MongoDB

[1 概述 1](#_Toc469730004)

[2 安装 1](#_Toc469730005)

[2.1 windows 1](#_Toc469730006)

[2.2 linux 1](#_Toc469730007)

[3 C/S架构 1](#_Toc469730008)

[3.1 服务器端 bin/mongod 1](#_Toc469730009)

[3.1.1 创建配置文件 conf/mongod.ini 1](#_Toc469730010)

[3.1.2 指定配置, 开启mongodb 1](#_Toc469730011)

[3.1.3 配置环境变量 1](#_Toc469730012)

[3.1.4 在rc.loca增加自动启动 1](#_Toc469730013)

[3.2 mongo客户端 1](#_Toc469730014)

[3.2.1 命令行 1](#_Toc469730015)

[3.2.2 PHP程序 1](#_Toc469730016)

[4 操作 1](#_Toc469730017)

[4.1 文档型数据库(相对于: 关系型数据库) 1](#_Toc469730018)

[4.1.1 db数据库(相对于: 数据库) 1](#_Toc469730019)

[4.1.2 collection集合(相对于: 表) 1](#_Toc469730020)

[4.1.3 document文档(相对于: 记录) 1](#_Toc469730021)

[4.1.4 property,field属性(相对于: 字段) 1](#_Toc469730022)

[4.1.5 文档型不要求同一集合中的文档结构一致 1](#_Toc469730023)

[4.2 BSON, Binary - Json 1](#_Toc469730024)

[4.2.1 文档就是json语法 1](#_Toc469730025)

[4.2.2 javascript Shell, javascript语法解释器 1](#_Toc469730026)

[4.2.3 Bson, 可执行的JSON 1](#_Toc469730027)

[4.2.4 PHP处理BSON 1](#_Toc469730028)

[4.3 无状态 1](#_Toc469730029)

[4.4 实例化mongo客户端对象 1](#_Toc469730030)

[4.5 访问数据库和集合 1](#_Toc469730031)

[4.6 mongodb数据类型的支持 1](#_Toc469730032)

[4.7 集合中文档的CRUD 1](#_Toc469730033)

[4.7.1 insert(), 增加 1](#_Toc469730034)

[4.7.2 find(), 查看 1](#_Toc469730035)

[4.7.3 update()修改 1](#_Toc469730036)

[4.7.4 remove, 删除 1](#_Toc469730037)

[4.8 条件 1](#_Toc469730038)

[4.8.1 关系运算 1](#_Toc469730039)

[4.8.2 逻辑运算 1](#_Toc469730040)

[4.8.3 数组类型运算 1](#_Toc469730041)

[4.8.4 对象型运算, 内联文档, 内嵌文档 1](#_Toc469730042)

[4.8.5 正则条件 1](#_Toc469730043)

[4.9 查询, find() 1](#_Toc469730044)

[4.9.1 参数1:　条件 1](#_Toc469730045)

[4.9.2 参数2: 　需要的字段 1](#_Toc469730046)

[4.9.3 \_id 1](#_Toc469730047)

[4.9.4 排序, $cursor->sort() 1](#_Toc469730048)

[4.9.5 文档偏移, $cursor->skip($n) 1](#_Toc469730049)

[4.9.6 限定文档数据量, $cursor->limit($n) 1](#_Toc469730050)

[4.10 分组统计 1](#_Toc469730051)

[4.10.1 PHP实现 1](#_Toc469730052)

[4.11 更新 1](#_Toc469730053)

[4.11.1 修改器更新 1](#_Toc469730054)

[4.11.2 多条处理 1](#_Toc469730055)

[4.11.3 upsert, update || insert 1](#_Toc469730056)

[5 索引 1](#_Toc469730057)

[5.1 管理索引语法 1](#_Toc469730058)

[5.1.1 创建索引: collection.ensureIndex() 1](#_Toc469730059)

[5.1.2 查看集合存在的索引:collection.getIndexes() 1](#_Toc469730060)

[5.1.3 删除索引: collection.dropIndex(index-name) 1](#_Toc469730061)

[5.2 索引类型 1](#_Toc469730062)

[5.3 2d索引 1](#_Toc469730063)

[5.3.1 测试数据 1](#_Toc469730064)

[5.3.2 创建索引 1](#_Toc469730065)

[5.3.3 利用索引检索 1](#_Toc469730066)

# 概述

非关系型数据库NOSQL的代表作.

文档型的NOSQL(document). 有点类似于关系型.

擅长的领域: 文档存储.

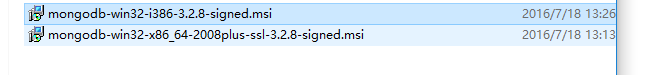
例如, 日志系统, 新闻系统, 同志报告信息. 博客系统.

不擅长做复杂的业务逻辑:

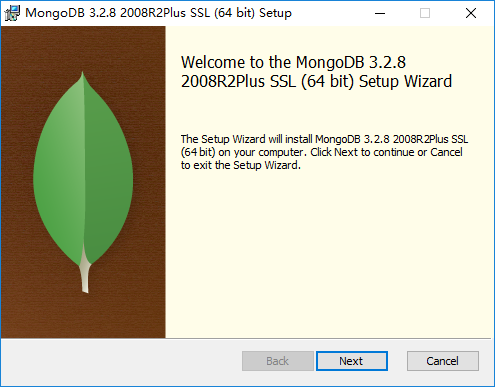
做事务, 做实体间的关系. OA办公自动化, ERP企业资源管理, CRM客户关系管理, 电子商城…

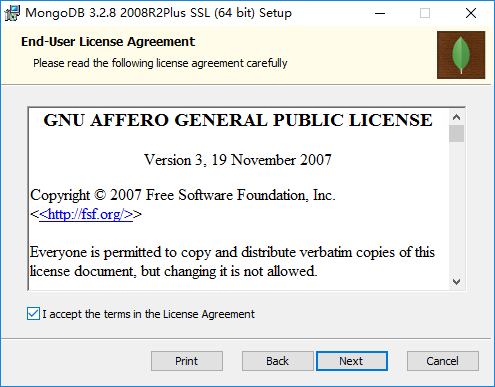
# 安装

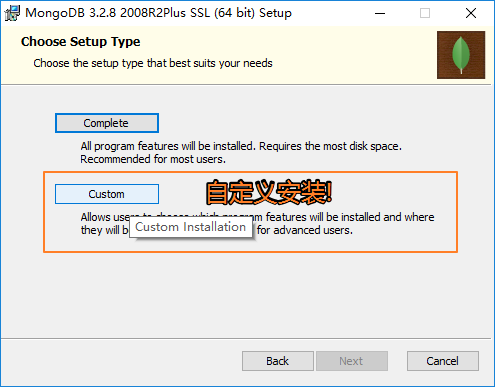
## windows

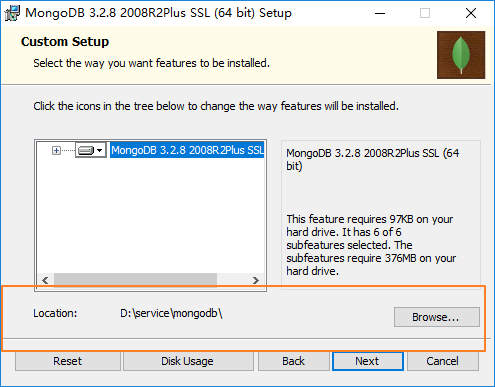


双击运行安装



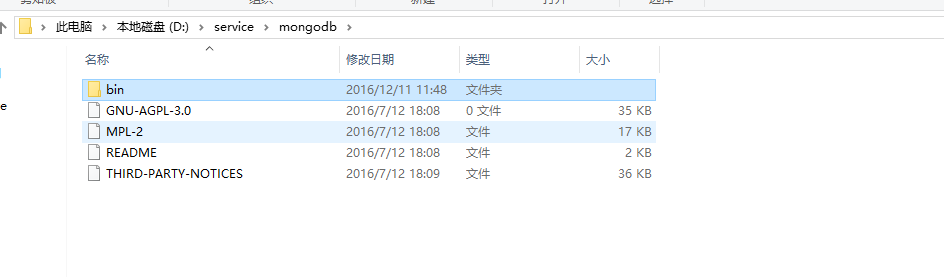




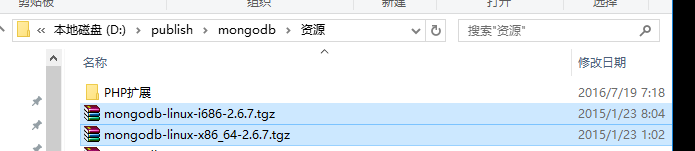




结果



## linux



上传到linux

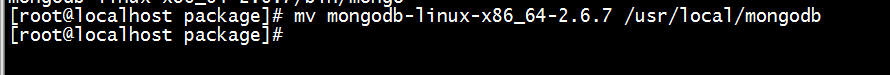


解压到指定目录

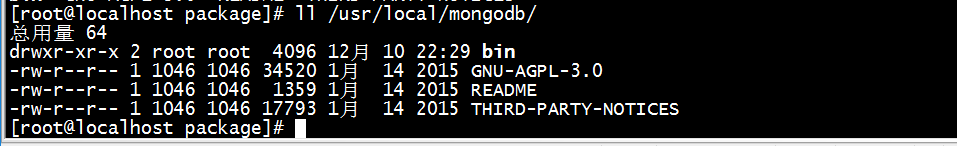


提供了而执行的执行版, 不需要make 安装

将解压好的内容, 移动到需要位置即可: /user/local/mongodb



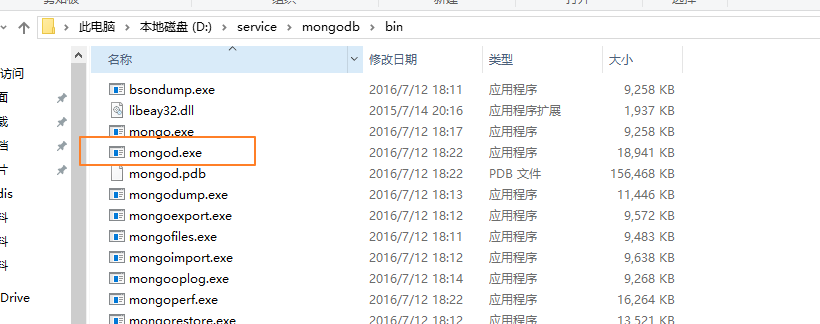
安装结果

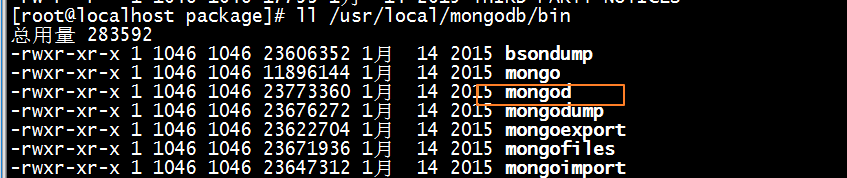


# C/S架构

## 服务器端 bin/mongod

bin/mongod 进程, 就是提供好的服务器端进程!



