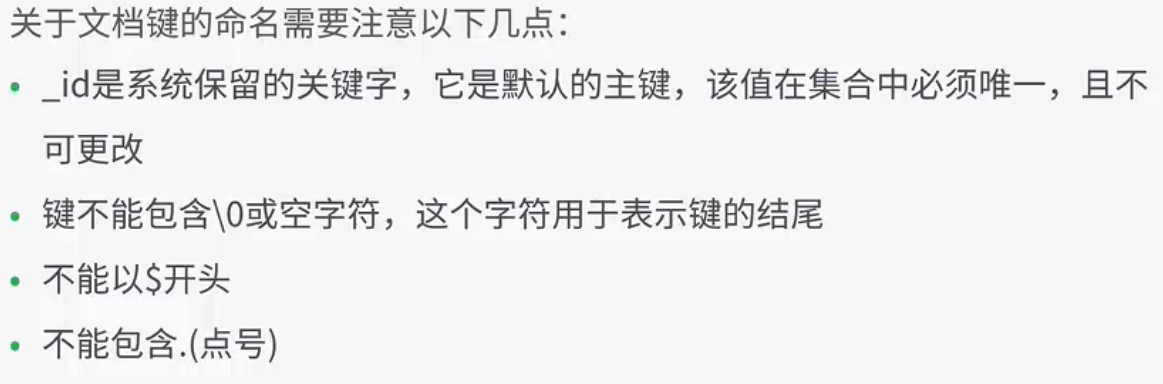


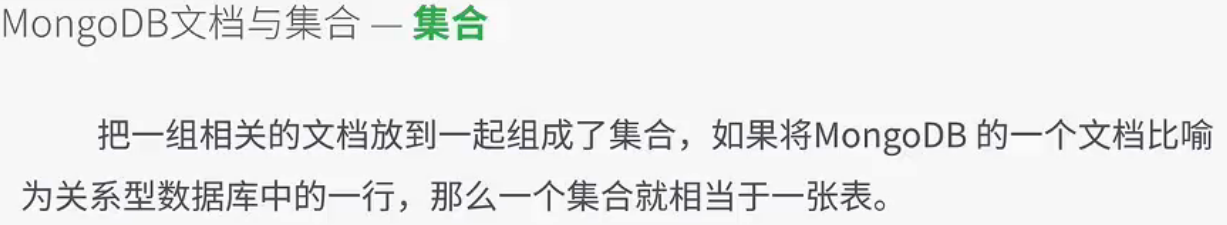


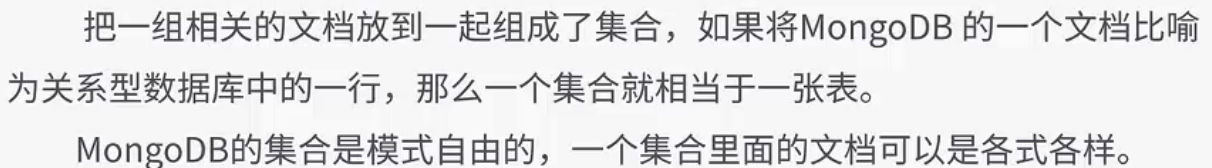
$用于表示操作符 .用于命名空间



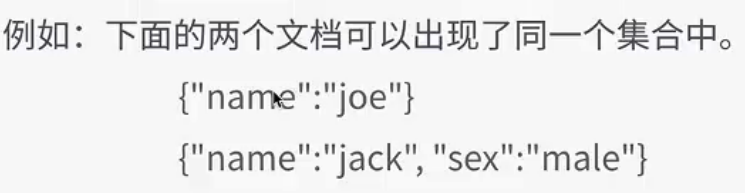
首字母的大小写不同 键名区分大小写 并且不能重复







一个集合中 文档中的格式是不同的



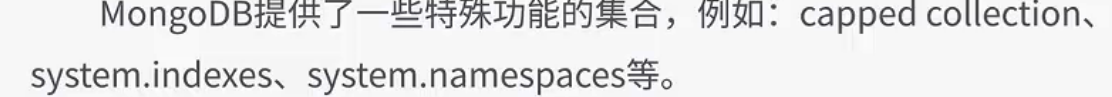
这是mongoDB模式自由决定的

过多使用模式自由的特点 对于开发和管理 都是噩梦

这样 就无法使用索引功能

开发阶段充分利用模式自由的特点 ---- 尝试各种类型的开发模式

生产阶段 使用固定的字段



Capped Colletion是固定集合 拥有固定的大小 集合要提前创建出来

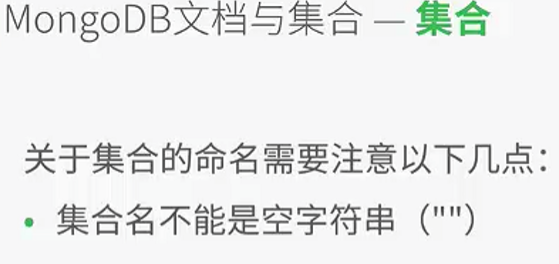
普通的集合的大小是插入第一条文档之的时候 集合才会被数据库自动创建 ----- 固定集合就是相当于一个循环队列 当集合空间用完时候 新的文档就会替换掉老的文档

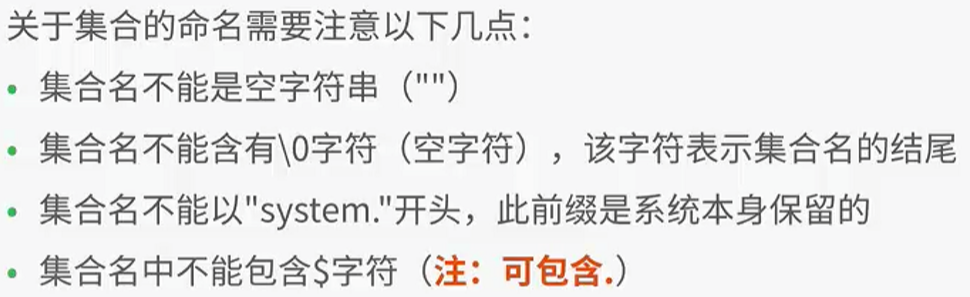
固定集合拥有很高的性能 并可以自动维护对象的插入顺序 非常适合类似记录日志的功能 ，mongoDB的复制集日志就是使用固定集合



