**Python, javascript和bash比较**

传统的关系[数据库](http://www.itokit.com/tag.php?tag=%CA%FD%BE%DD%BF%E2" \t "_blank)一般由[数据库](http://www.itokit.com/tag.php?tag=%CA%FD%BE%DD%BF%E2" \t "_blank)（database）、表（table）、记录（record）三个层次概念组成，MongoDB是由**数据库（database）、集合（collection）、文档对象（document）**三个层次组成。MongoDB对于关系型数据库里的表，但是集合中没有列、行和关系概念，这体现了模式自由的特点。

|  |  |  |
| --- | --- | --- |
| 语言 | python | **Object-c** |
| 版本 | Python --version |  |
| 执行 | Python file.py |  |
| 注释 | # comment  ‘’’ multi comment’’’ | // comment  /\* multi commnet \*/ |
| 变量 | X, y = 2, 3  Global x, x = 3 | Int i = 3;  Extern int 3; |
| 数据类型 | String,bool,int,long,list,  tuple,dict,set |  |
| bool | True, False |  |
| 计算长度 | Len(l) |  |
| 函数定义 | Def add(x,y): |  |
| 定义类 | Class test: |  |
| 数组定义 | L = [3,7] |  |
| 条件语句 | If statement;  Elif statement;else |  |
| 循环语句 | For x in xrange(7):  Print x |  |
| 遍历 | For x in l:  Print x |  |
| 字符串拼接 | S = ‘hello’ + ‘world’ |  |
| 字符串分割 | S[3:7] |  |
| 字符串替换 | S.replace(‘hello’,’hi’) |  |
| 导入 | From time import sleep |  |
| 字符串执行 | Eval(‘port=223’) |  |
| 字符串转出整型 | Int(‘32’[,16])  Float(‘32’) |  |
| 字符串大小写 | ‘hello’.lower()  ‘hi’.uppper() |  |
|  |  |  |
| 列表 | L = [] |  |
| 添加元素 | L.append(‘hello’) # 在末尾添加  L.insert(0, ‘word’) # 0位置添加 |  |
| 删除元素 | L.pop() # 删除最后一个  L.remove(value) # 删value |  |
| 修改元素 | L[ |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**mongodb语法**

MongoDB的好处挺多的，比如多列索引，查询时可以用一些统计函数，支持多条件查询，但是目前多表查询是不支持的，可以想办法通过数据冗余来解决多表查询的问题。

MongoDB对数据的操作很丰富，下面做一些举例说明，内容大部分来自官方文档，另外有部分为自己理解。

查询colls所有数据

db.colls.find() //select \* from colls

通过指定条件查询

db.colls.find({‘last\_name’: ‘Smith’});//select \* from colls where last\_name=’Smith’

指定多条件查询

db.colls.find( { x : 3, y : “foo” } );//select \* from colls where x=3 and y=’foo’

指定条件范围查询

db.colls.find({j: {$ne: 3}, k: {$gt: 10} });//select \* from colls where j!=3 and k>10

查询不包括某内容

db.colls.find({}, {a:0});//查询除a为0外的所有数据

支持<, <=, >, >=查询，需用符号替代分别为$lt，$lte，$gt，$gte

db.colls.find({ “field” : { $gt: value } } );

db.colls.find({ “field” : { $lt: value } } );

db.colls.find({ “field” : { $gte: value } } );

db.colls.find({ “field” : { $lte: value } } );

也可对某一字段做范围查询

db.colls.find({ “field” : { $gt: value1, $lt: value2 } } );

不等于查询用字符$ne

db.colls.find( { x : { $ne : 3 } } );

in查询用字符$in

db.colls.find( { “field” : { $in : array } } );

db.colls.find({j:{$in: [2,4,6]}});

not in查询用字符$nin

db.colls.find({j:{$nin: [2,4,6]}});

取模查询用字符$mod

db.colls.find( { a : { $mod : [ 10 , 1 ] } } )// where a % 10 == 1

$all查询

db.colls.find( { a: { $all: [ 2, 3 ] } } );//指定a满足数组中任意值时

$size查询

db.colls.find( { a : { $size: 1 } } );//对对象的数量查询，此查询查询a的子对象数目为1的记录

$exists查询

db.colls.find( { a : { $exists : true } } ); // 存在a对象的数据

db.colls.find( { a : { $exists : false } } ); // 不存在a对象的数据

$type查询$type值为bsonhttp://bsonspec.org/数 据的类型值

db.colls.find( { a : { $type : 2 } } ); // 匹配a为string类型数据

db.colls.find( { a : { $type : 16 } } ); // 匹配a为int类型数据

使用正则表达式匹配

db.colls.find( { name : /acme.\*corp/i } );//类似于SQL中like

内嵌对象查询

db.colls.find( { “author.name” : “joe” } );

1.3.3版本及更高版本包含$not查询

db.colls.find( { name : { $not : /acme.\*corp/i } } );

db.colls.find( { a : { $not : { $mod : [ 10 , 1 ] } } } );

sort()排序

db.colls.find().sort( { ts : -1 } );//1为升序2为降序

limit()对限制查询数据返回个数

db.colls.find().limit(10)

skip()跳过某些数据

db.colls.find().skip(10)

snapshot()快照保证没有重复数据返回或对象丢失

count()统计查询对象个数

db.students.find({‘address.state’ : ‘CA’}).count();//效率较高

db.students.find({‘address.state’ : ‘CA’}).toArray().length;//效率很低

group()对查询结果分组和SQL中group by函数类似

distinct()返回不重复值