spring-data-mongodb中的实体映射是通过MongoMappingConverter这个类实现的。它可以通过注释把java类转换为mongodb的文档。

它有以下几种注释：

@Id - 文档的唯一标识，在mongodb中为ObjectId，它是唯一的，通过时间戳+机器标识+进程ID+自增计数器（确保同一秒内产生的Id不会冲突）构成。

@Document - 把一个java类声明为mongodb的文档，可以通过collection参数指定这个类对应的文档。

@DBRef - 声明类似于关系数据库的关联关系。ps：暂不支持级联的保存功能，当你在本实例中修改了DERef对象里面的值时，单独保存本实例并不能保存DERef引用的对象，它要另外保存，如下面例子的Person和Account。

@Indexed - 声明该字段需要索引，建索引可以大大的提高查询效率。

@CompoundIndex - 复合索引的声明，建复合索引可以有效地提高多字段的查询效率。

@GeoSpatialIndexed - 声明该字段为地理信息的索引。

@Transient - 映射忽略的字段，该字段不会保存到mongodb。

@PersistenceConstructor - 声明构造函数，作用是把从数据库取出的数据实例化为对象。该构造函数传入的值为从DBObject中取出的数据。

以下引用一个官方文档的例子：

Person类

@Document(collection="person")

@CompoundIndexes({

@CompoundIndex(name = "age\_idx", def = "{'lastName': 1, 'age': -1}")

})

public class Person<T extends Address> {

@Id

private String id;

@Indexed(unique = true)

private Integer ssn;

private String firstName;

@Indexed

private String lastName;

private Integer age;

@Transient

private Integer accountTotal;

@DBRef

private List<Account> accounts;

private T address;

public Person(Integer ssn) {

this.ssn = ssn;

}

@PersistenceConstructor

public Person(Integer ssn, String firstName, String lastName, Integer age, T address) {

this.ssn = ssn;

this.firstName = firstName;

this.lastName = lastName;

this.age = age;

this.address = address;

}

Account类

@Document

public class Account {

@Id

private ObjectId id;

private Float total;

}

spring data 4 mongoDB自动创建复合索引

spring data 4 mongodb 在domain上添加annation，自动创建复合索引时需要使用CompoundIndexes。

例如：

@CompoundIndex(name = "shop\_index", def = "{platform : 1, shopId : 1}")

程序也不会有编译错误或者执行错误，但是spring data不会建立任何索引,

下面这样写才会启动时自动建立复合索引。

@CompoundIndexes({

     @CompoundIndex(name = "shop\_index", def = "{platform : 1, shopId : 1}")

})

MongoDB支持复合索引，即将多个键组合到一起创建索引。该方式称为复合索引，或者也叫组合索引，该方式能够满足多键值匹配查询使用索引的情形。其次复合索引在使用的时候，也可以通过前缀法来使用索引。MongoDB中的复合索引与关系型数据库基本上一致。在关系型数据库中复合索引使用的一些原则同样适用于MongoDB。本文主要描述MongoDB复合索引。

一、复合索引相关概述

1、复合索引创建语法

db.collection.createIndex( { <field1>: <type>, <field2>: <type2>, ... } )

同创建单键(列)索引一样，索引创建时需要指定每一个键索引的顺序

多个键直接用逗号分隔

索引创建语法可以参考：http://blog.csdn.net/leshami/article/details/53541978

2、复合索引的一些特性

复合索引可以支持要求匹配多个键的查询

复合索引每一个键的顺序非常重要，这将决定该索引在查询过程中能否被使用到

复合索引支持前导(缀)列索引查询

不能够创建基于哈希索引类型的复合索引

任意复合索引字段不能超过31个

二、复合索引示意图

如下图所示，在集合的userid以及score列上创建一个复合索引，其中userid为升序，score为降序

三、复合索引示例

1、演示环境

> db.version()

3.2.10

> db.example.find({},{"\_id":0})

{ "id" : 1, "ename" : "leshami", "blog" : "http://blog.csdn.net/leshami", "name" : "leshami" }

