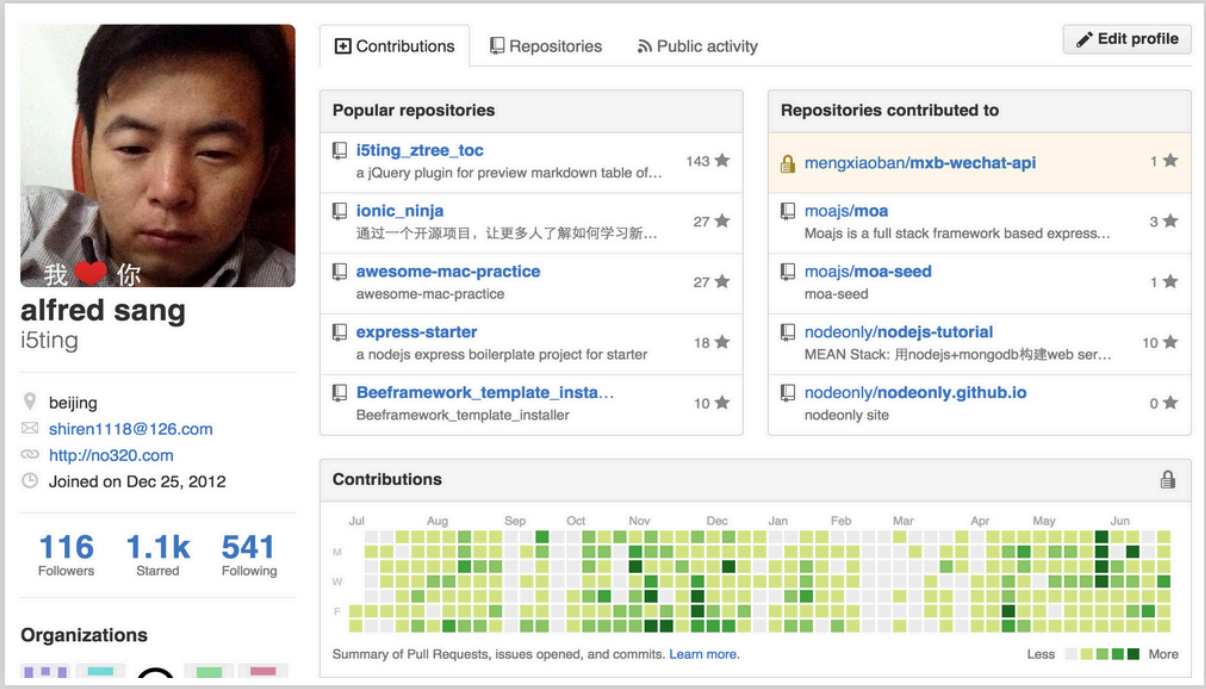
# NODEJS最新技术栈

Nodejs是比较简单的，只有你有前端js基础，那就按照我的办法来吧！一周足矣

Created by i5ting / @i5ting

# ABOUNT ME

## i5ting 一个开源爱好者



# 推荐技术栈

- express 4.x （express最新版本，初学者先别去碰koa）
- mongoose（mongodb）
- bluebird（Promise/A+实现）
- jade（视图层模板）
- mocha（测试）
- node-inspector(调试)

https://github.com/i5ting/express-starter

# PART 1：HTTP基础

## 了解http协议，尤其是express如何req

```
curl -d "a=1&b=2" http://127.0.0.1:3001/users/post
```

# HTTP是无状态的协议

tcp呢？

- 绝对地址和相对地址

```
相对地址的举例:
<a href="/visit">访谈</a>

绝对地址的举例:
<a href="http://www.xxxxx.cn/net/">广告网络平台</a>
```

- querystring

```
https://www.baidu.com/s?wd=querystring
```

in expressjs

```
req.query.wd
```

# URL 和 URI

## URI抽象结构

```
[scheme:]scheme-specific-part[#fragment]

[scheme:][//authority][path][?query][#fragment]

authority为[user-info@]host[:port]
```

# HTTP VERBS -1

verbs = 动词

http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html

```
app.get('/user:id', function (req, res) {
  res.send('Hello World!');
});

app.post('/user/create', function (req, res) {
  res.send('Got a POST request');
});

app.put('/user/:id', function (req, res) {
  res.send('Got a PUT request at /user');
});

app.delete('/user/:id', function (req, res) {
  res.send('Got a DELETE request at /user');
});
```

# HTTP VERBS -2

链式写法?

```
router.route('/')
  .get($.list)
  .post($.create);
```

all代表什么?

```
router.all('/new', $.new);
```

更多node里的verbs（26个），见

https://github.com/jshttp/methods/blob/master/index.js

# HTTP STATUS CODE

- http://www.restapitutorial.com/httpstatuscodes.html
- https://github.com/nodejs/io.js/blob/master/lib/_http_server.

```
500 : 'Internal Server Error',
403 : 'Forbidden',
404 : 'Not Found',
304 : 'Not Modified',
200 : 'OK',

app.get('/user/:id', function(req, res){
  res.status(200).json({
    a : 1,
    b : 2
  });
});
```

# REQ取参数的3种方法

expressjs里的请求参数，4.x里只有3种

- req.params
- req.body
- req.query
- 已经废弃的api：req.param

- req.params

```
app.get('/user/:id', function(req, res){
  res.send('user ' + req.params.id);
});
```

俗点：取带冒号的参数

- req.body

```
var app = require('express')();
var bodyParser = require('body-parser');

app.use(bodyParser.json()); // for parsing application/json
app.use(bodyParser.urlencoded({ extended: true })); // for parsing ap

app.post('/', function (req, res) {
  console.log(req.body);
  res.json(req.body);
})
```

