link: null
title: 珠峰架构师成长计划
description: 业务分层
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=47 sentences=105, words=865

1. 扩展mongoose模型

业务分层

> service(多个模型)->dao单个模型->model 模型定义

> service(多个模型)->dao单个模型->model (模型定义+扩展方法)

2. statics 对类进行扩展

根据用户名查找用户文档

```
PersonSchema.statics.findByUsername = function (username, callback) {
    return this.findOne({ username }, callback);
}
Person.findByUsername('zfpx', function (err, doc) {
    console.log(doc);
});
```

3. methods 对实例进行扩展

```
PersonSchema.methods.exist = function (callback) {
    let query = { username: this.username, password: this.password };
    return this.model('Person').findOne(query, callback);
}
let person = new Person({ username: 'zfpx', password: '123456', phone: '010-6255889', firstname: 'first', lastname: 'last' });
person.exist(function (err, doc) {
    console.log(err, doc);
});
```

4. virutal虚拟属性

- virtual是虚拟属性的意思，即原来Schema定义里是不存在该属性，后来通过virutal方法赋予的属性。
- Schema中定义的属性是要保存到数据库里的，而virtual属性基于已有属性做的二次定义。

> 模型属性 = Schema定义的属性+virtual属性

```
PersonSchema.virtual('area').get(function () {

    return this.phone.split('-')[0];
});
PersonSchema.virtual('number').get(function () {
    return this.phone.split('-')[1];
});
let Person = conn.model('Person', PersonSchema);
let person = new Person({ username: 'zfpx', password: '123456', phone: '010-6255889', firstname: 'first', lastname: 'last' });
console.log(person.fullname, person.area, person.number);
```

5. hook

在用户注册保存的时候，需要先把密码通过salt生成hash密码，并再次赋给password

```
PersonSchema.pre('save', function (next) {
    this.password = crypto.createHmac('sha256', 'zfpx').update(this.password).digest('hex');
    next();
});

PersonSchema.statics.login = function (username, password, callback) {
    password = crypto.createHmac('sha256', 'zfpx').update(password).digest('hex');
    return this.findOne({ username, password }, callback);
}

Person.login('zfpx', '123456', function (err, doc) {
    console.log(err, doc);
});
```

6. schema 插件

Schemas是可插拔的，也就是说，它们提供在应用预先打包能力来扩展他们的功能。

```
module.exports = exports = function lastModified(schema,options){
  schema.add({lastModify:Date});
  schema.pre('save',function(next){
    this.lastModify = new Date;
    next();
  });
  if(options&& options.index){
    schema.path('lastModify').index(options.index);
  }
}
```

```
let plugin = require('./plugin');
let Person = new Schema({});
Person.plugin(plugin,{index:true});
```

- Person 是用户自己定义的Schema

- Person.plugin 是为Person增加plugin
- plugin有2个参数
    - 插件对象 plugin
    - 配置项 {index:true}

```
schema.add({age:Number});
```

7.MongoDB 聚合

- MongoDB中聚合(aggregate)主要用于处理数据(诸如统计平均值,求和等)，并返回计算后的数据结果。有点类似sql语句中的 count(*)。
- MongoDB中聚合的方法使用aggregate()。**7.1 语法** aggregate() 方法的基本语法格式如下所示：

```
>db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)
```

现在我们通过以上集合计算每个作者所写的文章数，使用aggregate()计算结果如下：

```
> db.article.insert({uid:1,content:'1',visit:1});
> db.article.insert({uid:2,content:'2',visit:2});
> db.article.insert({uid:1,content:'3',visit:3});
```

```
 db.article.aggregate([{$group:{_id:'$uid',total:{$sum:1}}}]);
 { "_id" : 2, "total" : 1 }
{ "_id" : 1, "total" : 2 }
`
```

> select uid, count(*) total from article group by uid

表达式 描述 实例 $sum 计算总和。 db.article.aggregate([{$group : {_id : "$uid", num_tutorial : {$sum : "$visit"}}}]) $avg 计算平均值 db.article.aggregate([{$group : {_id : "$uid", num_tutorial : {$avg : "$visit"}}}]) $min 获取集合中所有文档对应值得最小值。 db.article.aggregate([{$group : {_id : "$uid", num_tutorial : {$min : "$visit"}}}]) $max 获取集合中所有文档对应值得最大值。 db.article.aggregate([{$group : {_id : "$uid", num_tutorial : {$max : "$visit"}}}]) $push 把某列的所有值都放到一个数组中 db.article.aggregate([{$group : {_id : "$uid", url : {$push: "$url"}}}]) $addToSet 返回一组文档中所有文档所选字段的全部唯一值的数组 db.article.aggregate([{$group : {_id : "$uid", url : {$addToSet : "$url"}}}]) $first 根据资源文档的排序获取第一个文档数据,可能为null db.article.aggregate([{$group : {_id : "$uid", first_url : {$first : "$url"}}}]) $last 根据资源文档的排序获取最后一个文档数据,可能为null db.article.aggregate([{$group : {_id : "$uid", last_url : {$last : "$url"}}}])

```
db.article.insert({uid:1,content:'3',url:'url1'});
db.article.insert({uid:1,content:'4',url:'url1'});
db.article.insert({uid:1,content:'5',url:'url2'});
把某列的所有值都放到一个数组中
db.article.aggregate([{$group : {_id : "$uid", url : {$push: "$url"}}}])
{ "_id" : 1, "url" : [ "url1", "url1", "url2"] }
```