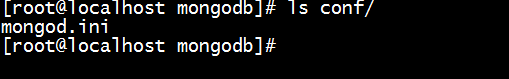


在shell(CMD)即可启动

启动前, 要增加一个配置文件, 便于管理服务器端程序

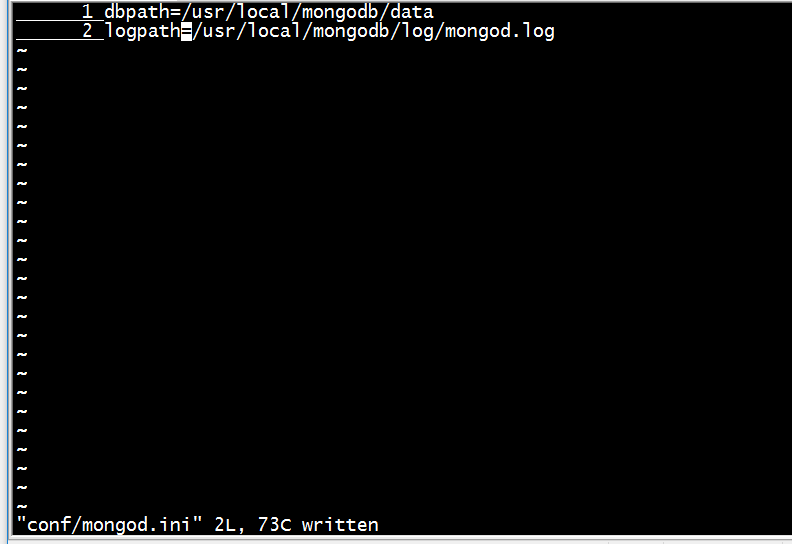
### 创建配置文件 conf/mongod.ini

任意指定的配置文件

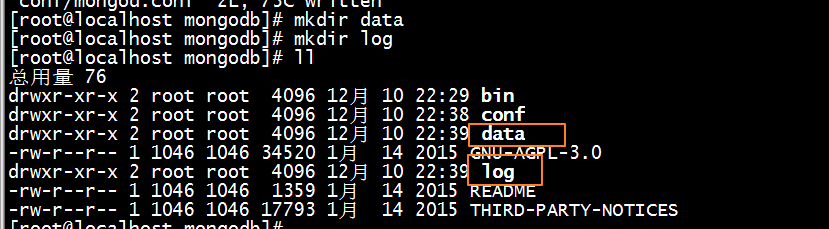


增加必要的配置

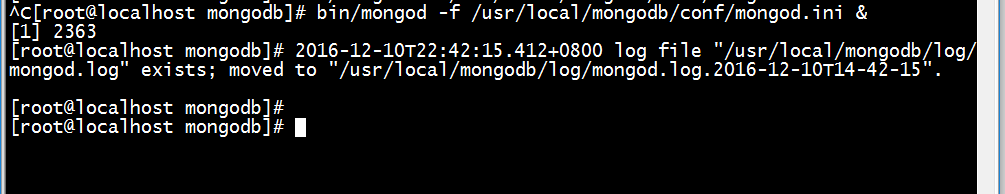
数据文件位置(目录), 日志文件位置(文件)

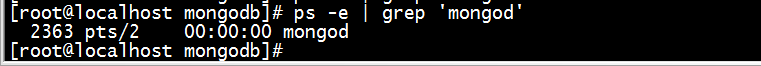


创建对应的目录



### 指定配置, 开启mongodb





windows也是如此!

### 配置环境变量

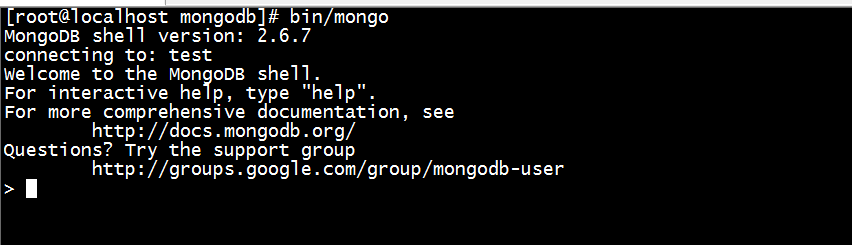
### 在rc.loca增加自动启动



## mongo客户端

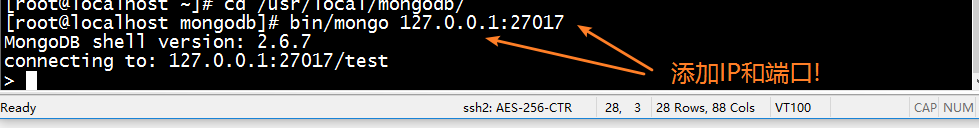
### 命令行

bin/mongo



exit可以退出

开启mongo客户端时, 如果需要指定特定的服务器, 需要携带:

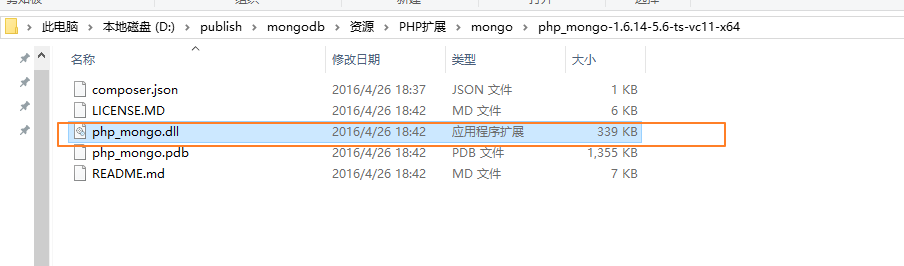


通常用于管理和测试!

### PHP程序

安装 PHP的mongo扩展

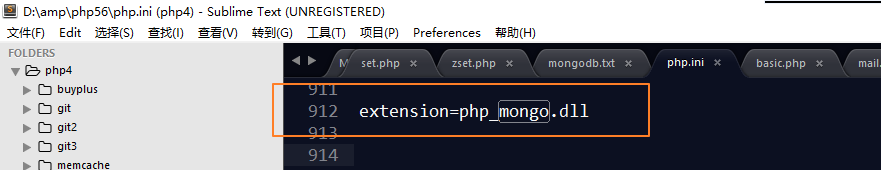
#### windows



PHP/ext



php.ini

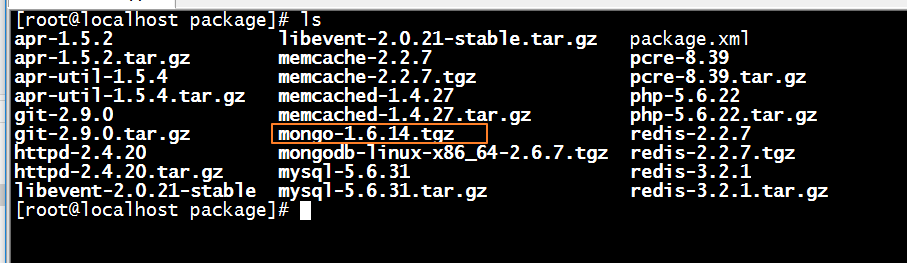


apache restart

结果

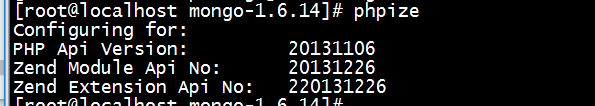


#### linux



解压, 进入

phpize

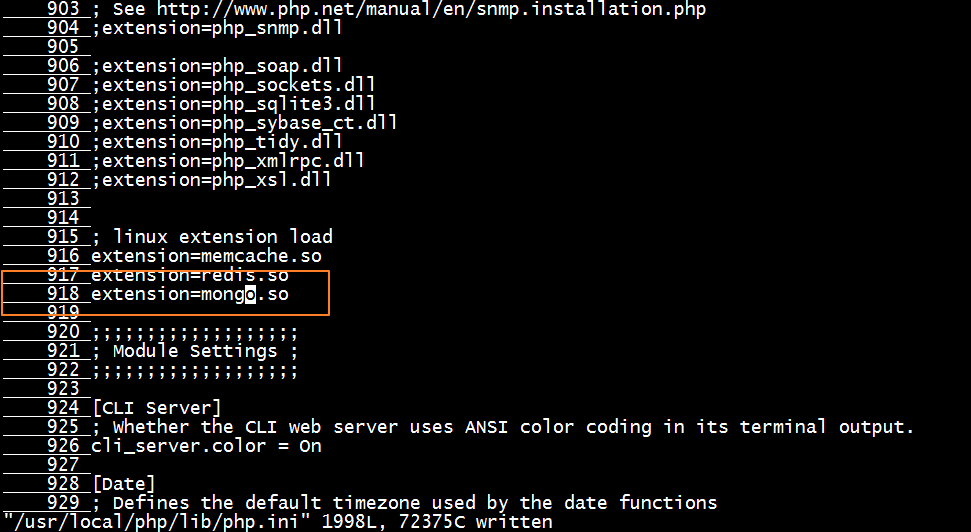


配置, 编译, 安装

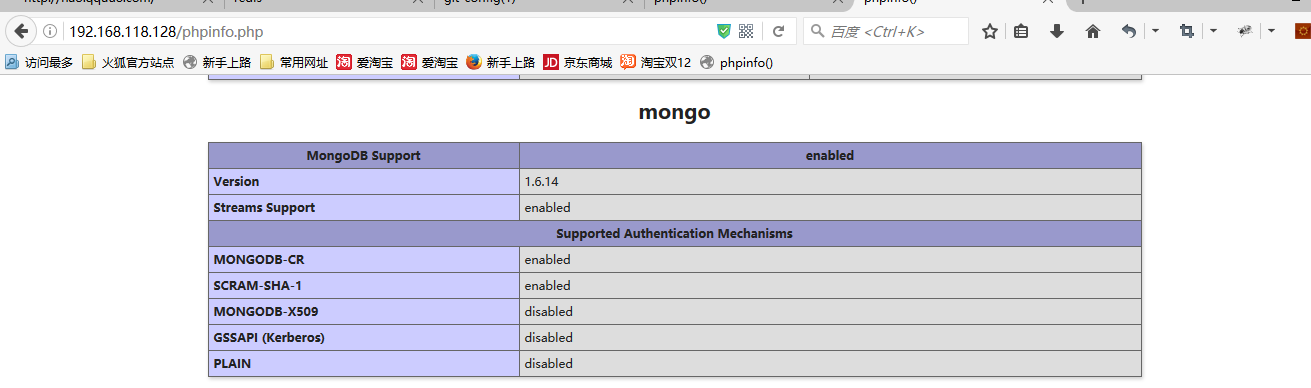




php.ini



apache 重启, 结果



# 操作

## 文档型数据库(相对于: 关系型数据库)

### db数据库(相对于: 数据库)

集合是存储在数据库中!

### collection集合(相对于: 表)

文档使用集合进行整理的.

多篇文档, 集中存储在一个集合中.

### document文档(相对于: 记录)

文档型数据库中, 存储的数据, 以文档为基本单位(结构).

### property,field属性(相对于: 字段)

文档由各种属性组成.

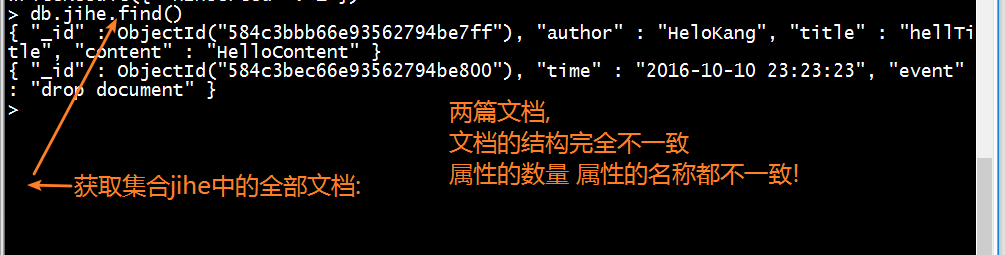
例如, 一篇文章, 有作者, 标题, 内容, 三个属性组成

一条日志, 由, 时间, 事件, 用户 组成

### 文档型不要求同一集合中的文档结构一致

在关系型数据库系统中, 一个表中的记录的结构完全一致, 都是由固定的字段组成的.

但是在文档型中, 集合仅仅是将多个文档整理在一起, 而不去要求文档的结构一定要一致. 例如:



理解:

将集合, 当作某个目录来看, 集合中的文档, 就是目录中的文件.

## BSON, Binary - Json

### 文档就是json语法

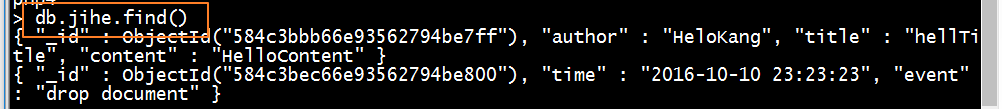
mongodb中, 操作文档, 就是在操作一个josn语法格式的数据!



### javascript Shell, javascript语法解释器

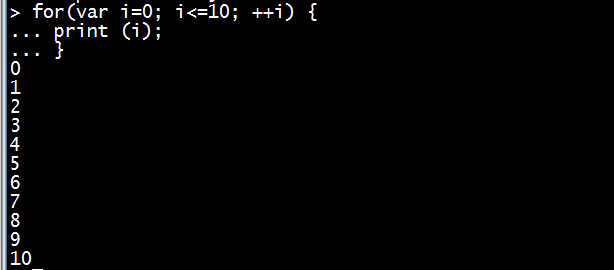
当前的基于命令行的mongo的客户端, 提供的一套javascript的语法解析支持:

db对象的jihe属性的find()方法



支持你学习过的JS的大多数语法:

例如for循环!



