Node.js Express框架介绍

**Express 简介**

Express 是一个简洁而灵活的 node.js Web应用框架, 提供了一系列强大特性帮助你创建各种 Web 应用，和丰富的 HTTP 工具。

使用 Express 可以快速地搭建一个完整功能的网站。

Express 框架核心特性：

* 可以设置中间件来响应 HTTP 请求。
* 定义了路由表用于执行不同的 HTTP 请求动作。
* 可以通过向模板传递参数来动态渲染 HTML 页面。
* **body-parser** - node.js 中间件，用于处理 JSON, Raw, Text 和 URL 编码的数据。
* **cookie-parser** - 这就是一个解析Cookie的工具。通过req.cookies可以取到传过来的cookie，并把它们转成对象。
* **multer** - node.js 中间件，用于处理 enctype="multipart/form-data"（设置表单的MIME编码）的表单数据。

## 路由

我们已经了解了 HTTP 请求的基本应用，而路由决定了由谁(指定脚本)去响应客户端请求。

在HTTP请求中，我们可以通过路由提取出请求的URL以及GET/POST参数。

## 静态文件

Express 提供了内置的中间件**express.static**来设置静态文件如：图片， CSS, JavaScript 等。

你可以使用**express.static**中间件来设置静态文件路径。例如，如果你将图片， CSS, JavaScript 文件放在 public 目录下，你可以这么写：

app.use(express.static('public'));

为了在无状态的 HTTP 协议之上实现会话，Cookie 诞生了。Cookie 是一些存储在客户端的信息，每次连接的时候由浏览器向服务器递交，服务器也向浏览器发起存储 Cookie 的请求，依靠这样的手段服务器可以识别客户端。我们通常意义上的 HTTP 会话功能就是这样实现的。具体来说，浏览器首次向服务器发起请求时，服务器生成一个唯一标识符并发送给客户端浏览器，浏览器将这个唯一标识符存储在 Cookie 中，以后每次再发起请求，客户端浏览器都会向服务器传送这个唯一标识符，服务器通过这个唯一标识符来识别用户。 对于开发者来说，我们无须关心浏览器端的存储，需要关注的仅仅是如何通过这个唯一标识符来识别用户。很多服务端脚本语言都有会话功能，如 PHP，把每个唯一标识符存储到文件中。

express 也提供了会话中间件，默认情况下是把用户信息存储在内存中，但我们既然已经有了 MongoDB，不妨把会话信息存储在数据库中，便于持久维护。为了使用这一功能，我们需要借助 express-session 和 connect-mongo 这两个第三方中间件，

var session = require('express-session');

var MongoStore = require('connect-mongo')(session);

app.use(session({

secret: settings.cookieSecret,

key: settings.db,//cookie name

cookie: {maxAge: 1000 \* 60 \* 60 \* 24 \* 30},//30 days

store: new MongoStore({

db: settings.db,

host: settings.host,

port: settings.port

})

}));

使用 express-session 和 connect-mongo 模块实现了将会化信息存储到mongoldb中。secret 用来防止篡改 cookie，key 的值为 cookie 的名字，通过设置 cookie 的 maxAge 值设定 cookie 的生存期，这里我们设置 cookie 的生存期为 30 天，设置它的 store 参数为 MongoStore 实例，把会话信息存储到数据库中，以避免丢失。在后面的小节中，我们可以通过 req.session 获取当前用户的会话对象，获取用户的相关信息。

<http://www.open-open.com/lib/view/open1454560780730.html#articleHeader15>

4. Express的Router对象

当express实例的路由越来越多的时候，最好把路由分类独立出去，express的实例(app) 能更好的处理其他逻辑流程。Express的Router对象是一个简化的 app实例，只具有路由相关的功能，包括use, http verbs等等。最后这个Router再通过app的use挂载到app的相关路径下。

router的路由必须通过app.use和app.verbs 挂载到app上才能被响应。所以上述代码，只有在app捕捉到 /foo路径上的路由时，才能router中定义的路由，虽然router中有针对 '/' 的路由，但是被app中的路由给覆盖了。