可以肯定的一点是req.body一定是post请求，express里依赖的中间件必须有bodyParser，不然req.body是没有的。

- req.query

```
// GET /search?q=tobi+ferret
req.query.q
// => "tobi ferret"

// GET /shoes?order=desc&shoe[color]=blue&shoe[type]=converse
req.query.order
// => "desc"

req.query.shoe.color
// => "blue"

req.query.shoe.type
// => "converse"
```

query是querystring

- req.query不一定是get

```
// POST /search?q=tobi+ferret
{a:1,b:2}
req.query.q
// => "tobi ferret"
```

post里的数据看不到得，需要用req.body取

- 3种不同类型的post

- Post with x-www-form-urlencoded

see post.html

```
$.ajaxSetup({
  contentType: "application/x-www-form-urlencoded; charset=utf-8"
});

$.post("/users/post", { name: "i5a6", time: "2pm" },
   function(data){
     console.log(data);
   }, "json");
```

in routes/users.js

```
router.post('/post', function(req, res) {
  // res.send('respond with a resource');
    res.json(req.body);
});
```

# POST WITH FORM-DATA 主要目的是为了上传

```
var express = require('express')
var multer  = require('multer')

var app = express()
app.use(multer({ dest: './uploads/'}))
```

You can access the fields and files in the request object

```
router.post('/post-form-data', function(req, res) {
  console.log(req.body)
  console.log(req.files)
});
```

Multer will not process any form which is not multipart/form-data

- Post with raw

To get the raw body content of a request with Content-Type: "text/plain" into req.rawBody you can do:

```
var express = require('./')
var app = express();
app.use(function(req, res, next){
  if (req.is('text/*')) {
    req.text = '';
    req.setEncoding('utf8');
    req.on('data', function(chunk){ req.text += chunk });
    req.on('end', next);
  } else {
    next();
  }
});
app.post('/', function(req, res){
  res.send('got "' + req.text + '"');
});
app.listen(3000)
```

# 命令行玩法

```
#! /bin/bash

echo -n "post common"
curl -d "a=1&b=2" http://127.0.0.1:3001/users/post

echo -n 'post formdata'
curl -F 'pic=@"img/post-common.png"' -F 'a=1' -F 'b=2'  http://127.0.

echo -n 'post raw json'

curl -d "{"a":"1","b":"2","c":{"a":"1","b":"2"}}" http://127.0.0.1:3(
```

如不清楚，请man curl.

# SUPERTEST用法

https://github.com/expressjs/restful-router/blob/master/test/restful-router.test.js

```
var request = require('supertest');
var app = require('../example/app');

describe('restful-router.test.js', function () {
  it('should get /users/new => user.new', function (done) {
     request(app)
     .get('/users/new')
     .expect('GET /users/new => new, query: {}')
     .expect(200, done);
   });
 });
```

# WHAT IS REST?

参考阮一峰的文章

http://www.ruanyifeng.com/blog/2011/09/restful.html

```
/**
 * Auto generate RESTful url routes.
 *
 * URL routes:
 *
 *  GET    /topics[/]         => topic.list()
 *  GET    /topics/new        => topic.new()
 *  GET    /topics/:id        => topic.show()
 *  GET    /topics/:id/edit   => topic.edit()
 *  POST   /topics[/]         => topic.create()
 *  PATCH  /topics/:id        => topic.update()
 *  DELETE /topics/:id        => topic.destroy()
 *
 */
```

# RESTFUL API OR RES.API?

res.api is an express middleware for render json api , it convention over api format like this :

```
{
  data: {

  },
  status: {
    code : x,
    msg  : 'some message'
  }
}
```

- 客户端 API 开发总结
  https://cnodejs.org/topic/552b3b9382388cec50cf6d95
- res.api用法说明
  https://cnodejs.org/topic/55818a0c395a0c1812f18273