浏览器是javascript shell.

只要有解释器(虚拟机, 运行语言的基础环境, PHP的叫ZendEngine), 就可以运行相应的语言.

NodeJs(服务器端的js), 也是一套JS的运行环境. 提供了一些特定的功能.

### Bson, 可执行的JSON

基于以上两点.

1, js可以在mongo执行. 2, 数据都是JSON.

mongodb,提出: Bson 可执行的JSON

### PHP处理BSON

js中的json: 可以理解成: 键值对集合.

PHP中的, 关联数组,就是键值对集合.

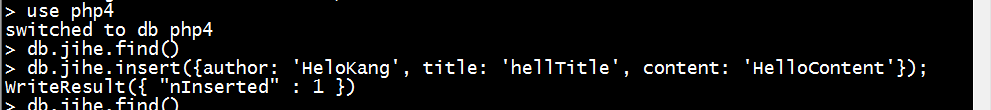
操作mongodb时, 使用的数据都是 关联数组的语法. 可以逻辑上转换为 JSON语法.

## 无状态

数据库, 集合 是无状态的

即时在数据库和集合不存在的情况下, 可以直接操作.

一旦产生了实质性的操作. 增加了数据.

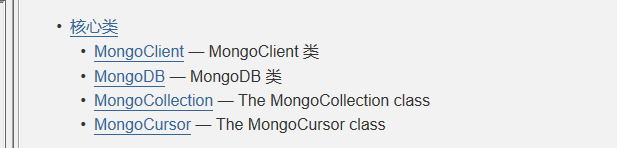


insert时 自动创建集合的数据库!

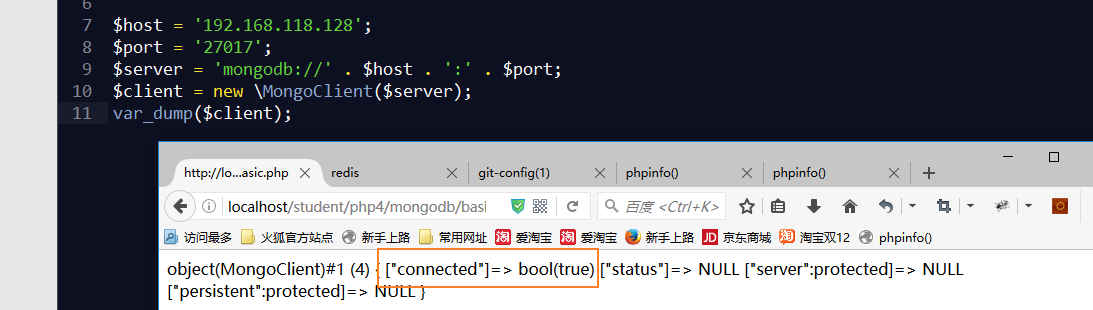
PHP操作时, 几乎也不需要操作 数据库和集合! 主要的操作都集中在 集合中的文档上!

## 实例化mongo客户端对象

mongo扩展, 提供了4个核心类:

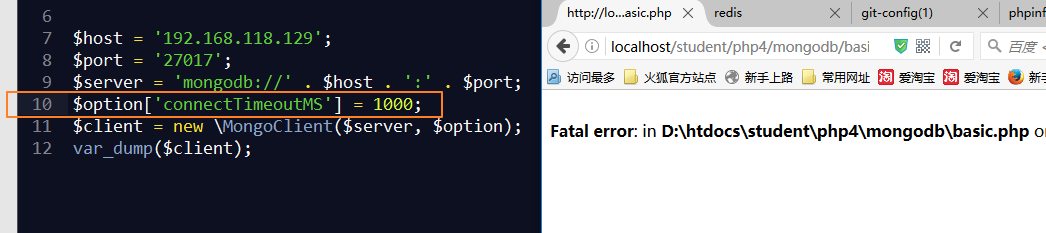


使用最频繁的类对象!



通常需要设置链接超时选项:

connectTimeoutMS

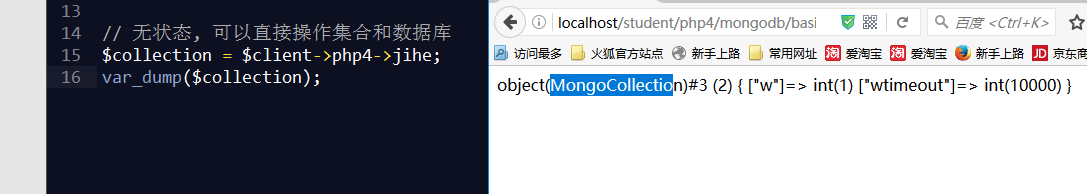


## 访问数据库和集合

基于无状态性, 提供属性既可以访问.

client的属性就是db

db的属性的collection



不存在也是ok的:



(内置实现时, 使用属性重载完成的)

## mongodb数据类型的支持

文档属性(字段), 也是强类型的. 支持各种数据类型.

支持的就是JS支持的数据类型.

undefined

字符串

数值型

布尔型

object

null

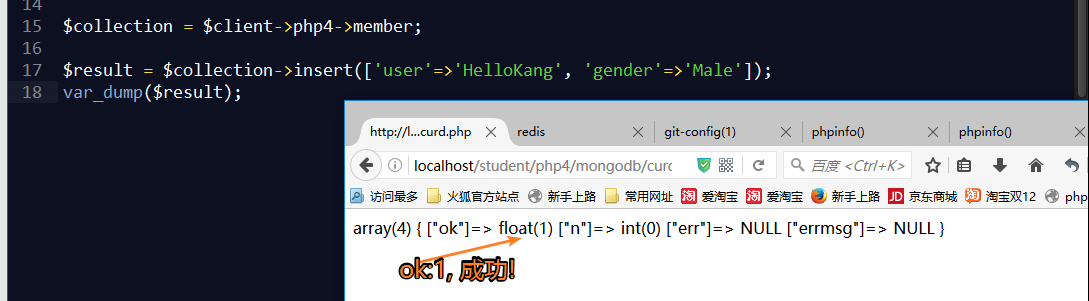
array

## 集合中文档的CRUD

所有的文档的操作都是集中在集合上.

### insert(), 增加

collection.insert(关联数组)



数据类型的处理

需要将所需要的JS的数据类型,使用PHP语法表示出来:

undefined

字符串 PHP字符串

数值型 PHP:整型+浮点型

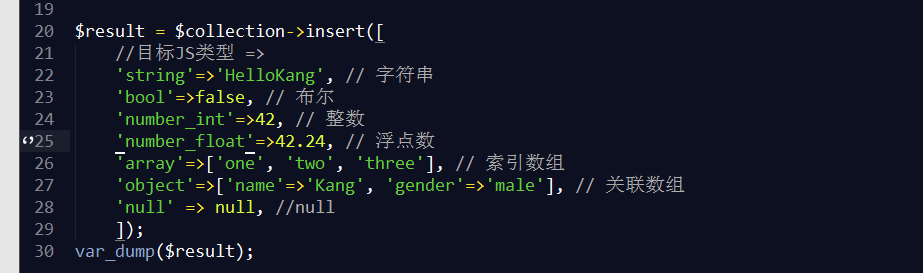
布尔型 PHP布尔型

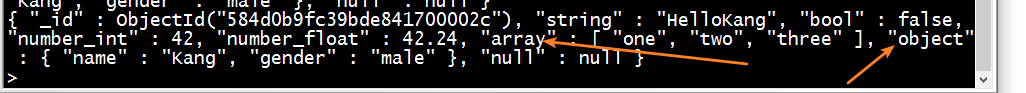
object 关联数组

null null

array 索引数组

PHP语法:





### find(), 查看

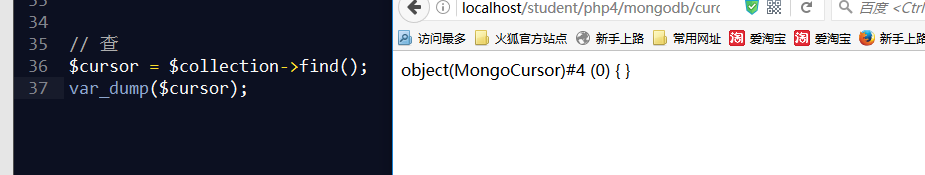
查找数据

collection.find(条件, 字段)

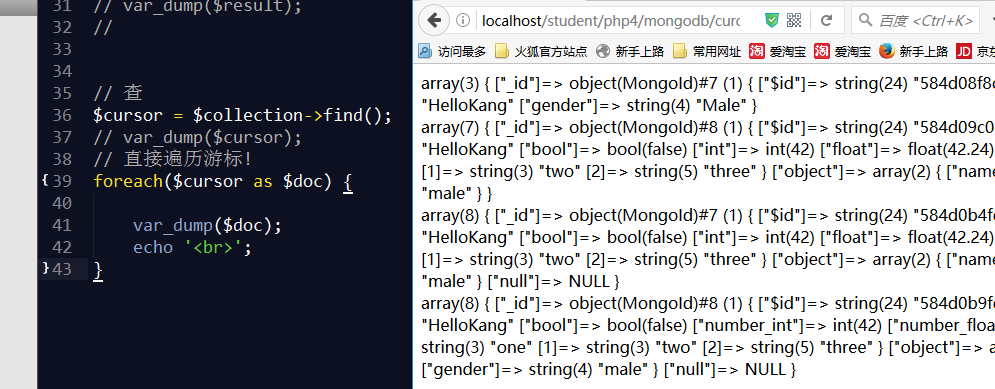
条件: 过滤条件

字段: 仅仅获取需要的属性(字段)

find()返回的是结果集, 游标. cursor



遍历结果集游标, 才可以获取每个查询到的文档:



为什么cursor对象可以在语法上遍历?

cursor类对象, 实现了: 迭代器接口!

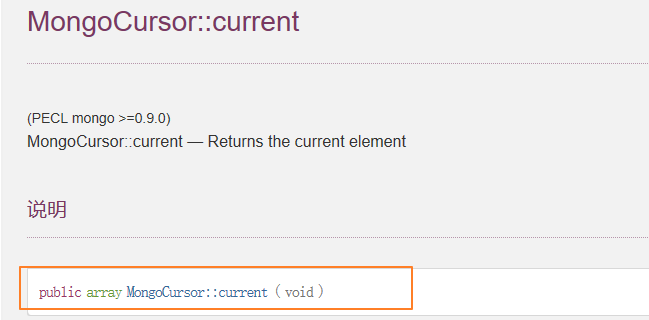


迭代器接口, 要求类必须实现foreach所需要的5个方法.



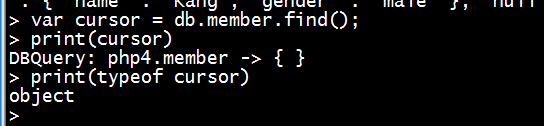
我们的MongoCursor类实现了以上的方法:

在使用foreach遍历$cursor时, $cursor的current方法会执行, 执行的过程中, 返回了一个文档数据.



在mongo客户端, 执行find()时直接返回数据?

客户端执行find, 也返回cursor对象.



不过是, 当对返回的cursor没有做任何处理时, mongo客户端,默认遍历游标对象.