管道在Unix和Linux中一般用于将当前命令的输出结果作为下一个命令的参数。 MongoDB的聚合管道将MongoDB文档在一个管道处理完毕后将结果传递给下一个管道处理。管道操作是可以重复的。

- $project：修改输入文档的结构。可以用来重命名、增加或删除字段，也可以用于创建计算结果以及嵌套文档。
- $match：用于过滤数据，只输出符合条件的文档。$match使用MongoDB的标准查询操作
- $limit：用来限制MongoDB聚合管道返回的文档数。
- $skip：在聚合管道中跳过指定数量的文档，并返回余下的文档。
- $unwind：将文档中的某一个数组类型字段拆分成多条，每条包含数组中的一个值。
- $group：将集合中的文档分组，可用于统计结果。
- $sort：将输入文档排序后输出。
- 修改输入文档的结构。可以用来重命名、增加或删除字段，也可以用于创建计算结果以及嵌套文档

```
db.article.aggregate(
  { $project : {
      _id:0,
      content : 1 ,
  }}
);
```

7.4.2 过滤文档

- 用于过滤数据，只输出符合条件的文档。$match使用MongoDB的标准查询操作

```
db.article.aggregate( [
  { $match : { visit : { $gt : 10, $lte : 200 } } },
  { $group: { _id: '$uid', count: { $sum: 1 } } }
]);
```

- 在聚合管道中跳过指定数量的文档，并返回余下的文档。 ````js var db = connect('school'); var vistors = []; for(var i=1;i

db.vistors.aggregate([{ $match : { visit : { $gt : 10, $lte : 200 } } }, { $group: { _id: '$uid', count: { $sum: 1 } } }, { $skip : 1 }]);

```
#### 7.4.5 $unwind
-
&#x5C06;&#x6587;&#x6863;&#x4E2D;&#x7684;&#x67D0;&#x4E00;&#x4E2A;&#x6570;&#x7EC4;&#x7C7B;&#x578B;&#x5B57;&#x6BB5;&#x62C6;&#x5206;&#x6210;&#x591A;&#x6761;&#xFF0C;
#x6BCF;&#x6761;&#x5305;&#x542B;&#x6570;&#x7EC4;&#x4E2D;&#x7684;&#x4E00;&#x4E2A;&#x503C;&#x3002;
-
&#x4F7F;&#x7528;$unwind&#x53EF;&#x4EE5;&#x5C06;weekday&#x4E2D;&#x7684;&#x6BCF;&#x4E2A;&#x6570;&#x636E;&#x90FD;&#x88AB;&#x5206;&#x89E3;&#x6210;&#x4E00;&#x4E2A;&#
x6587;&#x6863;,,&#x5E76;&#x4E14;&#x9664;&#x4E86;weekday&#x7684;&#x503C;&#x4E0D;&#x540C;&#x5916;,,&#x5176;&#x4ED6;&#x7684;&#x503C;&#x90FD;&#x662F;&#x76F8;&#x540C;&
#x7684;
```js
db.vistors.aggregate( [
    { $project : {_id:1,uid:1,type:1,visit:1}},
    { $match : { visit : { $gte : 1, $lte : 10 } } },
    { $unwind:'$type'}
]);
```

- 将集合中的文档分组，可用于统计结果。

```
db.vistors.aggregate( [
  { $project : {_id:1,uid:1,type:1,visit:1}},
  { $match : { visit : { $gte : 1, $lte : 10 } } },
  { $unwind:'$type'},
  { $group: { _id: '$uid', count: { $sum: 1 } } },
  { $sort: {_id:1} },
  { $skip : 5 },
  { $limit: 5 }
]);
```
```

```
Article.aggregate([
                        { $match : { visit : { $gt : 10, $lte : 200 } } },
                        { $group: { _id: '$uid', count: { $sum: 1 } } },
                        { $skip : 1 }
])
                .
```