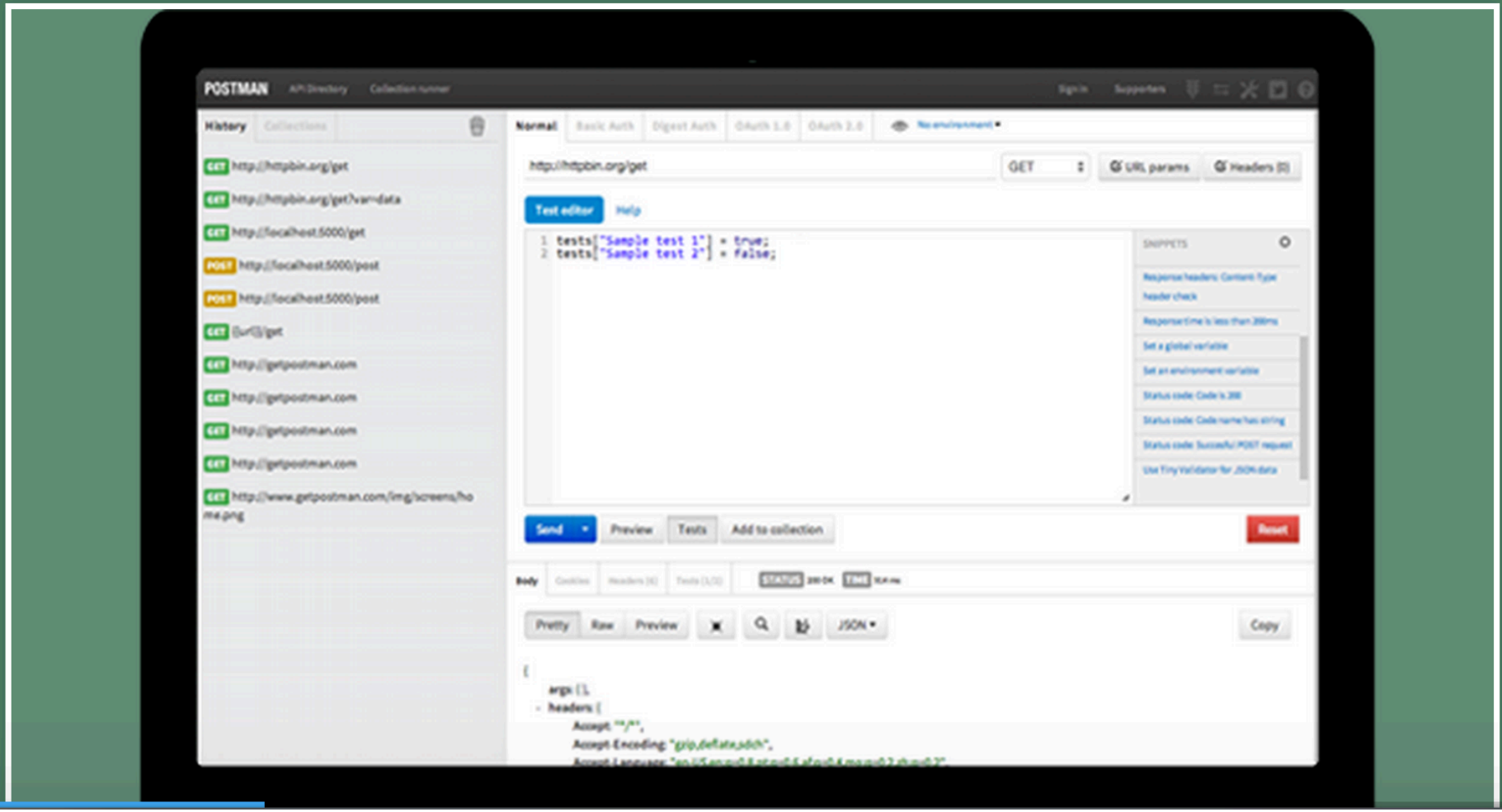
# 如果还不懂HTTP协议?

花点时间会用个软件就能学会，有兴趣吗?