如果需要数据, 则使用cursor的toArray()

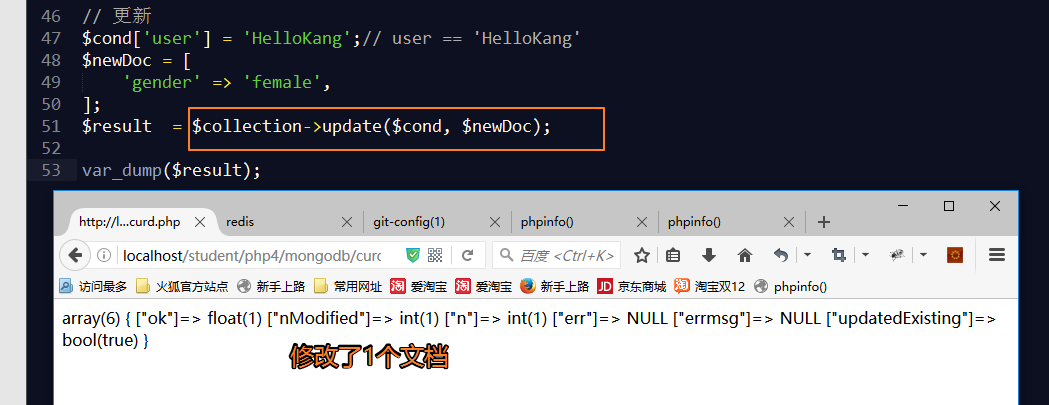


### update()修改

两个基本参数:

1, 条件

2, 更新的json文档

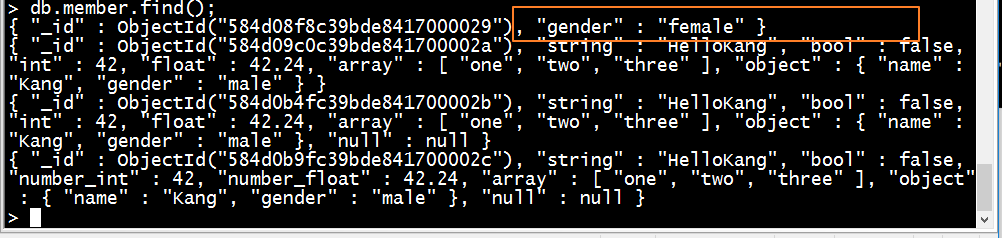


注意修改结果

修改前:



修改后:



修改, 默认是, 新文档替换旧文档. 而不是 字段(属性的)更新!

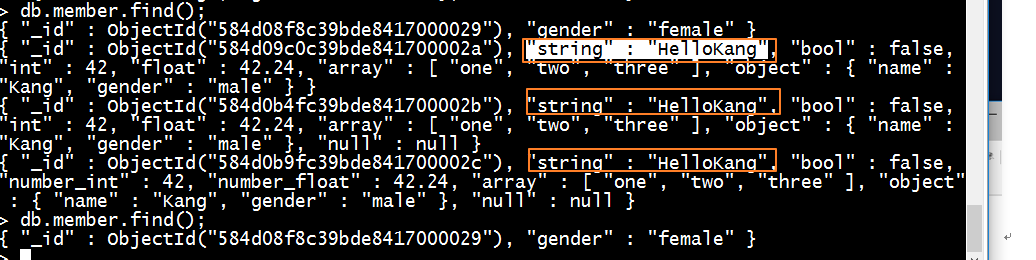
修改支持更复杂的操作!

### remove, 删除

根据条件进行移除



结果



## 条件

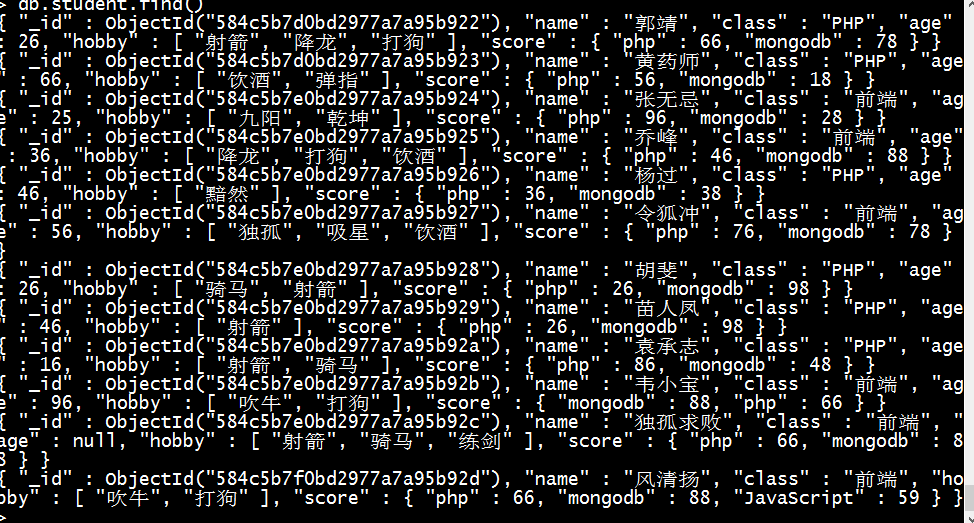
语法如下:

{

‘field1’ : {‘运算符’: 运算数}

}

建立测试数据



### 关系运算

支持的运算符

$eq 等于

$lt 小于

$lte 小于等于

$gt 大于

$gte 大于等于

$ne 不等于

$in in

{'$in': [93, 96]}

$nin

{'$nin': [93, 96]}

$exists 是否存在

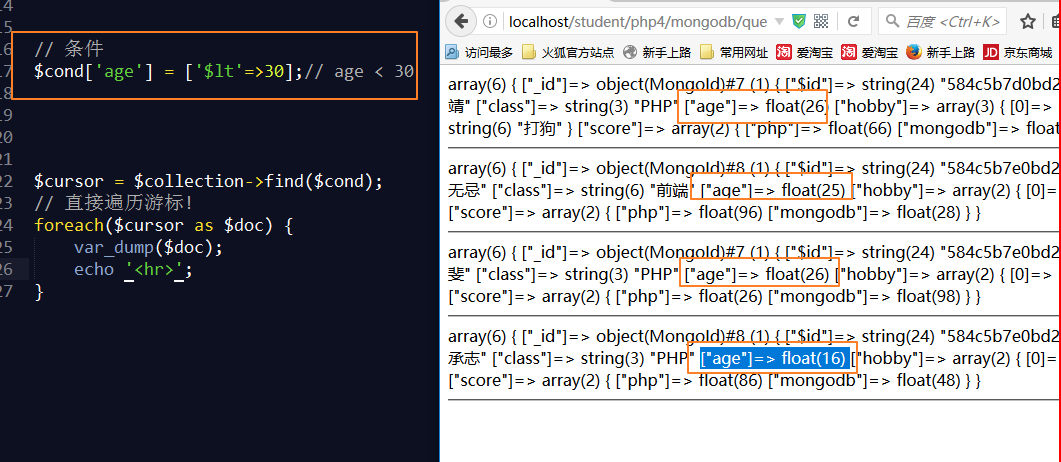
{$exists:false}

$mod 求余为几

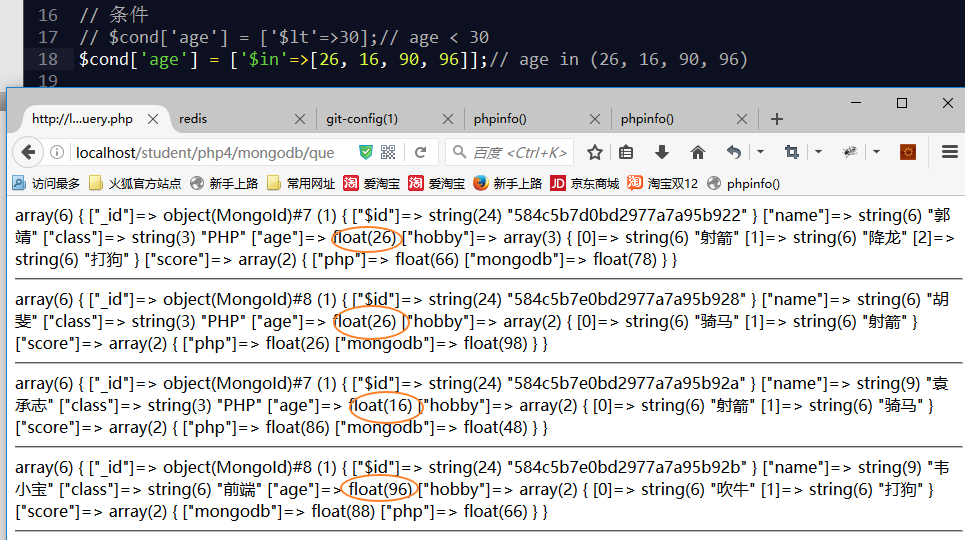
{'$mod': [3, 1]}

例如

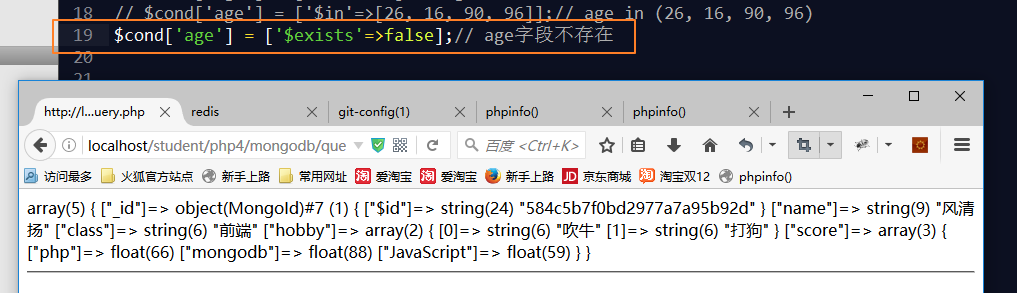
小于



在范围

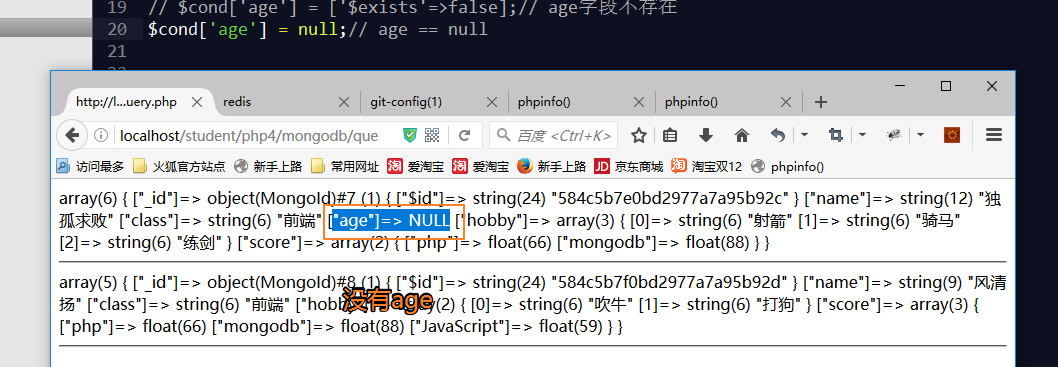


是否存在$exists, 没有age字段



null值的处理,

如果使用 = null, 同时匹配, 值为null以及不存在的字段.



仅仅检索值为null(要求存在且为null)的文档:



可见, 如果一个字段需要同时满足多个条件. 需要使用逗号隔开多个条件表达式!



语法注意, PHP中$表示变量, 为了防止$lt不被解析成变量$lt, 建议, 使用单引号定义.

或者增加\$转义.

