# 一 简介

● 原生Node开发，会发现有很多问题。比如：

■ 呈递静态页面很不方便，需要处理每个HTTP请求，还要考虑304问题

■ 路由处理代码不直观清晰，需要写很多正则表达式和字符串函数

● EXPRESS的哲学是在你的想法和服务器之间充当薄薄的一层。这并不意味着他不够健壮，或者没有足够的有用特性，而是尽量少干预你，让你充分表达自己的思想，同时提供一些有用的东西。

英语官网：http://expressjs.com/

中文官网：<http://www.expressjs.com.cn/>

注意：Express4.X和Express3.X差别非常大。

安装Express框架，就是使用npm的命令：npm install express --save

# 二 路由

## 1 路由展示

const express = require('express');  
let app = express();  
  
app.get('/',function (req,res) {  
 res.send('hi');  
});  
  
app.get('/showname',function () {  
 res.send('name si lisi');  
});  
  
app.get(/^\/student\/([\d]{10})$/,function(req,res){  
 res.send("学生信息，学号" + req.params[0]);  
});  
  
app.get("/teacher/:gonghao",function(req,res){  
 res.send("老师信息，工号" + req.params.gonghao);  
});  
  
app.listen(3000);

## 2 路由使用介绍

|  |
| --- |
| app.get("网址",function(req,res){ }); //get请求  app.post("网址",function(req,res){ }); //post请求  app.all("网址",function(){ }); //处理这个网址的任何类型请求 |

这里的网址，不分大小写；

所有的GET参数，? 后面的都已经被忽略，锚点#也被忽略，路由到/a ， 实际/a?id=2&sex=nan 也能被处理；

正则表达式可以被使用。正则表达式中，未知部分用圆括号分组，然后可以用req.params[0]、[1]得到，req.params是一个类数组对象。

app.get(**/^\/student\/([\d]{10})$/**,function(req,res){

res.send("学生信息，学号" + **req.params[0]**);

});

表单可以自己提交到自己上。

|  |
| --- |
| app.get("/",function(req,res){  res.render("form");  });  app.post("/",function(req,res){  //将数据添加进入数据库  res.send("成功");  }); |

适合进行 RESTful路由设计。简单说，就是一个路径，但是http method不同，对这个页面的使用也不同。

/student/345345

## 3 参数获取

GET请求的参数在URL中，在原生Node中，需要使用url模块来识别参数字符串。在Express中，不需要使用url模块了。可以直接使用req.query对象。

POST请求的参数在express中不能直接获得，必须使用body-parser模块。使用后，将可以用req.body得到参数。但是如果表单中含有文件上传，那么还是需要使用formidable模块。

## 3 next()

如果get、post回调函数中，没有next参数，那么就匹配上第一个路由，就不会往下匹配了。如果想往下匹配的话，那么需要写**next()**

|  |
| --- |
| app.get("/",function(req,res,**next**){  console.log("1");  **next();**  });  app.get("/",function(req,res){  console.log("2");  }); |

## 4 冒号参数引起的路由冲突的解决

下面两个路由，感觉没有关系：

|  |
| --- |
| app.get("**/:username/:id**",function(req,res){ //express推荐使用冒号获得参数  console.log("1");  res.send("用户信息" + req.params.username);  });  app.get("**/admin/login**",function(req,res){  console.log("2");  res.send("管理员登录");  }); |

但是实际上冲突了，因为admin可以当做用户名 login可以当做id。

解决方法1：交换位置。 也就是说，express中所有的路由（中间件）的顺序至关重要。

匹配上第一个，就不会往下匹配了。 具体的往上写，抽象的往下写。

|  |
| --- |
| app.get("/admin/login",function(req,res){  console.log("2");  res.send("管理员登录");  });  app.get("/:username/:id",function(req,res){  console.log("1");  res.send("用户信息" + req.params.username);  }); |

解决方法2：

|  |
| --- |
| app.get("/:username/:id",function(req,res,**next**){  var username = req.params.username;  //检索数据库，如果username不存在，那么next()  if(检索数据库){  console.log("1");  res.send("用户信息");  }else{  **next();**  }  });  app.get("/admin/login",function(req,res){  console.log("2");  res.send("管理员登录");  }); |

# 三 常用技术

## 1 静态服务

app.use(express.static("./public"));

一句话，就将public文件夹变为了静态服务文件夹。当然也可以指定访问路由名称：

app.use('/test',express.static('public')); //test是虚拟路径

## 2 模板引擎使用

app.set("view engine","ejs");  
app.get("/",function(req,res){  
 res.render("haha",{  
 "news" : ["我是小新闻啊","我也是啊","哈哈哈哈"]  
 });  
});

但是上述模板引擎定义后，我们使用的模板依然需要修改后缀为设定的引擎后缀，进行下列设定，无需修改html后缀：

app.engine('html',ejs.renderFile);

app.set('view engine','html');