# 如果还不懂HTTP协议?

## 花点时间会用个软件就能学会，有兴趣吗?

## 推荐Postman



# 如果还还不懂HTTP协议?

看书吧

# 如果还还不懂HTTP协议？

看书吧

推荐1 《HTTP下午茶》

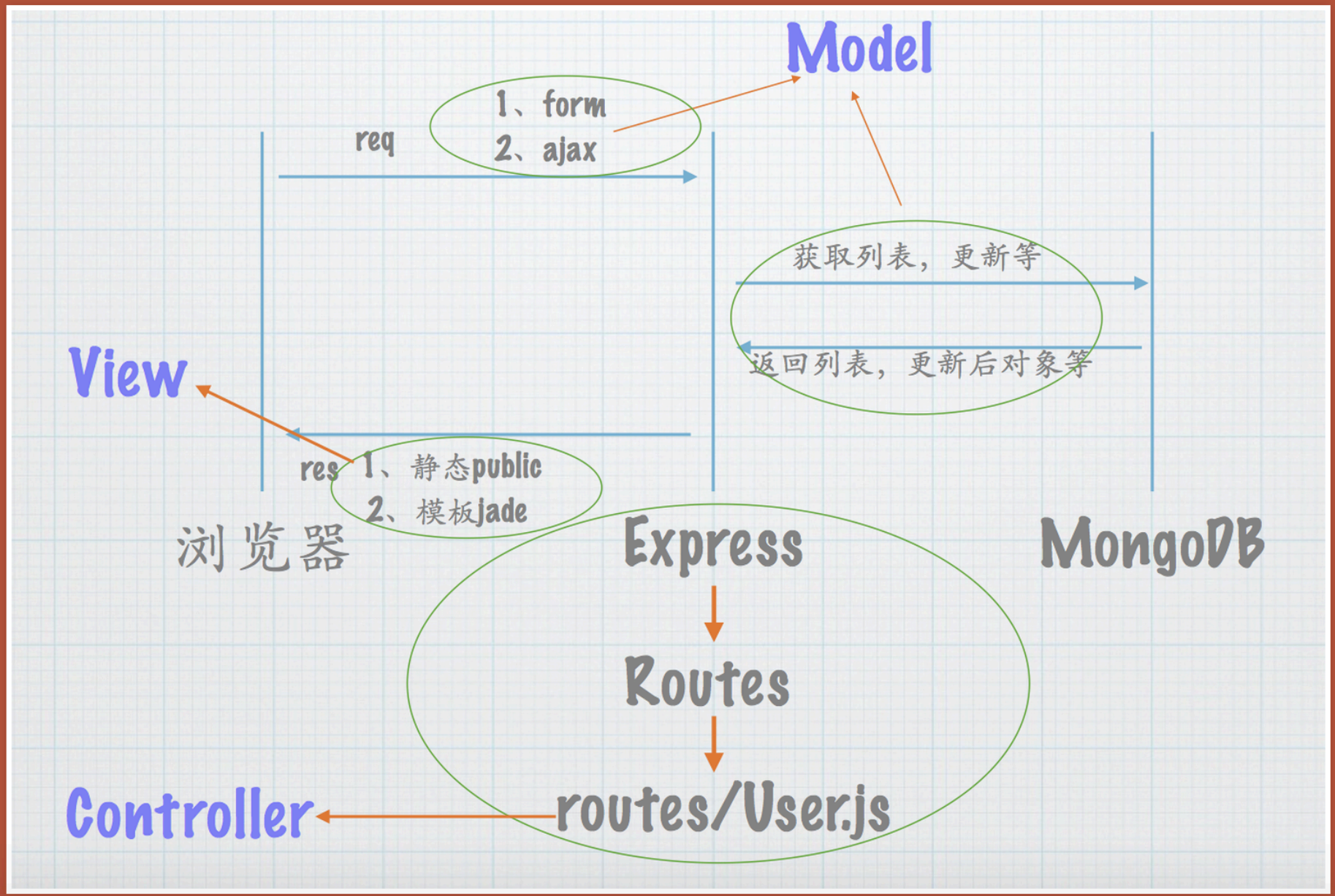
推荐2 《图解HTTP》

# PART 2： DB相关

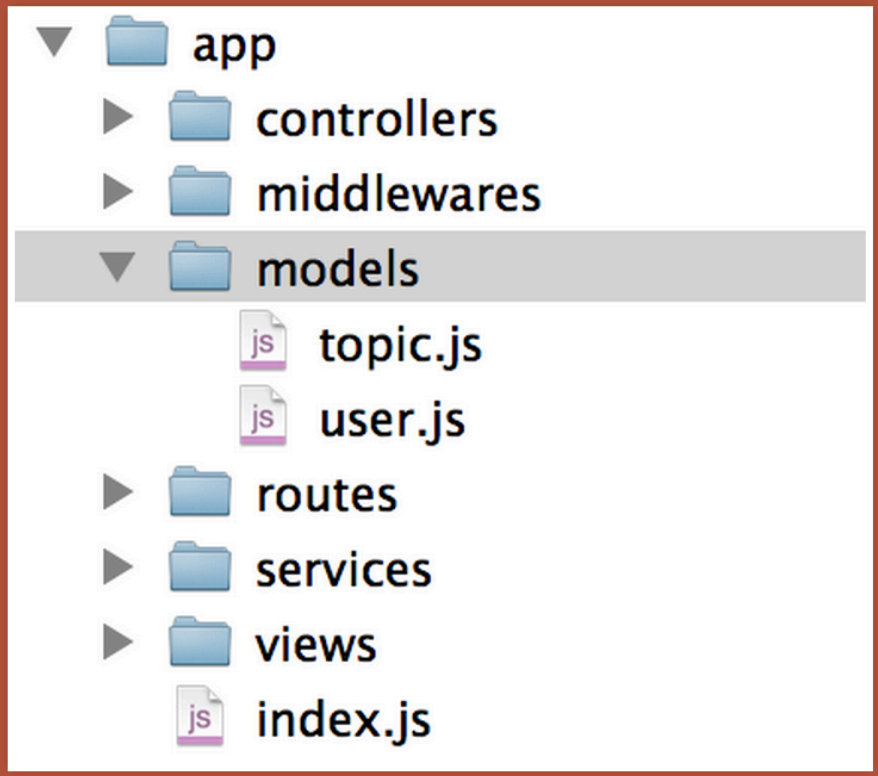
MEAN

- M = mongodb

- 了解mvc里m的作用


Model
1、form
2、ajax
req
获取列表，更新等
返回列表，更新后对象等
View
res
1、静态public
2、模板jade
浏览器
Express
MongoDB
Routes
Controller
routes/User.js

- 代码结构


app
controllers
middlewares
models
topic.js
user.js
routes
services
views
index.js

- mongoose用法

```
var mongoose = require('mongoose');
mongoose.connect('mongodb://localhost/test');

var Cat = mongoose.model('Cat', { name: String });

var kitty = new Cat({ name: 'Zildjian' });
kitty.save(function (err) {
  if (err) // ...
  console.log('meow');
});
```

# CRUD（增删改查）

- save
- find | findOne
- update
- remove

```
Kitten.find({ name: /^Fluff/ }, callback)
Comment.remove({ title: 'baby born from alien father' }, callback);
MyModel.update({ age: { $gt: 18 } }, { oldEnough: true }, fn);
```

- 了解分页

常见写法

```
db.users.find().skip(pagesize*(n-1)).limit(pagesize)
```

更好的写法

```
db.usermodels.find({'_id' :{
   "$gt" :ObjectId("55940ae59c39572851075bfd")}
 }).limit(20).sort({_id:-1})
```

- 了解关系（1对1，1对多）在mongoose里如何实现

```
UserSchema = new Schema({
    ...
    contacts:[]
});
```

- 了解关系（1对1，1对多，多对多）在mongoose里如何实现

```
ContactSchema = new Schema({
    ...
    owner: {
      type: Schema.ObjectId,
      required: true,
      index: true
    }
});
```

- 了解populate

```
ContactSchema = new Schema({
    ...
    owner: {
      type: Schema.ObjectId,
      ref: 'user'
    }
});

ContactSchema.find({}).populate('owner').exec(callback);
```

- 扩展mongoose模型：statics类方法

```
UserSchema.statics.find_by_openid = function(openid, cb) {
  return this.findOne({
    openid: openid
  }, cb);
};
```

调用

```
User.find_by_openid(openid, cb)
```

- 扩展mongoose模型：methods对象方法

```
UserSchema.methods.is_exist = function(cb) {
  var query;
  query = {
    username: this.username,
    password: this.password
  };
  return this.model('UserModel').findOne(query, cb);
};
```

调用

```
var user = new User({});
user.is_exist(cb)
```

- hook.js
- 了解pre和post的差别

```
UserSchema.pre('save', function(next) {
    var user = this;
    if (!user.isModified('password')) return next();
    bcrypt.genSalt(SALT_WORK_FACTOR, function(err, salt) {
      if (err) return next(err);
      bcrypt.hash(user.password, salt, function (err, hash) {
        if (err) return next(err);
          user.password = hash;
          next();
        });
    });
});
```