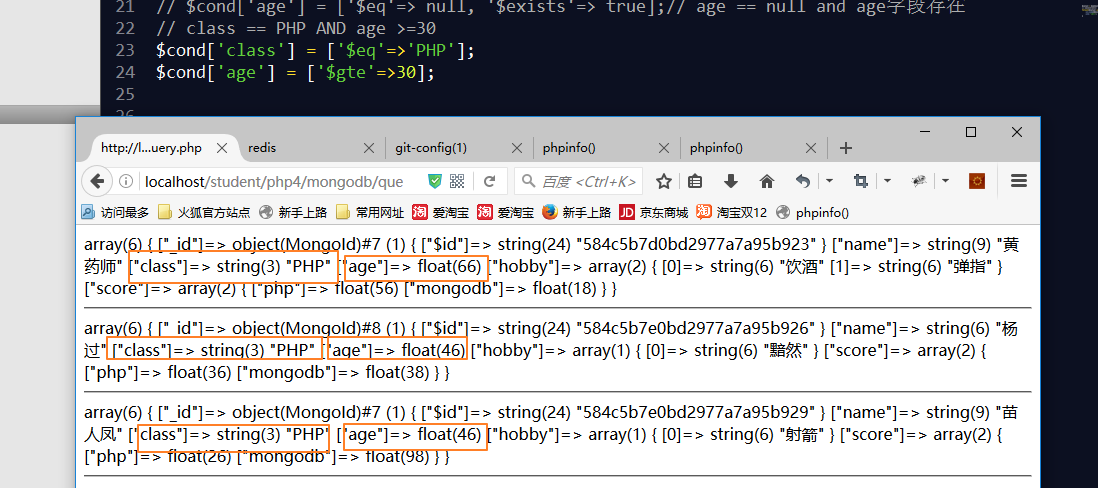
### 逻辑运算

$and

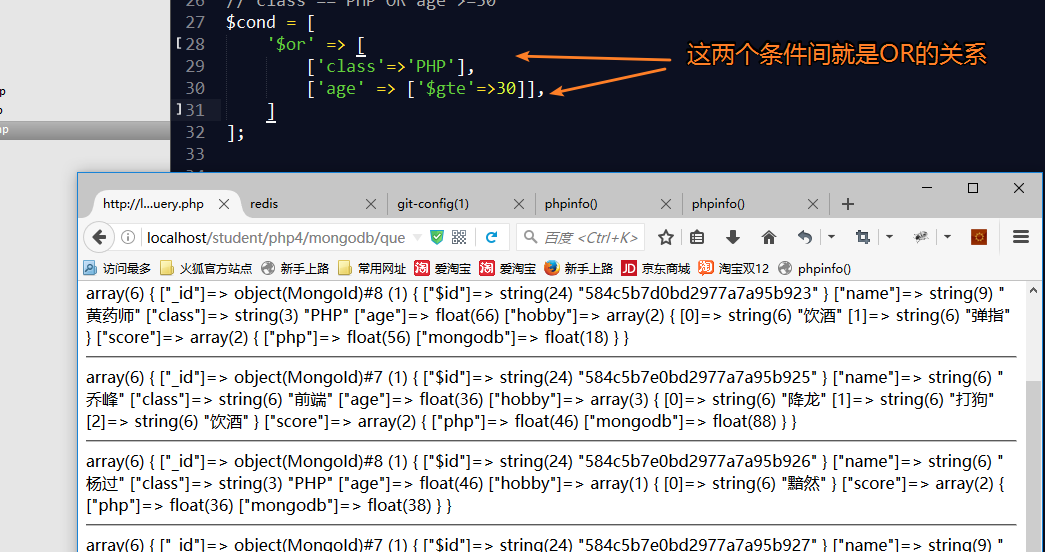
$or

将两个条件, 使用不同的逻辑运算

默认就是AND运算, 因此, 在条件中存在多个元素, 多个元素间, 就是AND



如果是OR关系呢:



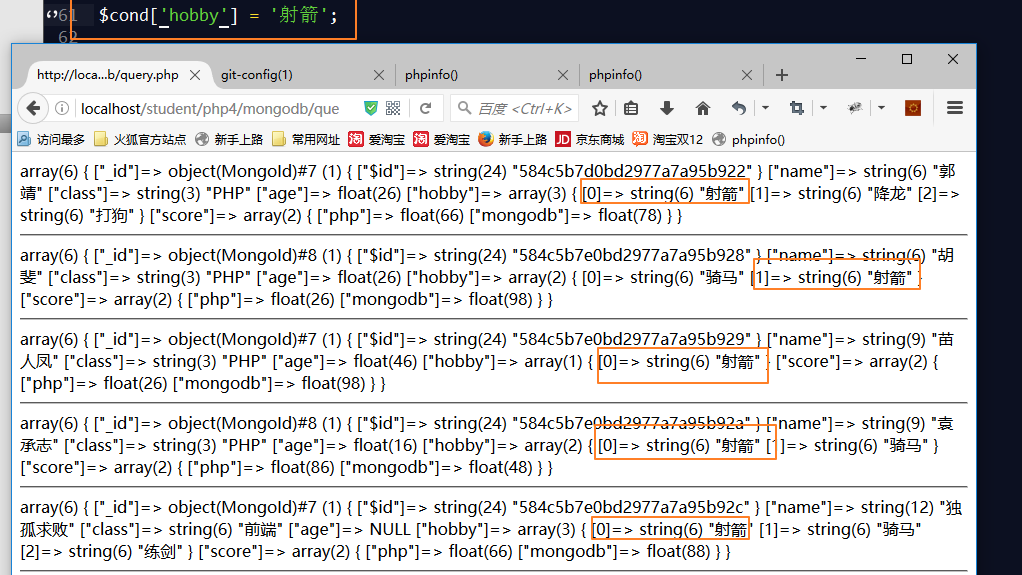
如果是复杂关系呢? 多维数组表示:



### 数组类型运算

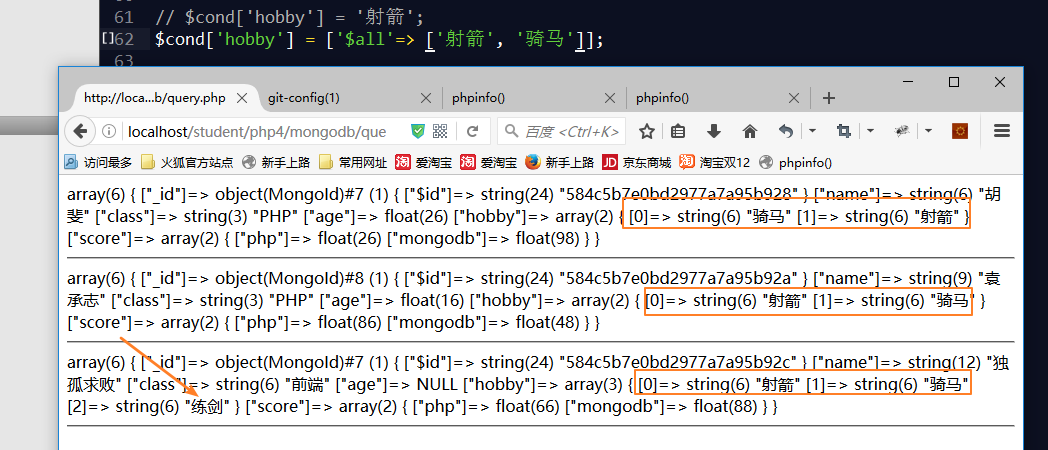
#### 数组中包含某个元素

直接判断是否想等于某个元素即可



#### 同时包含多个元素

于是否存在其他元素没有关系



#### 严格于数组相等

要求元素, 及其顺序



骑马, 射箭 是不能匹配的

### 对象型运算, 内联文档, 内嵌文档

也叫: 内联, 内嵌文档

文档就是json数据, json中包含的json, 就是内嵌的json.

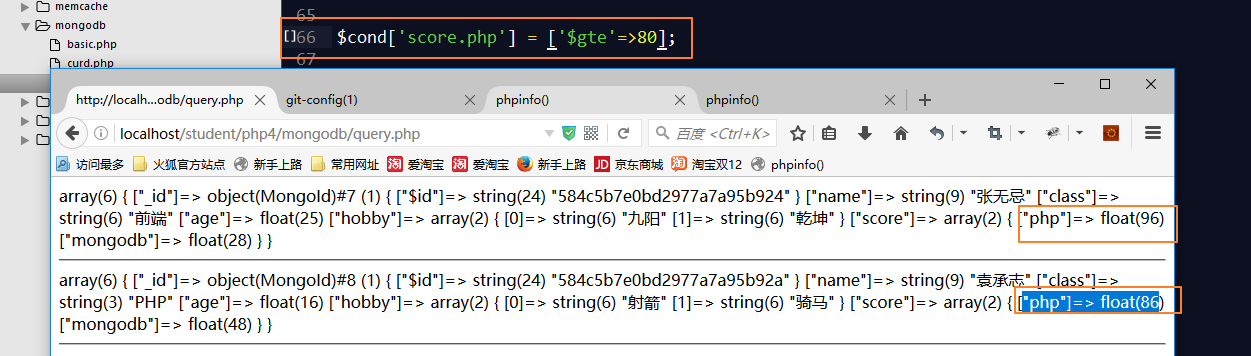


表示于当前文档相关的其他文档, 使用内联文档的方式.

{title: ‘xxxx’, ‘content’ : ‘yyyyyy’, author: {‘name’: ‘xxxx’, ‘age’: xxx}}

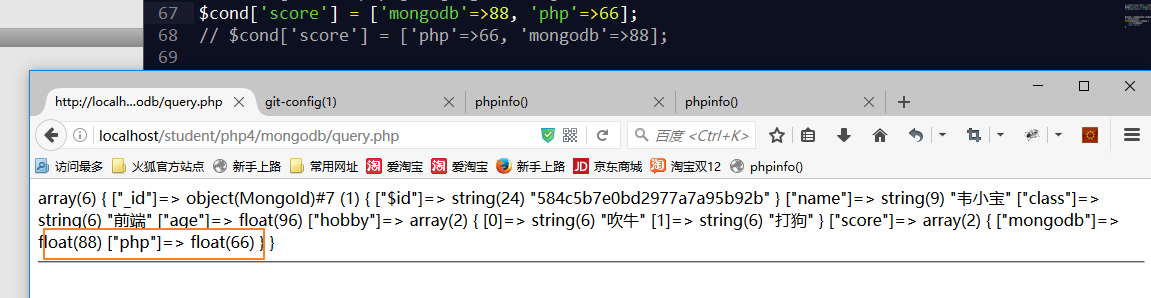
#### 某个字段是否满足条件

直接访问 对象的属性 使用 .即可, 像普通字段, 参与运算即可!



#### 严格相等于某个对象

同时要求顺序和元素

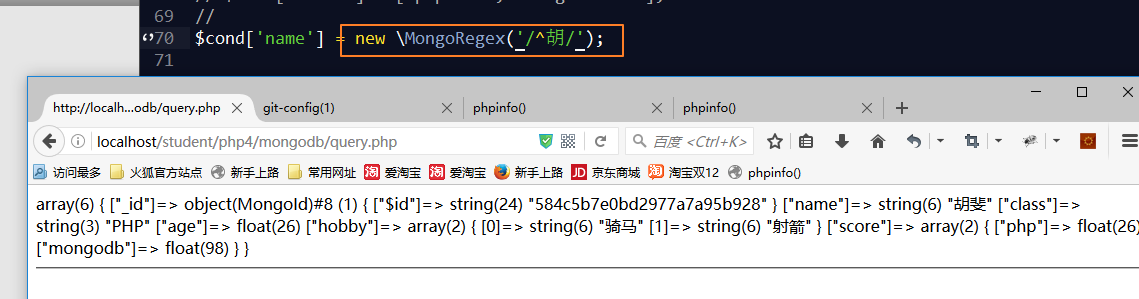


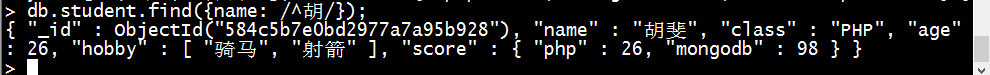
### 正则条件

找到以胡开头的name:

使用 MongoRegex类的对象, 表示正则表达式.

仅仅写表达式, 被当前做字符串去做相等的匹配, 而不是正则match!