演示集合数据，可以参考：http://blog.csdn.net/leshami/article/details/52672310

//查看任意的一个文档

> db.persons.find().limit(1).pretty()

{

"\_id" : ObjectId("5812cbaaa129eed14b46458d"),

"name" : "robinson.cheng",

"age" : 25,

"email" : "robinson.cheng@qq.com",

"score" : {

"c" : 89,

"m" : 96,

"e" : 87

},

"country" : "USA",

"books" : [

"JS",

"C++",

"EXTJS",

"MONGODB"

]

}

2、创建复合索引

//如下示例，我们在集合persons上的name及age键上创建复合索引，且2个都为升序

> db.persons.createIndex({name:1,age:1})

{

"createdCollectionAutomatically" : false,

"numIndexesBefore" : 1,

"numIndexesAfter" : 2,

"ok" : 1

}

//在上面的示例中索引首先会按照name的值升序进行排列

//其次是age键，在name之后也按照升序排列

//下面过滤条件仅使用一个name键来查看执行计划

> db.persons.find({name:"robinson.cheng"}).explain()

{

......

"stage" : "FETCH",

"inputStage" : {

"stage" : "IXSCAN", //使用索引扫描

"keyPattern" : {

"name" : 1,

"age" : 1

},

"indexName" : "name\_1\_age\_1",

......

"direction" : "forward",

"indexBounds" : {

"name" : [

"[\"robinson.cheng\", \"robinson.cheng\"]"

],

"age" : [

"[MinKey, MaxKey]"

.........

"ok" : 1

}

//下面过滤条件仅使用name及age键来查看执行计划

> db.persons.find({name:"robinson.cheng",age:25}).explain()

{

.........

"stage" : "FETCH",

"inputStage" : {

"stage" : "IXSCAN", //使用索引扫描

"keyPattern" : {

"name" : 1,

"age" : 1

},

"indexName" : "name\_1\_age\_1",

.........

"direction" : "forward",

"indexBounds" : {

"name" : [

"[\"robinson.cheng\", \"robinson.cheng\"]"

],

"age" : [

"[25.0, 25.0]"

...........

"ok" : 1

}

//下面过滤条件仅使用name及age键来查看执行计划，但是将age键放在name键之前

> db.persons.find({age:25,name:"robinson.cheng"}).explain()

{

"queryPlanner" : {

.....

"winningPlan" : {

"stage" : "FETCH",

"inputStage" : {

"stage" : "IXSCAN", //使用索引扫描

"keyPattern" : {

"name" : 1,

"age" : 1

},

"indexName" : "name\_1\_age\_1",

...........

"direction" : "forward",

"indexBounds" : {

"name" : [

"[\"robinson.cheng\", \"robinson.cheng\"]"

],

"age" : [

"[25.0, 25.0]"

........

"ok" : 1

}

//下面单独基于age键作为过滤条件进行查询

> db.persons.find({age:25}).explain()

{

................

"winningPlan" : {

"stage" : "COLLSCAN", //此处为使用集合扫描方式

"filter" : {

"age" : {

"$eq" : 25

}

},

"direction" : "forward"

},

"rejectedPlans" : [ ]

..............

"ok" : 1

}

3、复合索引与排序

复合索引创建时按升序或降序来指定其排列方式。对于单键索引，其顺序并不是特别重要，因为MongoDB可以在任一方向遍历索引

对于复合索引，按何种方式排序能够决定该索引在查询中能否被使用到。

//以下内容基于前面在{name:1,age:1}键上创建的索引来考察这个复合索引在排序时被使用到的场景

//基于{name:1,age:1}的排序

> db.persons.find().sort({name:1,age:1}).explain()

{

"queryPlanner" : {

....

"winningPlan" : {

"stage" : "FETCH",

"inputStage" : {

"stage" : "IXSCAN", //索引扫描

"keyPattern" : {

"name" : 1,

"age" : 1

},

"indexName" : "name\_1\_age\_1",

..........

"direction" : "forward",

"indexBounds" : {

"name" : [

"[MinKey, MaxKey]"

],

"age" : [

"[MinKey, MaxKey]"

....