- 了解virtual属性

```
UserSchema.virtual('is_valid').get(function(){
  console.log('phone_number = ' +this.phone_number)
  if(this.phone_number == undefined | this.invite_code == undefined){
    return false;
  }
  return this.invite_code.length >= 2 && this.phone_number > 0
});
```

- 了解mongoose的插件机制

```
// lastMod.js
module.exports = exports = function lastModifiedPlugin (schema, optic
  schema.add({ lastMod: Date })

  schema.pre('save', function (next) {
    this.lastMod = new Date
    next()
  })

  if (options && options.index) {
    schema.path('lastMod').index(options.index)
  }
}
```

调用

```
// game-schema.js
var lastMod = require('./lastMod');
var Game = new Schema({ ... });
Game.plugin(lastMod, { index: true });
```

# AGGREGATION 关联

《SQL to Aggregation Mapping Chart》
http://docs.mongodb.org/manual/reference/sql-aggregation-comparison/

- 了解索引优化

```
ContactSchema = new Schema({
    ...
    owner: {
      type: Schema.ObjectId,
      required: true,
      index: true
    }
});
```

也可以这样的

```
ContactSchema.ensureIndexes(owner);
```

- 了解explain

```
db.usermodels.find({
  '_id' :{
    "$gt" :ObjectId("55940ae59c39572851075bfd")
  }
}).explain()

关注点

- stage：查询策略
- nReturned：返回的文档行数
- needTime：耗时（毫秒）
- indexBounds：所用的索引
```

- 了解profile

```
profile级别有三种:

- 0: 不开启
- 1: 记录慢命令，默认为大于100ms
- 2: 记录所有命令
- 3、查询profiling记录
```

开启

```
db.setProfilingLevel(2, 20)
```

默认记录在system.profile中

```
db['system.profile'].find()
```

# 了解MONGODB的部署

- replset
- shard

```
我写的《 mongodb运维之副本集实践》
https://cnodejs.org/topic/5590adbbebf9c92d17e734de
```

# MONGOOSEDAO

## 模型

```
var TopModel = mongoose.model('TopModel', TopSchema);
var TopModelDao = new MongooseDao(TopModel);
module.exports = TopModelDao;
```

## 用法

```
require('./db');

var User = require('./User');

User.create({"username":"sss","password":"password"},function(err, us
  console.log(user);
});

User.delete({"username":"sss"},function(err, user){
  console.log(user);
});
```

https://github.com/moajs/mongoosedao

# PART 3： PROMISE/A+规范

A promise is defined as an object that has a function as the value for the property then: then(fulfilledHandler, errorHandler, progressHandler)

```
http://promisesaplus.com/
```

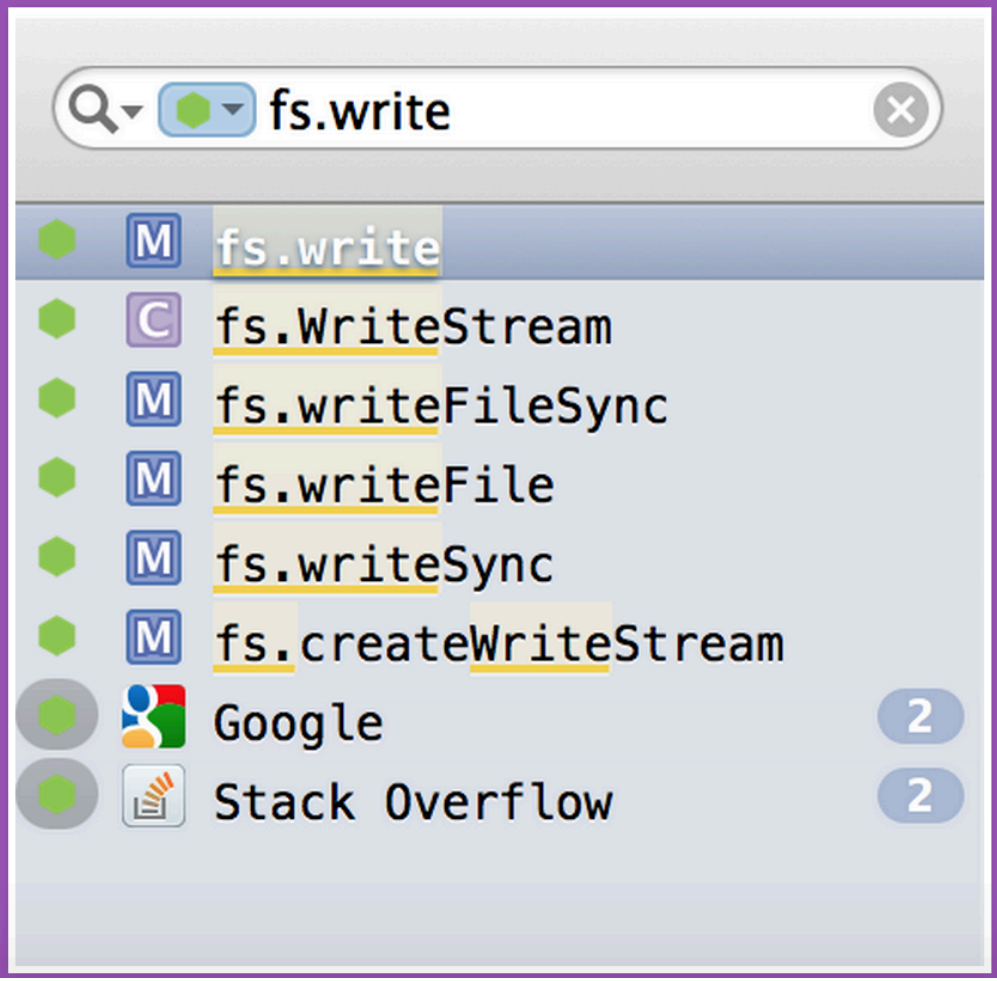
- 了解jquery里的then和$.deferred

```
function successFunc(){ console.log( “success!” ); }
function failureFunc(){ console.log( “failure!” ); }

$.when(
   $.ajax( "/main.php" ),
   $.ajax( "/modules.php" ),
   $.ajax( “/lists.php” )
).then( successFunc, failureFunc );
```