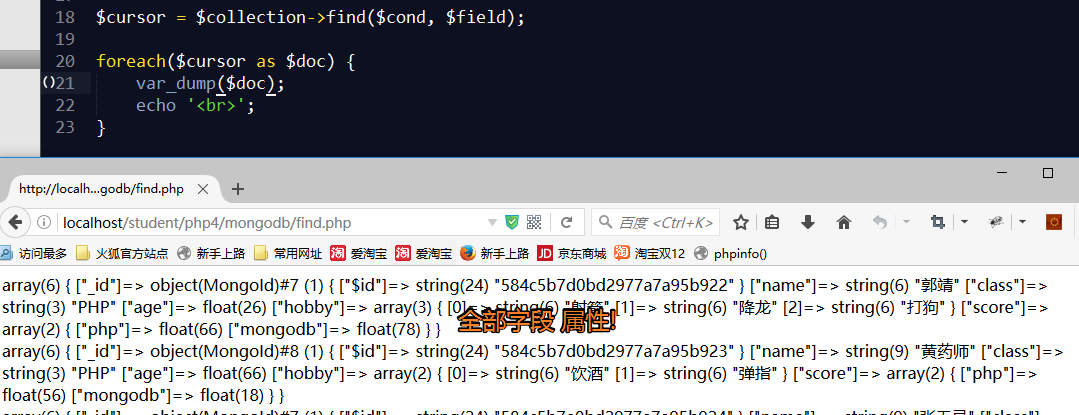




## 查询, find()

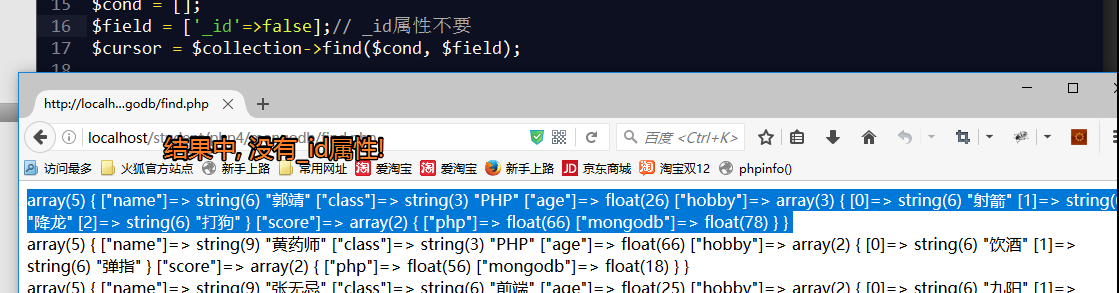
### 参数1:　条件

### 参数2: 　需要的字段

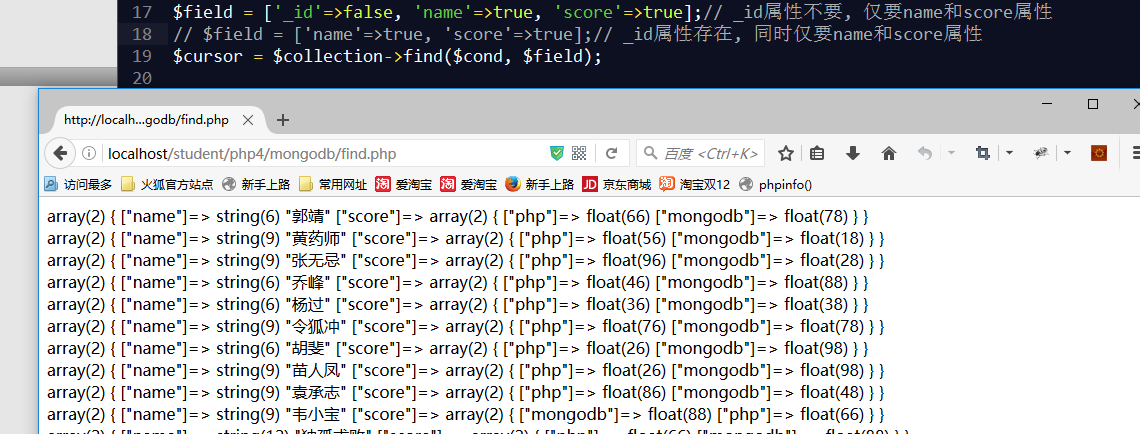


通过指定字段为true, false,表示保留或删除获取的字段:

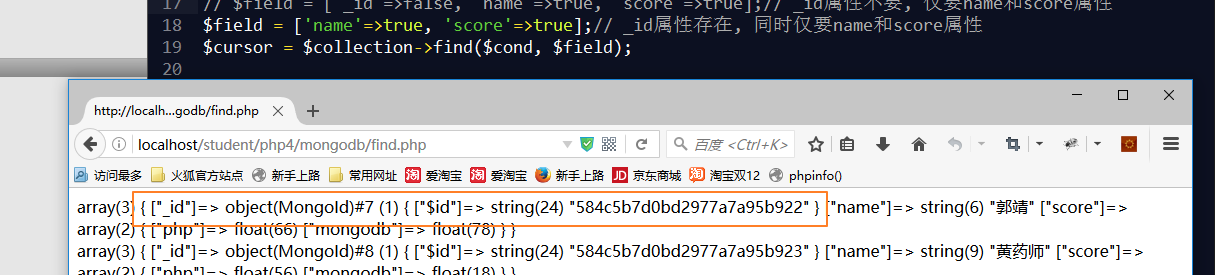
例如:



仅仅需要某些属性



\_id属性, 是mongodb附加的主键属性, 如果没有明确不要, 则默认就会加上该属性, 上例中, 没有\_id=>false, 则也会出现\_id



### \_id

mongodb为每个文档分配的唯一的ID, 用于内部标志.

由一串特殊的字符串组成:

584c5b7d0bd2977a7a95b922

584c5b7d0bd2977a7a95b923

…..

584c5b7f0bd2977a7a95b92d

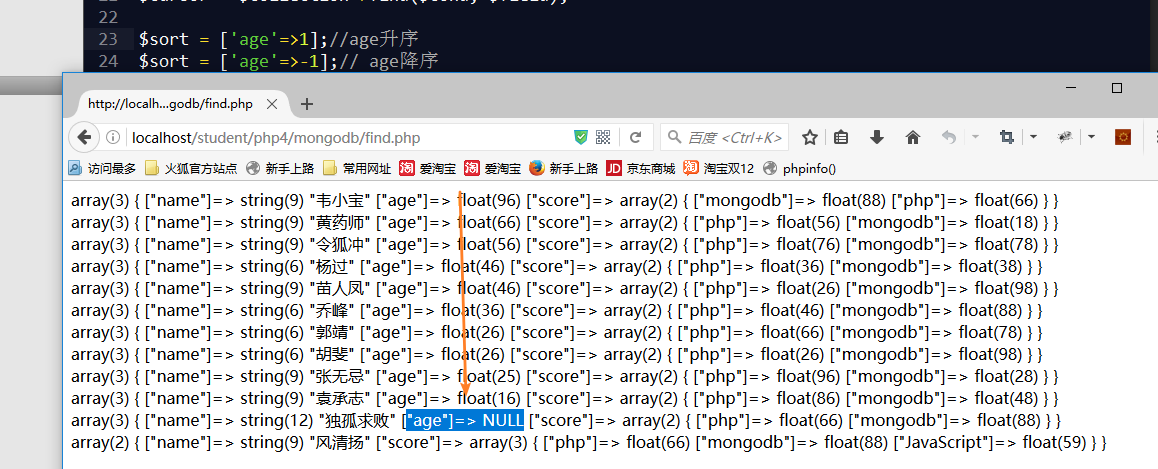
格式：

* 使用 12 个字节来存储，每个字节 由 两位 16 进制数字构成，所以整体是一个24个长度的字符串
* 前 4 个字节 是从标准 纪元开始的时间戳，以秒为单位。
* 接下来 3 个字节 表示所在主机的唯一标识符，通常是主机名的散列值。
* 接下来的 2 个字节 表示 进程 id号
* 最后 3 个字节 是 计数器的计数值，对于任意一秒钟，可以产生2 ^ 24个数

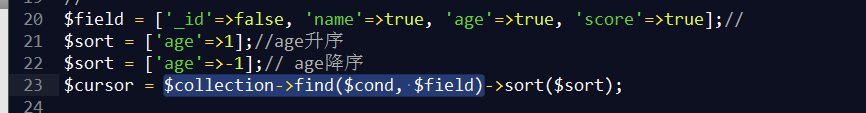
### 排序, $cursor->sort()

字段对应1或者-1, 表示升序或者降序排序.

如果文档没有该属性或者该属性为null, 则不参与排序. 认为是最小的



由于find返回的是cursor, 则可以直接调用cursor的sort方法()

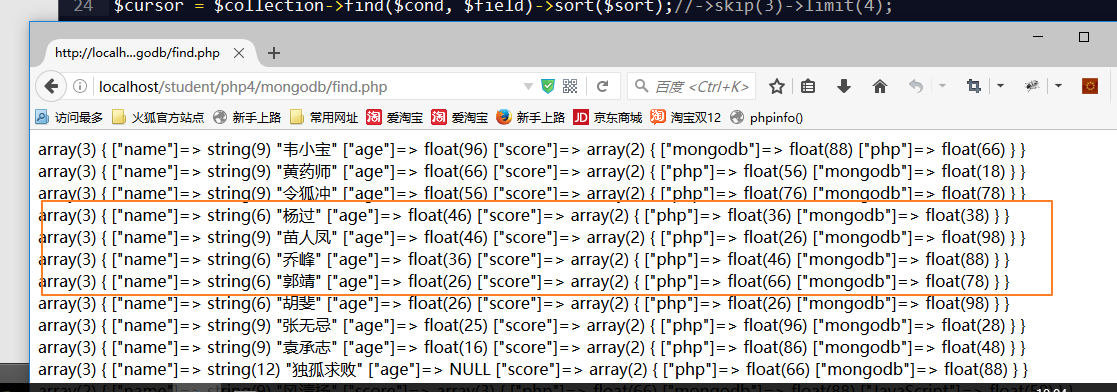


### 文档偏移, $cursor->skip($n)

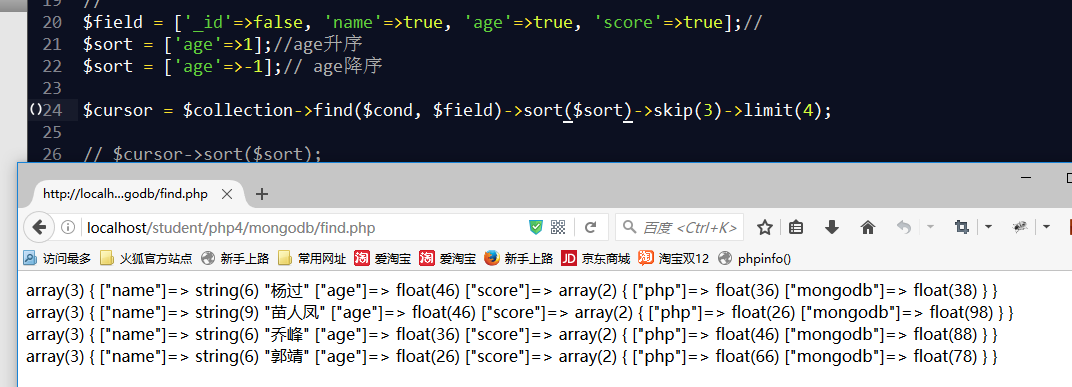
### 限定文档数据量, $cursor->limit($n)

跳过3个, 获取4个

原始数据:



结果



## 分组统计

需要自己定义, 在分组时统计的内容. 没有提供相应的统计函数!

语法为, 使用 集合的 group方法, 完成分组统计:

统计, 每个学科(PHP, 前端)的考试成绩, 平均分, 最高分, 最低分, 学科人数.

代码如下:

|  |
| --- |
| db.student.group({  key: {class: true},// 分组的字段  initial: {total: 0, php: {max:0, min:100, avg:0, sum: 0}, mongodb: {max:0, min:100, avg:0, sum: 0}}, // 组内初始化数组, 根据需要统计的数据, 进行初始化. 该初始值, 针对于每个组都会有. 每组初始化一次. 分组字段, 决定有几组  condition: {'score.php':{'$exists':true}, 'score.mongodb': {'$exists': true}}, // 哪些文档需要参与统计. 仅仅统计有 php和mongodb成绩的文档  '$reduce': function(currDoc, currGroup) {// 自定义的统计方法, 针对于每个满足条件文档, 都会被调用一次  // 函数被调用时, 接收2个参数,  // 参数1, 引用当前遍历的文档, 就可以获取当前文档数据  // 参数2, 引用的当前文档所属的组对象. 可以获取组的初始化的数据  ++ currGroup.total;// 组内人数+1  currGroup.php.sum += currDoc.score.php;// 累加PHP分值到组内PHP总分  currGroup.mongodb.sum += currDoc.score.mongodb;// 累加mongodb总分  // 统计最大值  if (currGroup.php.max < currDoc.score.php) {  // 发现了更大的php分数  currGroup.php.max = currDoc.score.php;  }  if (currGroup.mongodb.max < currDoc.score.mongodb) {  // 发现了更大的mongodb分数  currGroup.mongodb.max = currDoc.score.mongodb;  }  // 统计最小值  if (currGroup.php.min > currDoc.score.php) {  // 发现了更小的php分值  currGroup.php.min = currDoc.score.php;  }  if (currGroup.mongodb.min > currDoc.score.mongodb) {  // 发现了更小的mongodb分值  currGroup.mongodb.min = currDoc.score.mongodb;  }  },  finalize: function(currGroup) { // 在遍历完毕后被调用, 有几个组, 调用几次, 用于组内遍历结束的数据统计  // 接收一个参数  // 参数1, 当前的组内数据的引用  currGroup.php.avg = currGroup.php.sum / currGroup.total;  currGroup.mongodb.avg = currGroup.mongodb.sum / currGroup.total;  }  }); |

group共需要5个选项:

key, 分组字段

initial, 组内的初始化数据, 有几组, 初始化几次

condition: 参与统计的文档条件

$redure, 回调函数, 每个文档调用一次, 会得到当前文档和当前组的参数进行处理(统计的主体参数)

finalize, 回调函数, 在文档遍历完毕后, 每个组调用一次, 用来做最后的统计信息.