System.indexes是用于存储索引信息 system.namespaces是用于存储命名空间信息

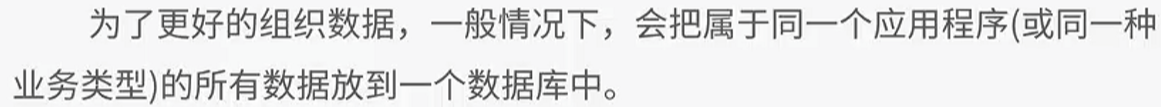
集合的命名





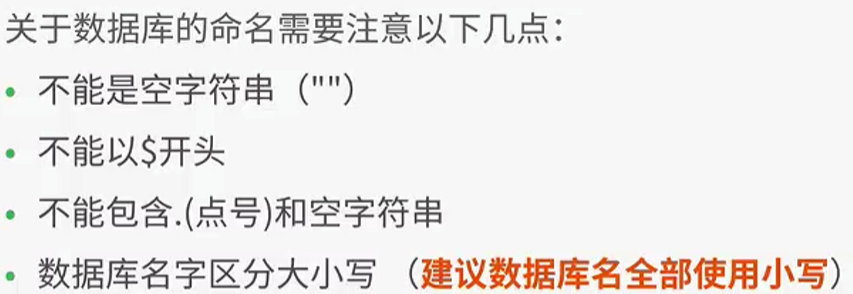
.用于命名子集合



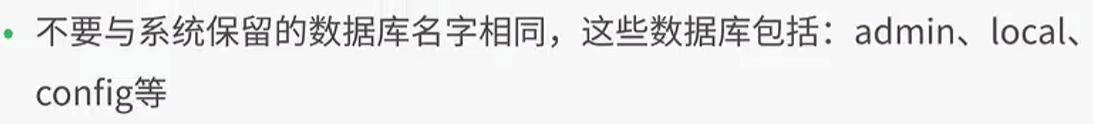


多个文档 ->集合 ----- 多个集合 ->数据库

数据库的命名规则



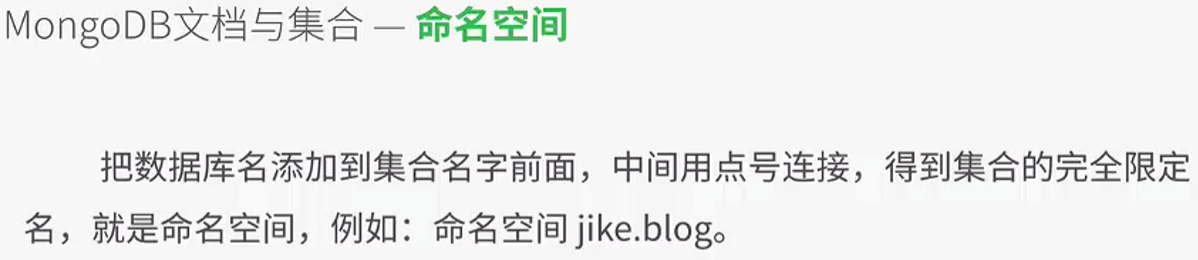
建议数据库名为小写



Admin是身份验证的root数据库 如果一个用户添加到admin数据库 那么这个用户将获得所有数据库的访问权限

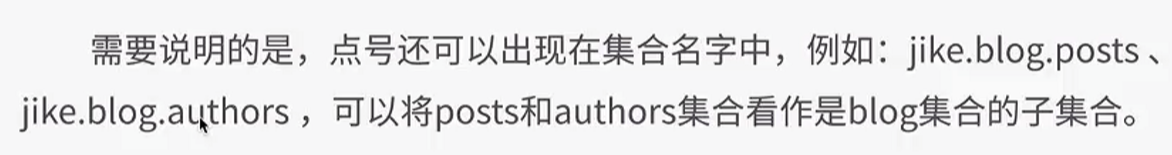
Local 存储了元数据信息

Config用于分片集群 以及集群状态更改的日志数据



Jike是数据库的名字 blog是集合的名字

.可以出现在集合名字中 下面的两个例子



Blog集合可以不存在

mongoDB使用的是这种命名方式 grid fs使用的是 fs.chuncks保存数据块信息 使用fs.files保存的是文件的元数据信息

这样的命名方式可以使得数据的结构更加清晰

命名空间的长度不可以超过100个字节







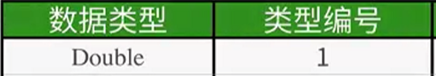
Json支持六种数据类型 分别是 **数字类型**--- json只有一种数字类型 无法区分浮点数 **字符串类**---- 双引号来表示 **逻辑值类型** **数组类型**---[]表示 **对象类型**---{}表示 空值null

如果作为数据交换格式 足够了 但是 作为数据库的基本数据类型 表现力是远远不够的

Bson具有更好的便利性



对于数字类型 进行了划分 浮点数 32位类型的整数 64位类型的证书







还添加了Date 正则表达式类型等

所以MongoDB的数据类型非常丰富 并且每一种类型 有类型编号

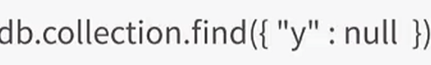
这个就是查询collection这个集合中 name字段的数据类型的编号是2的所有文档