# 了解的NODE的异步

- 一切都是异步的
- 同步是奢求，要查Api



# 了解异步的恶心

1. *用户登录2. 增加用户日志3. 更新用户登录次数*

```
return User.login('username','password',fucntion(err, user){
    if(err){
        //
    }

    var log = new UserLog({
        uid: user._id,
        user_name: user.username
    })

    log.save(function(err, log){
        if(err){
            //
        }

        var loginCount = user.loginCount + 1;
        var options = {}
        User.findByIdAndUpdate(user._id, { loginCount: loginCount }, options, function(err, result){
           ....
        });
    });
});
```

# 如果用PROMISE写呢?

```
return User.loginAsync('username','password').then(function(user){
    var log = new UserLog({
        uid: user._id,
        user_name: user.username
    })
    return log.saveAsync();
}).then(log){
    var loginCount = user.loginCount + 1;
    var options = {}
    return User.findByIdAndUpdateAsync(user._id, { loginCount: loginCount }, options);
}).then(result){
    // last step
}).catch(function(err){
    // catch exception
});
```

# 如何实现一个PROMISE库

- 了解的then

```
var promise = {
  okCallbacks: [],
  koCallbacks: [],
  then: function (okCallback, koCallback) {
    okCallbacks.push(okCallback);
    if (koCallback) {
      koCallbacks.push(koCallback);
    }
  }
}
```

# 如何实现一个PROMISE库

- 了解的resolve和reject

```
var defer = {
  promise: promise,
  resolve: function (data) {
    this.promise.okCallbacks.forEach(function(callback) {
      window.setTimeout(function () {
        callback(data)
      }, 0);
    });
  },

  reject: function (error) {
    this.promise.koCallbacks.forEach(function(callback) {
      window.setTimeout(function () {
        callback(error)
      }, 0);
    });
  }
```

# 如何实现一个PROMISE库

- 测试代码

```
function test() {
  var defer = new Defer();
  // an example of an async call
  serverCall(function (request) {
    if (request.status === 200) {
      defer.resolve(request.responseText);
    } else {
      defer.reject(new Error("Status code was " + request.status));
    }
  });
  return defer.promise;
}
test().then(function (text) {
  alert(text);
}, function (error) {
  alert(error.message);
});
```

- 了解异步基本场景，比如waterfall这样的路程使用async如何处理

```
var async = require('async');
async.waterfall([
    function(callback) {
        callback(null, 'one', 'two');
    },
    function(arg1, arg2, callback) {
      // arg1 now equals 'one' and arg2 now equals 'two'
        callback(null, 'three');
    },
    function(arg1, callback) {
        // arg1 now equals 'three'
        callback(null, 'done');
    }
], function (err, result) {
    // result now equals 'done'
});
```

- 了解q和bluebird用法（如果有angularjs经验，推荐q，其他只推荐bluebird）

- 了解bluebird的promisifyAll用法

```
//Read more about promisification in the API Reference:
//API.md
var fs = Promise.promisifyAll(require("fs"));

fs.readFileAsync("myfile.json").then(JSON.parse).then(function (json)
    console.log("Successful json");
}).catch(SyntaxError, function (e) {
    console.error("file contains invalid json");
}).catch(Promise.OperationalError, function (e) {
    console.error("unable to read file, because: ", e.message);
});
```

- model上应用promisifyAll

```
1  var mongoose = require('mongoose');
2  var Schema = mongoose.Schema;
3  var Promise = require("bluebird");
4  var MongooseDao = require('mongooseDao');
5
6  TopSchema = new Schema({
7    username: String,
8    address: String,
9    created_at: {
10     type: Date,
11     "default": Date.now
12   }
13 });
14
15 var TopModel = mongoose.model('TopModel', TopSchema);
16
17 Promise.promisifyAll(TopModel);
18 Promise.promisifyAll(TopModel.prototype);
19
20 var TopDao = new MongooseDao(TopModel);
21
22 Promise.promisifyAll(TopDao);
23 // Promise.promisifyAll(TopDao.prototype);
24
25 module.exports = TopDao;
```

- 了解如何重构流程，以及代码的可读性

- controller上应用promisifyAll

```
var Promise = require('bluebird');

function order_detail(req, res) {
  var receive_data = req.params,
    order_id = receive_data.id,
    ReceiveDelivery = req.model.ReceiveDelivery,
    Product = req.model.Product;

  //查找此订单的信息
  ReceiveDelivery.findByIdAsync(order_id).then(function(receive) {
    var product_ids = [];
    for(var i = 0;i < receive.items.length; i++) {
      product_ids.push(receive.items[i].product);
    }

    Product.find_all_with_idsAsync(product_ids, function(err, products) {
      if(err){
        return Promise.reject(err);
      }

      receive.products = products
      return Promise.resove(receive);
    });
  }).then(function(receive) {
    res.status(200).json({
      data  : receive,
      status: {
```

- more

https://github.com/kriskowal/q/tree/v1/design

# PART 4：TDD/BDD测试

朴灵说：“不写测试的项目都不是好项目”

- 理解最小问题思想，培养程序员该有的强大的内心

- mocha的基本用法

```
npm install --save-dev mocha
```

```
https://github.com/mochajs/mocha

var assert = require("assert")

describe('truth', function(){
  it('should find the truth', function(){
    assert.equal(1, 1);
  })
})
```