### PHP实现

也是集合的group方法:

做对应的转换

将回调, 直接使用字符串JS代码表示即可:

返回统计好的数组

|  |
| --- |
| <?php  namespace php4;  $host = '192.168.118.128';  $port = '27017';  $server = 'mongodb://' . $host . ':' . $port;  $option['connectTimeoutMS'] = 1000;  $client = new \MongoClient($server, $option);  $collection = $client->php4->student;  $key = ['class'=>true];  $initial = ['total'=> 0, 'php'=> ['sum'=>0, 'max'=>0, 'min'=>100, 'avg'=>0, ], 'mongodb'=> ['sum'=>0, 'max'=>0, 'min'=>100, 'avg'=>0, ], ];  $reduce = <<<JS  function(currDoc, currGroup) {// 自定义的统计方法, 针对于每个满足条件文档, 都会被调用一次  // 函数被调用时, 接收2个参数,  // 参数1, 引用当前遍历的文档, 就可以获取当前文档数据  // 参数2, 引用的当前文档所属的组对象. 可以获取组的初始化的数据  ++ currGroup.total;// 组内人数+1  currGroup.php.sum += currDoc.score.php;// 累加PHP分值到组内PHP总分  currGroup.mongodb.sum += currDoc.score.mongodb;// 累加mongodb总分  // 统计最大值  if (currGroup.php.max < currDoc.score.php) {  // 发现了更大的php分数  currGroup.php.max = currDoc.score.php;  }  if (currGroup.mongodb.max < currDoc.score.mongodb) {  // 发现了更大的mongodb分数  currGroup.mongodb.max = currDoc.score.mongodb;  }  // 统计最小值  if (currGroup.php.min > currDoc.score.php) {  // 发现了更小的php分值  currGroup.php.min = currDoc.score.php;  }  if (currGroup.mongodb.min > currDoc.score.mongodb) {  // 发现了更小的mongodb分值  currGroup.mongodb.min = currDoc.score.mongodb;  }  }  JS;  $option['condition'] = ['score.php'=>['$exists'=>true], 'score.mongodb'=>['$exists'=>true]];  $option['finalize'] = <<<JS  function(currGroup) { // 在遍历完毕后被调用, 有几个组, 调用几次, 用于组内遍历结束的数据统计  // 接收一个参数  // 参数1, 当前的组内数据的引用  currGroup.php.avg = currGroup.php.sum / currGroup.total;  currGroup.mongodb.avg = currGroup.mongodb.sum / currGroup.total;  }  JS;  $result = $collection->group($key, $initial, $reduce, $option);  var\_dump($result); |

返回结果, retval元素,就是统计的数据, 同时携带了几个特殊信息:

|  |
| --- |
| array(4) {  ["retval"]=>  array(2) {  [0]=>  array(4) {  ["class"]=>  string(3) "PHP"  ["total"]=>  float(6)  ["php"]=>  array(4) {  ["sum"]=>  float(296)  ["max"]=>  float(86)  ["min"]=>  float(26)  ["avg"]=>  float(49.333333333333)  }  ["mongodb"]=>  array(4) {  ["sum"]=>  float(378)  ["max"]=>  float(98)  ["min"]=>  float(18)  ["avg"]=>  float(63)  }  }  [1]=>  array(4) {  ["class"]=>  string(6) "前端"  ["total"]=>  float(6)  ["php"]=>  array(4) {  ["sum"]=>  float(416)  ["max"]=>  float(96)  ["min"]=>  float(46)  ["avg"]=>  float(69.333333333333)  }  ["mongodb"]=>  array(4) {  ["sum"]=>  float(458)  ["max"]=>  float(88)  ["min"]=>  float(28)  ["avg"]=>  float(76.333333333333)  }  }  }  ["count"]=>  float(12)  ["keys"]=>  int(2)  ["ok"]=>  float(1)  } |

## 更新

### 修改器更新

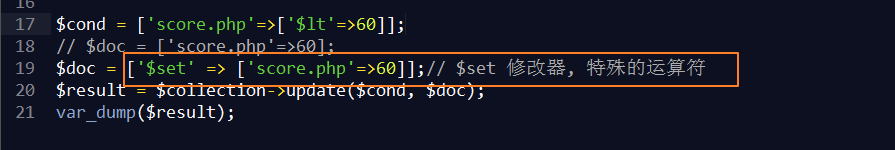
简单更新, 仅仅可以通过条件, 使用新文档替换旧文档!

支持复杂更新, 可以更新某个属性:

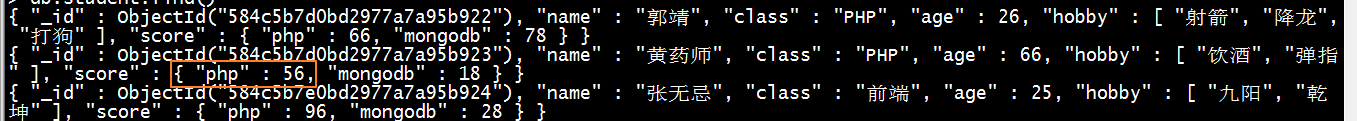
mongodb中使用, 修改器 操作完成, 属性的高级更新!

修改器 就是mongodb支持的一些特殊的运算符, 用于更新文档的:

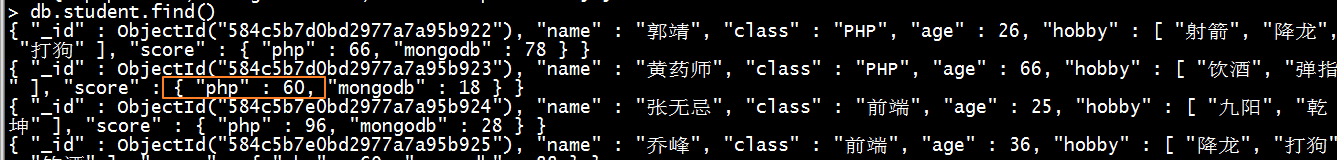
例如, 修改PHP分数小于60的, 改为60



修改前:



修改后:



(mongodb: 介于 NoSQL(非关系) 和 SQL(关系) 之间的数据库产品)

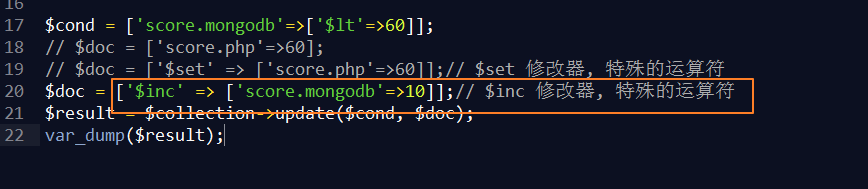
除了set, 还有如下的修改器

$unset, 删除某个属性

{$unset: {class:true}}

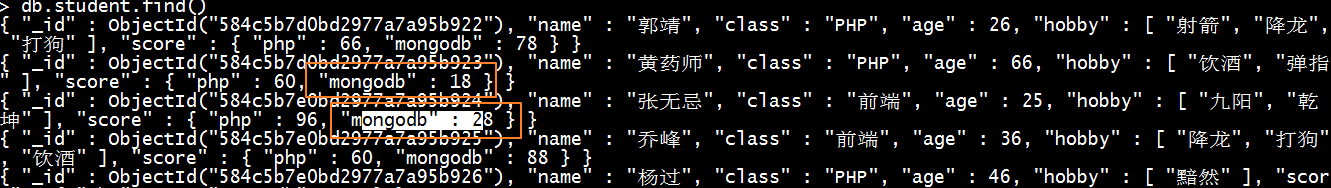
$inc, 递增某个属性

{$inc:{php:10}}

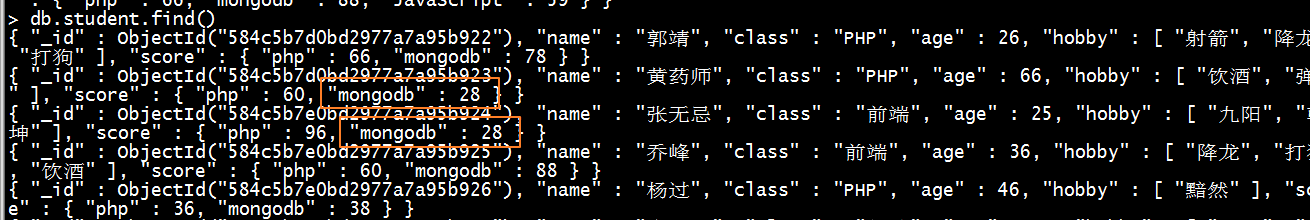




结果,修改前

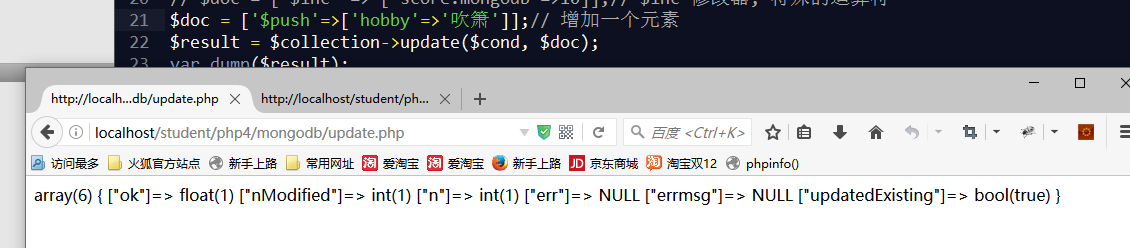


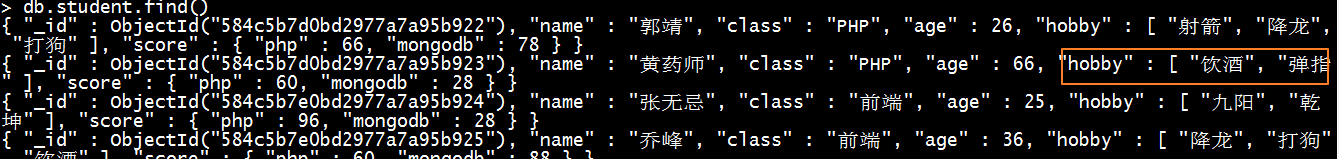
修改后(默认仅仅修改一条)



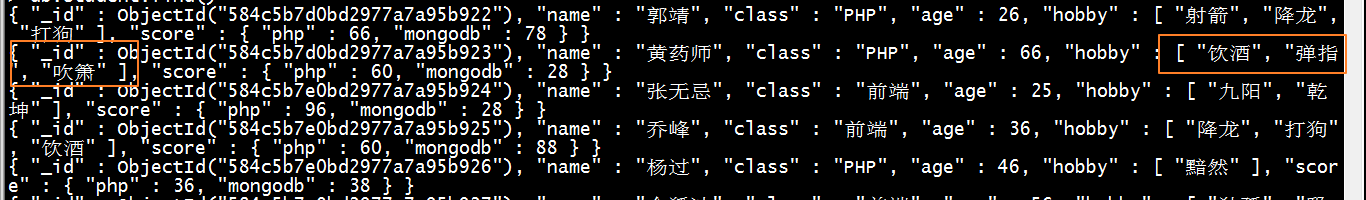
$push, 数组中增加元素

{$push:{hobby:'new-hobby'}}

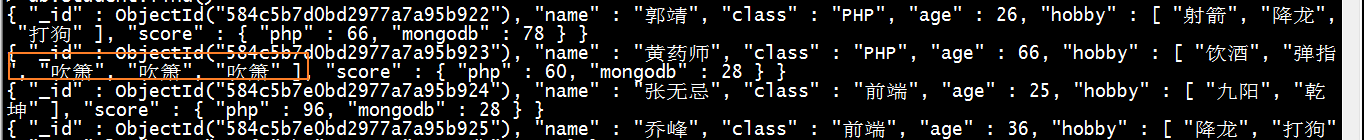




增加了元素



继续增加:可以增加重复元素



$pushAll

{$push:{hobby:['new-hobby-1', 'new-hobby-2']}}

$addToSet, 集合中增加元素, 不能添加重复元素

{$addToSet:{hobby:'new-hobby'}}

相对于push来说, addToSet不能添加重复元素.