数据类型的编号是2对应的类型是String类型 那么这条语句的意思就是 查询collection集合中 name字段是String类型的文档

---- 可以过滤非法格式的文档



Null可以表示值为null 或者 值不存在

 就表示返回字段y的值是null 和 值不存在的文档





注意 使用MongoDB shell的数字 都是64位浮点数

因为MongoDB的shell是JavaScript的shell 而js仅仅支持64位浮点数

所以 当我们使用 MongoDB shell 插入到数据库中的时候 插入的实际上是浮点数

如果想插入整形  ---- 保存为32位整数

使用NumberLong保存为64位整数

例如



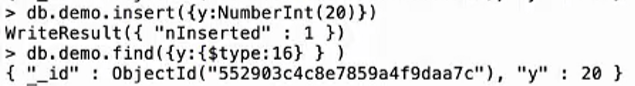
现在查询y字段的类型是double的集合 double对应的数据类型是1

Db.demo.find({“y”:{$type:1}); ----- 查询的结果是



说明刚才直接输入的10 插入到Mongo中的数据类型实际上是double

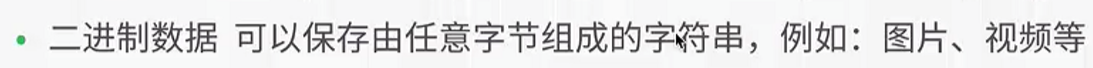
这次使用numberint插入



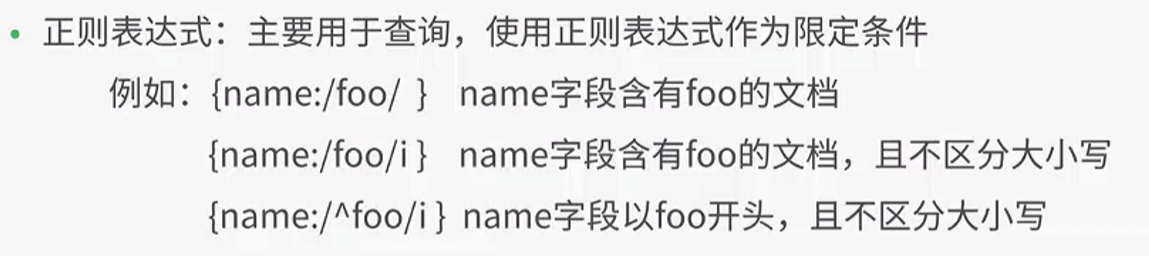
这个存储为32位的整数



表示方法与json一样 使用双引号标记出来





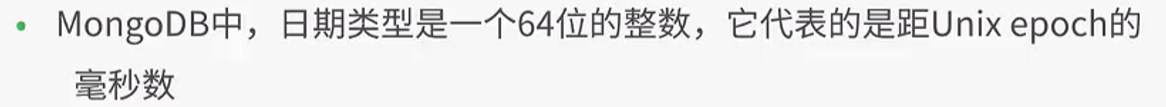




这样 常用的js脚本 就可以存储到mongDB中 这样就可以将经常使用js脚本存储到数据库中 方便使用

这个存储的js的函数

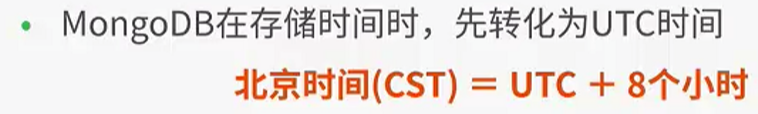
**日期类型**



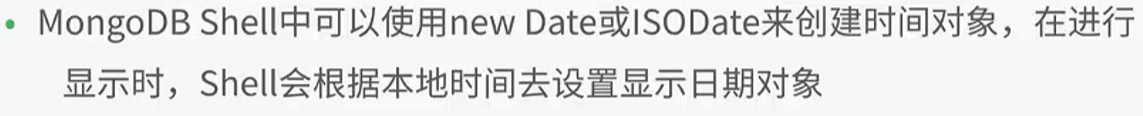
距离1970年1月1日的毫秒数

[存储的时候 先转化]

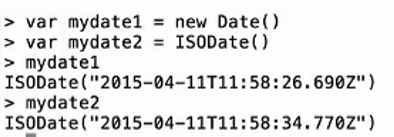