- 理解断言

## 系统Api内置的assert

```
https://iojs.org/api/assert.html
```

### Should

```
chai.should();

foo.should.be.a('string');
foo.should.equal('bar');
foo.should.have.length(3);
tea.should.have.property('flavors')
  .with.length(3);
```

Visit Should Guide

### Expect

```
var expect = chai.expect;

expect(foo).to.be.a('string');
expect(foo).to.equal('bar');
expect(foo).to.have.length(3);
expect(tea).to.have.property('flavors')
  .with.length(3);
```

Visit Expect Guide

### Assert

```
var assert = chai.assert;

assert.typeOf(foo, 'string');
assert.equal(foo, 'bar');
assert.lengthOf(foo, 3)
assert.property(tea, 'flavors');
assert.lengthOf(tea.flavors, 3);
```

Visit Assert Guide

rspec里推荐用expect，其实看个人习惯

- 理解测试生命周期

```
describe('Test', function(){
  before(function() {
    // runs before all tests in this block
  })
  after(function(){
    // runs after all tests in this block
  })
  beforeEach(function(){
    // runs before each test in this block
  })
  afterEach(function(){
    // runs after each test in this block
  })
})
```

- 理解done回调

```
describe('MongooseDao', function(){
    before(function(done) {
      Top.deleteAll(function(err){
        if(err){
          console.log(err);
        }
        var files = [];
        for (var i = 0; i < 100; ++i) {
          files.push(Top.createAsync({"username":"fixture-user-" + i,
        }
        Promise.all(files).then(function() {
          done();
        });
    });
  })
```

- 生命周期方法有done
- 测试方法也有done

# 常用测试模块

1. mocha

2. chai（Chai is a BDD / TDD assertion library for node and the browser that can be delightfully paired with any javascript testing framework.

3. sinon（Standalone test spies, stubs and mocks for JavaScript.）

4. zombie (页面事件模拟Zombie.js is a lightweight framework for testing client-side JavaScript code in a simulated environment. No browser required.)

5. supertest(接口测试 Super-agent driven library for testing node.js HTTP servers using a fluent API)

# 反复测试?

CI = Continuous integration 持续集成

- jenkins
- travis-ci

教程 https://cnodejs.org/topic/558df089ebf9c92d17e73358

# 理解测试覆盖率

ci实际上解决了反复测试的自动化问题。

但是如何看我的程序里的每一个函数都测试到了呢?

## 安装

```
$ npm install -g istanbul
```

## 执行

```
$ istanbul cover my-test-script.js -- my test args
```

# 测试报告

```
./node_modules/.bin/istanbul cover ./node_modules/mocha/bin/_mocha
    #MongooseDao()
      ✓ should return ok when record create
      ✓ should return ok when record delete fixture-user
      ...
      ✓ should return ok when record query


  8 passing (50ms)

=============================== Coverage summary ===============================
Statements   : 47.27% ( 26/55 )
Branches     : 8.33% ( 1/12 )
Functions    : 60% ( 9/15 )
Lines        : 47.27% ( 26/55 )
================================================================================
```

- 理解基于gulp自动化测试方法

```
var gulp = require('gulp');
var watch = require('gulp-watch');
var path = 'test/**/*.js';

gulp.task('watch', function() {
  gulp.watch(['test/**/*.js', 'lib/*.js'], ['mocha']);
});

var mocha = require('gulp-mocha');
gulp.task('mocha', function () {
    return gulp.src(path , {read: false})
        .pipe(mocha({reporter: 'spec'}));
});

gulp.task('default',['mocha', 'watch']);
```

# MORE

如果有兴趣，可以去了解更多bdd/tdd内容，甚至是cucumber.js,从用户故事开始写测试

# PART 5：调试

# 3种方法

- node debugger
- node inspector
- 测试驱动开发

# NODE DEBUGGER -1

V8 提供了一个强大的调试器，可以通过 TCP 协议从外部访问。

Nodejs提供了一个内建调试器来帮助开发者调试应用程序。

```
var hello = 'hello';
var world = 'nodejs';

debugger;

var hello_world = hello + ' ' + world;
console.log(hello_world);
```

# NODE DEBUGGER -2

执行命令：

```
node debug helloword-debug.js
```

就可以进入调试模式。

# 全是命令行能习惯么？

```
node-debug-tutorial git:(master) ✗ node debug helloword-debug.js
< debugger listening on port 5858
connecting... ok
break in helloword-debug.js:1
  1 var hello = 'hello';
  2 var world = 'nodejs';
  3
debug> help
Commands: run (r), cont (c), next (n), step (s), out (o), backtrace (
watch, unwatch, watchers, repl, restart, kill, list, scripts, breakOr
debug>
debug> n
break in helloword-debug.js:2
  1 var hello = 'hello';
  2 var world = 'nodejs';
  3
  4 debugger;
```