"ok" : 1

}

//基于{name:1,age:-1}的排序

> db.persons.find().sort({name:1,age:-1}).explain()

{

"queryPlanner" : {

....

"winningPlan" : {

"stage" : "SORT",

"sortPattern" : {

"name" : 1,

"age" : -1

},

"inputStage" : {

"stage" : "SORT\_KEY\_GENERATOR",

"inputStage" : {

"stage" : "COLLSCAN", //集合扫描

"filter" : {

"$and" : [ ]

},

"direction" : "forward"

.........

"ok" : 1

}

//基于{name:-1,age:1}的排序

> db.persons.find().sort({name:-1,age:1}).explain()

{

"queryPlanner" : {

.........

"winningPlan" : {

"stage" : "SORT",

"sortPattern" : {

"name" : -1,

"age" : 1

},

"inputStage" : {

"stage" : "SORT\_KEY\_GENERATOR",

"inputStage" : {

"stage" : "COLLSCAN", //集合扫描

"filter" : {

"$and" : [ ]

},

"direction" : "forward"

.........

"ok" : 1

}

//基于{age:1,name:1}的排序

> db.persons.find().sort({age:1,name:1}).explain()

{

....

"inputStage" : {

"stage" : "SORT\_KEY\_GENERATOR",

"inputStage" : {

"stage" : "COLLSCAN", //集合扫描

"filter" : {

"$and" : [ ]

},

"direction" : "forward"

..........

"ok" : 1

}

//基于{age:-1,name:1}的排序

> db.persons.find().sort({age:-1,name:1}).explain()

{

..........

"inputStage" : {

"stage" : "SORT\_KEY\_GENERATOR",

"inputStage" : {

"stage" : "COLLSCAN", //集合扫描

"filter" : {

"$and" : [ ]

},

"direction" : "forward"

..........

"ok" : 1

}

//基于{name:-1,age:-1}的排序

> db.persons.find().sort({name:-1,age:-1}).explain()

{

.........

"inputStage" : {

"stage" : "IXSCAN", //索引扫描

"keyPattern" : {

"name" : 1,

"age" : 1

},

"indexName" : "name\_1\_age\_1",

............

"direction" : "backward", //注意，这里的方向为向后扫描

"indexBounds" : {

"name" : [

"[MaxKey, MinKey]"

],

"age" : [

"[MaxKey, MinKey]"

......

"ok" : 1

}

通过上面的不同场景，得出如下：

排序使用到索引的情形

db.persons.find().sort({name:1,age:1})

db.persons.find().sort({name:-1,age:-1})

排序未使用到索引的情形

db.persons.find().sort({name:1,age:-1})

db.persons.find().sort({name:-1,age:1})

db.persons.find().sort({age:1,name:1})

db.persons.find().sort({age:-1,name:1})

4、复合索引与索引前缀

索引前缀指的是复合索引的子集

假如存在如下索引

{ "item": 1, "location": 1, "stock": 1 }

那存在下列索引前缀

{ item: 1 }

{ item: 1, location: 1 }

在MongoDB中，下列查询过滤条件情形中，索引将会被使用到

item字段

item字段 + location字段

item字段 + location字段 + stock字段

item字段 + location字段(尽管索引被使用，但不高效)

以下过滤条件查询情形，索引将不会被使用到

location字段

stock字段

location + stock字段

5、小结

a、复合索引是基于多个键(列)上创建的索引

b、复合索引在创建的时候可以为其每个键(列)来指定排序方法

c、索引键列的排序方法影响查询在排序时候的操作，方向一致或相反的才能被匹配

d、复合索引与前缀索引通常在匹配的情形下才能被使用

MongoClient.connect("...", function(err, db){

var c = db.collection('yourCollection');

var s = c.find({/\* your query \*/}).stream();

s.on('data', function(doc){

c.update({\_id: doc.\_id}, {$set: {name : doc.firstName + ' ' + doc.lastName}}, function(err, result) { /\* result == true? \*/} }

});

s.on('end', function(){

// stream can end before all your updates do if you have a lot

})

})