$pull, 往外拉, 利用元素值, 比对拉取

|  |
| --- |
| $doc = ['$pull' => ['hobby'=>'吹箫']]; |

$pullAll

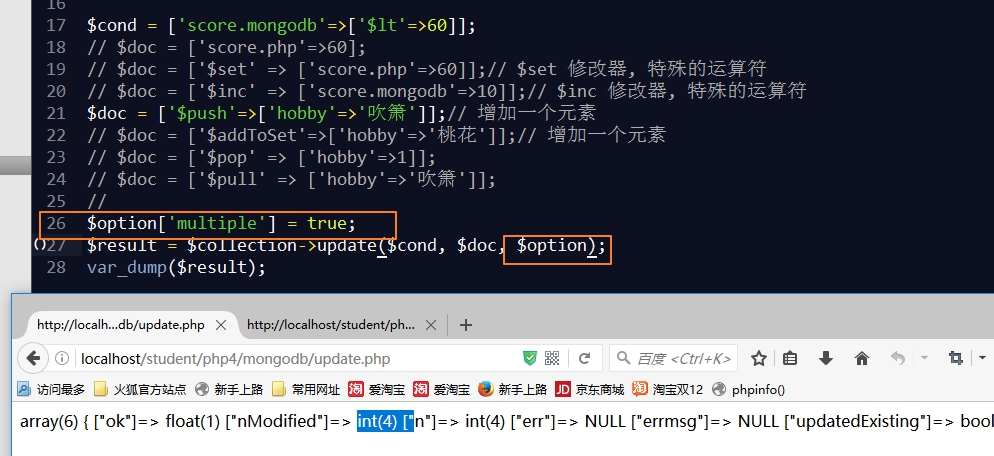
$pop, 弹出,最后的元素, 于$push配合, 可以形成 栈操作.

|  |
| --- |
| $doc = ['$pop' => ['hobby'=>1]]; |

### 多条处理

默认的更新, 仅仅更新匹配的第一条

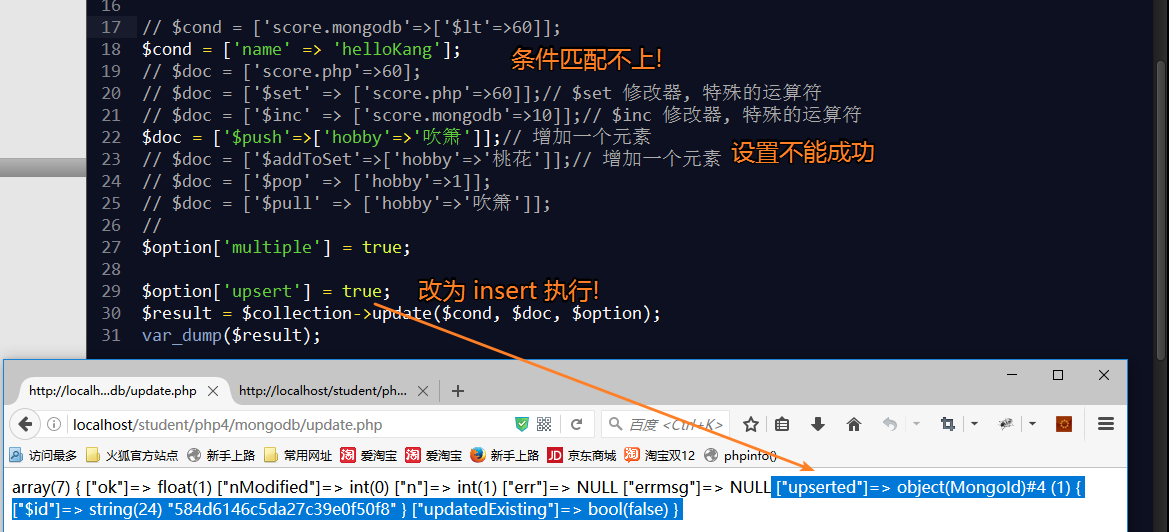
可以通过设置选项, 达到一次性更新多条的目录



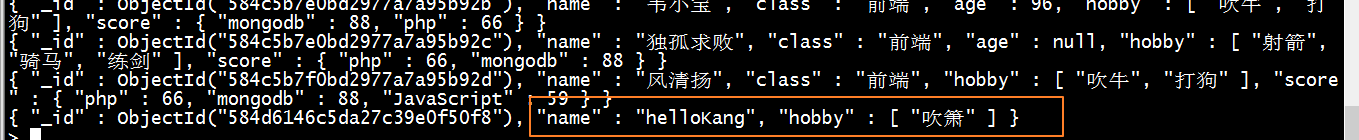
### upsert, update || insert

如果update失败, 则执行insert操作.

update失败的原因, 条件匹配失败!



结果



# 索引

mongodb是数据库产品.

检索数据, 是非常常规的工作.

增加检索的效率. mongodb提供了索引的支持.(越来越倾向于关系型数据库)

通常建立索引的, 基本要求, 就是文档的结构要尽可能一致.

## 管理索引语法

### 创建索引: collection.ensureIndex()

创建索引

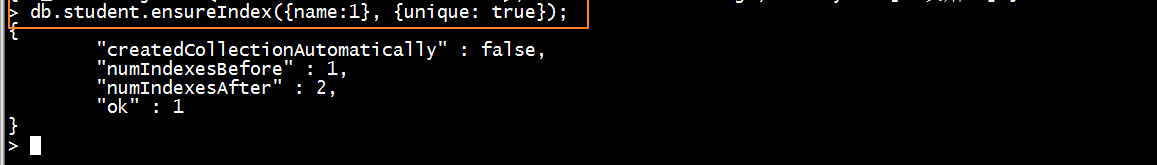
需要指定

在那个属性上建立索引

索引自身的选项(是否唯一性)

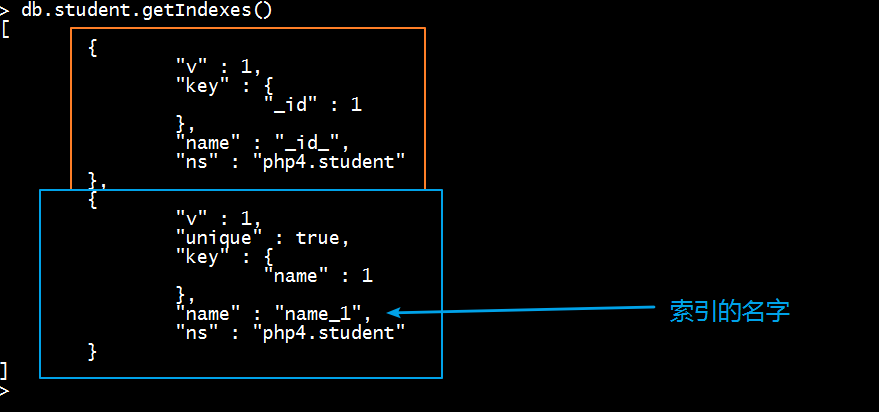
演示

在student集合上, 的name属性中, 建立唯一的索引

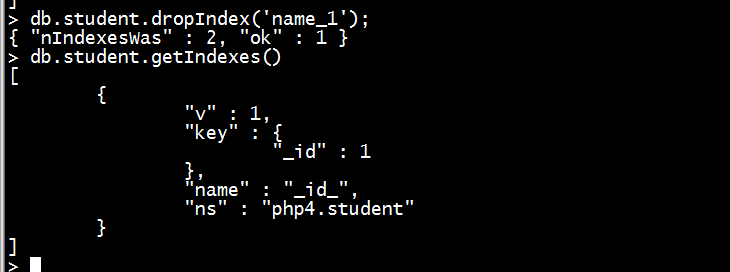


使用时, 不用考虑索引是否存在的. 自动启用!

### 查看集合存在的索引:collection.getIndexes()



### 删除索引: collection.dropIndex(index-name)



## 索引类型

普通索引, 没有对索引内容的要求的. 不增加任何选项即可

唯一索引, 要求索引的内容, 不能重复. 增加{unique: true}选项即可

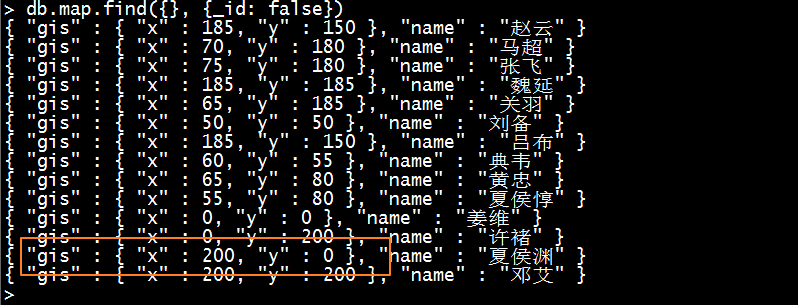
2d平面索引, 2维坐标作引, 用于快速检索一张平面地图的位置.

## 2d索引

### 测试数据

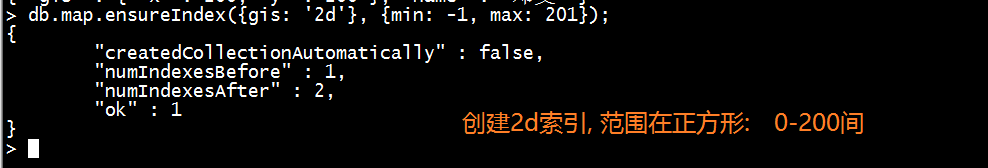


利用2d索引, 可以检索 离某个点, 最近的点有哪些. 某个范围内的所有的点, 例如, 如下的测试数据



### 创建索引

db.map.ensureIndex({gis: '2d'}, {min: -1, max: 201});



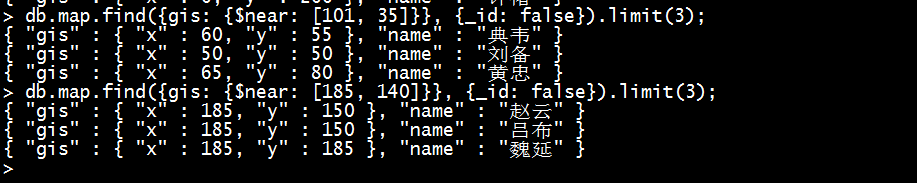
### 利用索引检索

#### 最近点检索

我当前在 101, 35, 找到离我最近的几个点

使用运算符:

$near,提供参考点的坐标即可

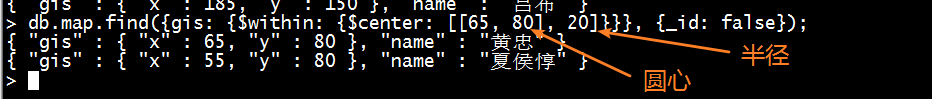


#### 检索圆形范围点

使用运算符$within来表示. 需要的参数, 需要表示当前是圆形, 使用$center表示.

需要表示出来圆心和半径.

db.map.find({gis: {$within: {$center: [[65, 80], 20]}}}, {\_id: false});

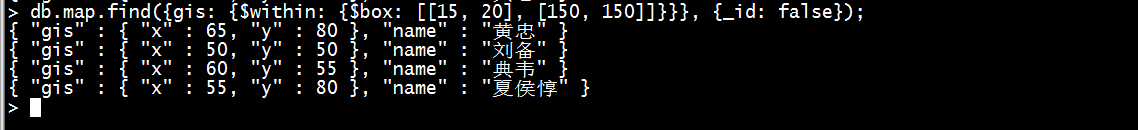


#### 检索矩形范围

需要表示2个点, 确定矩形范围.

使用运算符$box表示矩形

db.map.find({gis: {$within: {$box: [[15, 20], [150, 150]]}}}, {\_id: false});



三国:

1

2

3

4

5