官方文档

```
http://nodejs.org/api/debugger.html
```

- node inspector

```
https://github.com/node-inspector/node-inspector
```

## 安装

```
npm install -g node-inspector
```

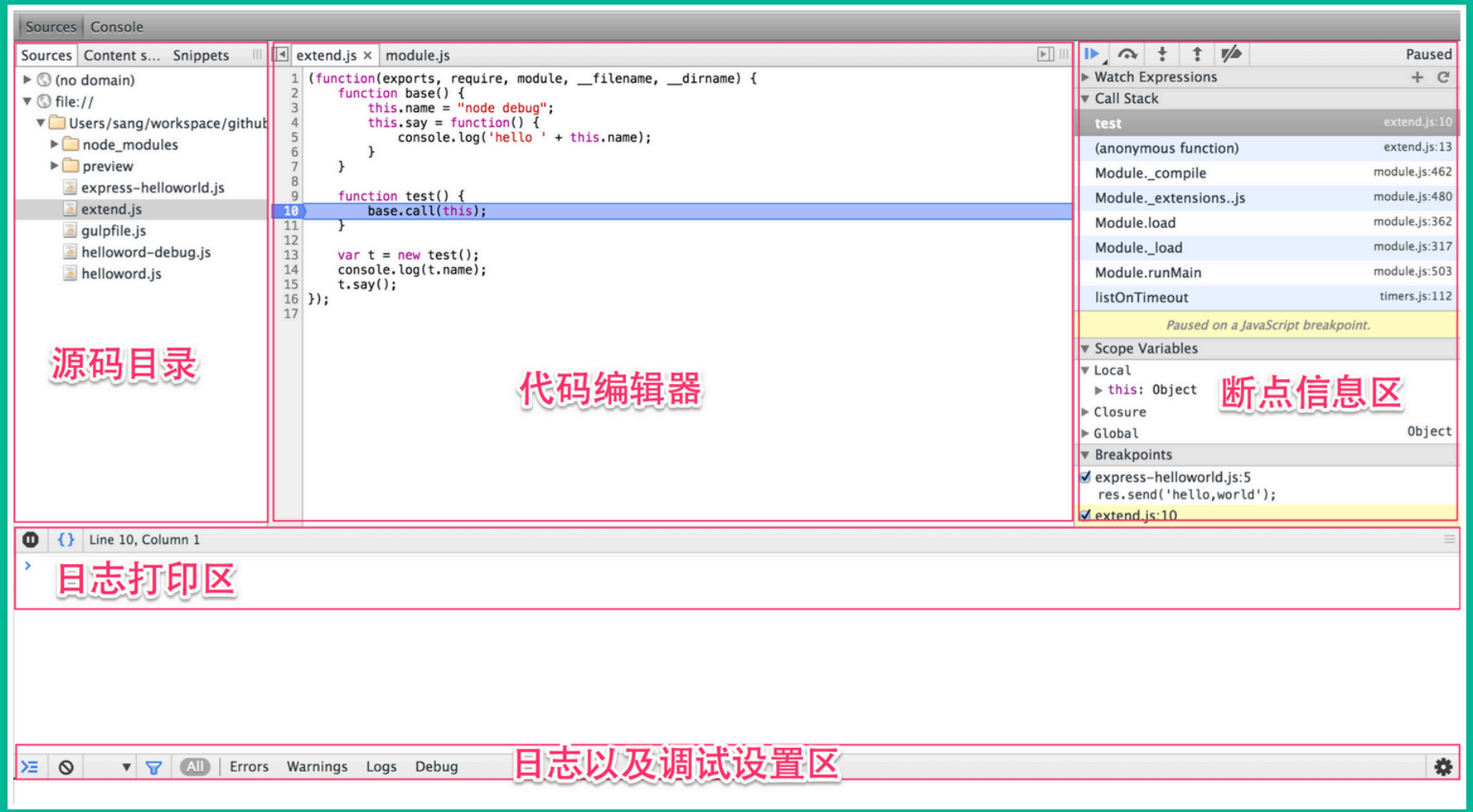
安装完成之后，通常可以直接这样启动在后台：

```
node-inspector &
```

我更喜欢这样用

```
node-debug app.js
```

# 可视化界面


Sources
Console
Sources
Content s...
Snippets
(no domain)
file://
Users/sang/workspace/github
node_modules
preview
express-helloworld.js
extend.js
gulpfile.js
helloword-debug.js
helloword.js
源码目录
extend.js ×
module.js
(function(exports, require, module, __filename, __dirname) {
function base() {
this.name = "node debug";
this.say = function() {
console.log('hello ' + this.name);
}
}
function test() {
base.call(this);
}
var t = new test();
console.log(t.name);
t.say();
});
代码编辑器
Paused
Watch Expressions
Call Stack
test extend.js:10
(anonymous function) extend.js:13
Module._compile module.js:462
Module._extensions..js module.js:480
Module.load module.js:362
Module._load module.js:317
Module.runMain module.js:503
listOnTimeout timers.js:112
Paused on a JavaScript breakpoint.
Scope Variables
Local
this: Object
Closure
Global Object
Breakpoints
express-helloworld.js:5
res.send('hello,world');
extend.js:10
断点信息区
Line 10, Column 1
日志打印区
All
Errors
Warnings
Logs
Debug
日志以及调试设置区

# 断点操作

- resume script execution（F8） 挂起断点，也可以理解为放弃当前断点，如果有下一个断点，会自动断住得
- step over（F10） 跳过这行，到下一行，如果当前函数结束，会跳到调用栈的上一级的下一行
- step into（F11） 进入当前行代码里的函数内部
- step out（Shift + F11） 从当前函数退出到之前进入的代码处

# 总结

- inspector 足够简单了，功能够用，需要断点的时候用
- tdd/bdd 平常做业务测试
- 如果想了解更详细的，可以看我写的教程和视频：node-debug 三法三例

```
https://cnodejs.org/topic/5463f6e872f405c829029f7e
```

# 回顾一下

- 了解http协议，尤其是express如何req
- 了解db相关操作，以mongoose为主
- 了解Promise/A+规范，合理规避回调陷阱
- 使用tdd/bdd测试，最小化问题
- 你无论如何都要会的：调试

# 招聘NODEJS工程师, 我亲自带

目标全栈

# Q & A

少抱怨，多思考，未来更美好。有的时候我看的不是你一时的能力，而是你面对世界的态度。

console.log('The End, Thanks~')