又称**协调世界时世界标准时间**



MongoDB中



创建两个时间对象



显示的都是UTC时间

字符串显示



这样显示的是CST格式的时间 --- 所以 shell会依据本地时间显示对象

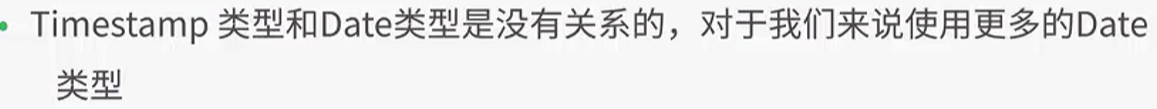
**时间戳类型**



64位组成



复制集中的ts字段就是记录这个时间



 尽量不用

ObjectId类型



每个字节存储两位十六进制的数字 ----- 一共有24位十六进制的数字 那么 需要24/2个字节

ObjectId可以保证在分布式环境唯一 下面给出了12个字节隐含存储的信息



 ---- 前4个字节（一个字节存储两位 所以 这里面存储了8位的信息）Unix开始的时间戳 单位为秒

这样ObjectId也包含了文档创建时间的信息

所以 我们可以使用getTimestamp()来从ObjectId中提取时间戳信息【从ObjectId中提取timestamp】

下面三个字节 是机器的标示 --- 是机器主机名的散列值 保证不同主机生成的机器标示不一样 不会冲突

这样就可以保证同一秒内 是一致的

后面两个字节是PID 来自进程的进程的

前面七个字节就可以保证 同一秒 不同主机 生成的objectId是不一样的

后面两个字节存储的是PID 存储的是进程标示符 ---- 可以保证同一台机器上 并发的不同的进程是唯一的

这样前九个字节 同一秒 不能机器的不同进程产生的ObjectId是唯一的

最后三个字节是自动增加计数器加上去的 是 保证同一秒 同一台机器 的 同一个进程产生的多个ObjectId是唯一的

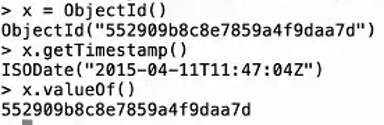
所以 通过以上12个字节 分布式环境生成的标示是一致的 这在分布式环境中 是十分重要的 这个ObjectID设计是轻量型的 生成速度非常快 可以由MongoDB客户端生成 也可以由服务端生成 ------ 感兴趣可以看一下mongodb java客户端生成ObjectId的代码