<http://www.expressjs.com.cn/guide/routing.html>

1、例如：**127.0.0.1:3000/index**，这种情况下，我们为了得到index，我们可以通过使用**req.params**得到，通过这种方法我们就可以很好的处理Node中的路由处理问题，同时利用这点可以非常方便的实现MVC模式；

2、例如：**127.0.0.1:3000/index?id=12**，这种情况下，这种方式是获取客户端get方式传递过来的值，通过使用**req.query.id**就可以获得，类似于PHP的get方法；

3、例如：**127.0.0.1：300/index，然后post了一个id=2的值**，这种方式是获取客户端post过来的数据，可以通过**req.body.id**获取，类似于PHP的post方法；

Express 依赖 bodyParser 对请求的包体进行解析，默认支持：application/json, application/x-www-form-urlencoded, multipart/form-data.可惜对xml没有支持,需要自己代码来实现,下面我们一一简单介绍.

(一) www-form-urlencoded

http默认的post请求是这种方式,注意这是是默认的提交方式,比如你写一个<form>....<input type="submite" /></form> form表单,里面的submite按钮默认就是这种 www-form-urlencoded 方式提交的.

**nodejs之async异步编程**

# 1.什么是异步编程？

异步编程是指由于异步I/O等因素，无法同步获得执行结果时， 在回调函数中进行下一步操作的代码编写风格,常见的如setTimeout函数、ajax请求等等。

示例： for (var i = 1; i <= 3; i++) { setTimeout(function(){ console.log(i); }, 0); };

这里大部分人会认为输出123，或者333。其实它会输出 444

这里就是我们要说的异步编程了。

高级函数的定义

这里为什么会说到高级函数，因为高级函数是异步编程的基础。

那什么是高级函数呢？ 其实高级函数就是把函数作为参数或者是作为返回值。

示例： function test(v){ return function(){ return v; } }

如上就是把一个函数作为一个返回值。

## series(tasks, [callback]) （多个函数依次执行，之间没有数据交换）

<http://my.oschina.net/huangsz/blog/176203>

有多个异步函数需要依次调用，一个完成之后才能执行下一个。各函数之间没有数据的交换，仅仅需要保证其执行顺序。这时可使用series。   
纯js代码：

step1(**function**(err, v1) {

step2(**function**(err, v2) {

step3(**function**(err, v3) {

*// do somethig with the err or values v1/v2/v3*

}

}

});

从中可以看到这嵌套还是比较多深的，如果再多几步，会更深。在代码中忽略对了每一层err的处理，否则还都等加上 if(err) return callback(err)，那就更麻烦了。   
对于这种情况，使用async来处理，就是这样的：

**var** **async** = **require**(‘**async**’) **async**.series([step1, step2, step3],

**function**(err, values) *{*

*// do somethig with the err or values v1/v2/v3*

*}*);

可以看到代码简洁了很多，而且自动处理每个回调中的错误。当然，这里只给出来最最简单的例子，在实际中，我们常会在每个step中执行一些操作，这时可写成：

var async = **require**(‘async’) async.series([**function**(cb) {

step1(**function**(err, v1) {

// do something with v1

cb(err, v1);

}),

**function**(cb) {

step2(**...**)

},

**function**(cb) {

step3(**...**)

}],

**function**(err, values) {

// do somethig with the err or values v1/v2/v3

});

该函数的详细解释为：   
依次执行一个函数数组中的每个函数，每一个函数执行完成之后才能执行下一个函数。   
如果任何一个函数向它的回调函数中传了一个error，则后面的函数都不会被执行，并且将会立刻会将该error以及已经执行了的函数的结果，传给series中最后那个callback。   
当所有的函数执行完后（没有出错），则会把每个函数传给其回调函数的结果合并为一个数组，传给series最后的那个callback。   
还可以json的形式来提供tasks。每一个属性都会被当作函数来执行，并且结果也会以json形式传给series最后的那个callback。这种方式可读性更高一些。   
具体例子可参考：https://github.com/freewind/async\_demo/blob/master/series.js   
其代码中还包含了：   
如果中间某个函数出错，series函数如何处理   
如果某个函数传给回调的值为undefined, null, {}, []等，series如何处理   
另外还需要注意的是：多个series调用之间是不分先后的，因为series本身也是异步调用。