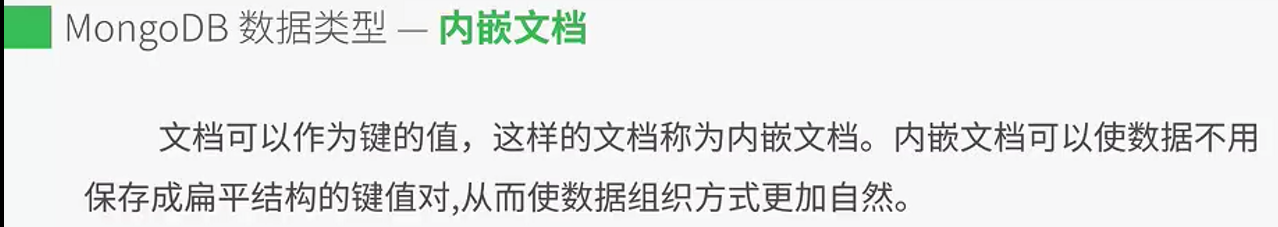


ObjectId()用于生成ObjectId

getTimestamp()用于从ObjectId中获取前面的时间戳 返回值类型是date

valueOf()用于取得ObjectId()的字符串表示





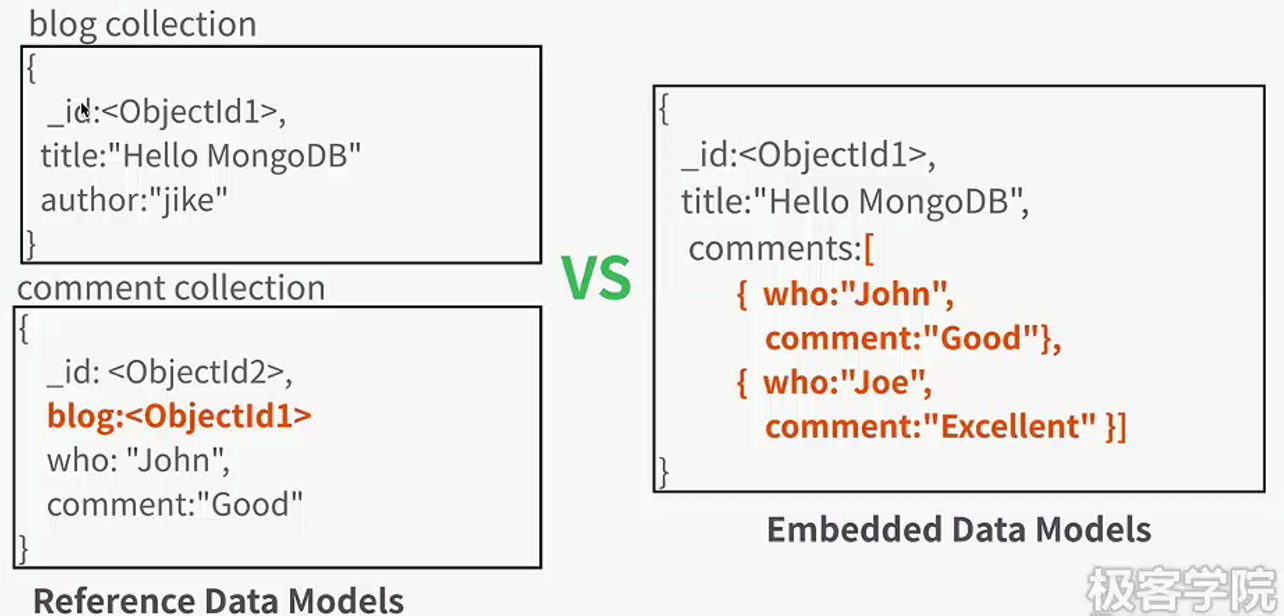
可以非常复杂的数据结构



评论使用嵌入文档表示

如果是关系型数据库使用引用型 那么就要分成两个表

现在对Mongo中使用内嵌文档和引用文档进行对比



这两个什么场合引用？

嵌套文档模式 ---- 扁平化 所以 信息表达自然

---- 读取速度变快 因为仅仅读取一次

---- 如果子文档的数据量大的时候 整个文档的加载速度变慢 影响性能效率

---- 如果子文档更新频繁的话 会严重影响性能和效率