# 四 中间件

## 0 中间件介绍

路由get、post这些东西，就是中间件，中间件讲究顺序，匹配上第一个之后，就不会往后匹配了，next函数才能够继续往后匹配。

app.use()也是一个中间件。与get、post不同的是，他的网址不是精确匹配的，而是能够有小文件夹拓展的。使用user(path,fn())时候，user内的路由，将匹配/path /path/images/ /path/images/1.png 等路由情况。

比如网址： http://127.0.0.1:3000/admin/aa/bb/cc/dd

|  |
| --- |
| app.use("/admin",function(req,res){  res.write(req.originalUrl + "\n"); // /admin/aa/bb/cc/dd  res.write(req.baseUrl + "\n"); // /admin  res.write(req.path + "\n"); // /aa/bb/cc/dd  res.end("你好");  }); |

如果写一个 / 实际上就相当于"/"，就是所有网址

|  |
| --- |
| app.use(function(req,res,next){  console.log(new Date());  next();  }); |

app.use()就给了我们增加一些特定功能的便利场所，实际上app.use()的东西，基本上都从第三方能得到。

● 大多数情况下，渲染内容用res.render()，将会根据views中的模板文件进行渲染。如果不想使用views文件夹，想自己设置文件夹名字，那么app.set("views","aaaa");

● 如果想写一个快速测试页，当然可以使用res.send()。这个函数将根据内容，自动帮我们设置了Content-Type头部和200状态码。send()只能用一次，和end一样。

● 如果想使用不同的状态码，可以：

res.status(404).send('Sorry, we cannot find that!');

● 如果想使用不同的Content-Type，可以：

res.set('Content-Type', 'text/html');

大坑：express的中间件函数，不需要传入req，res，他是在中间件函数执行回调的时候自动传入：



main路由下的showIndex函数不需要传入req，res。

## 1 中间件的基本使用

const express = require('express');  
  
*//自定义中间件1*function logger1(req,res,next) {  
 console.log('log1...');  
 next(); *//没有next就不会执行下一个*}  
  
*//自定义中间件2*function logger2(req,res,next) {  
 console.log('log2...');  
 next(); *//没有next就不会执行下一个*}  
  
let app = express();  
app.use(logger1);  
app.use('/log',logger2);  
  
app.get('/log',function (req,res) {  
 res.end('log');  
});  
app.listen(3000);

## 2 创建可配置的中间件

*//setup函数可以用不同配置多次调用*

function setup(options) {

*//设置逻辑* return function (req,res,next) {  
 *//中间件逻辑* }  
}

## 3 可配置中间件案例

/\*  
实现可配置的logger中间件需要先定义一个setup函数，能接收一个字符串参数format，  
调用setup后返回一个函数，即真正的中间件，因为闭包的原因，这个返回的函数仍然可以使用format参数，  
然后lgger会使用req对象中关联的请求属性替换format中的属性。  
 \*/  
function setup(format) {  
 let regexp = /:(\w+)/g; *//使用正则匹配请求属性* return function logger(req,res,next) { *//真实的logger组件* let str = format.replace(regexp,function (match,property) {  
 return req[property];  
 });  
 console.log(str);  
 next();  
 }  
}  
module.exports = setup;

## 4 常见中间件 cookieParser

cookie与项目中介绍

## 5 app.use 与 app.all区别

****app.use**** only sees whether url starts with the specified path

app.use( "/product" , mymiddleware);

// will match /product

// will match /product/cool

// will match /product/foo

****app.all**** will match complete path

app.all( "/product" , handler);

// will match /product

// won't match /product/cool <-- important

// won't match /product/foo <-- important

app.all( "/product/\*" , handler);

// won't match /product <-- Important

// will match /product/

// will match /product/cool

// will match /product/foo

# 五 ajax2.0 上传文件 给express

## 1 前台上传代码

<input type="file" id="myfile" multiple><br>  
<input type="submit" value="上传" id="btn">  
<script>  
 let btn = document.querySelector('#btn');  
 let myFile = document.querySelector('#myfile');  
 btn.onclick = function () {  
  
 *//获取上传文件* let data = new FormData();  
 *//myfile.value只是文件名  
 // console.log(myfile.files);  
 //files只是类似数组，没有forEach方法，不能直接使用* Array.from(myFile.files).forEach(file=>{  
 data.append('f1',file);  
 });  
  
 *//实现上传* let ajax = new XMLHttpRequest();  
 ajax.open('POST',`http://localhost:8000/upload`,true);  
 *//FormData提交，下列语句并没有意义，服务器永远以文件上传形式解析  
 //ajax.setRequestHeader('Content-Type','application/x-www-form-urlencoded');  
 //发送数据* ajax.send(data);  
  
 ajax.onreadystatechange=function (){  
 if(ajax.readyState==4){  
 if(ajax.status>=200 && ajax.status<300 || ajax.status==304){  
 alert('成功');  
 }else{  
 alert('失败');  
 }  
 }  
 };  
 }  
</script>

## 2 express后台接收代码

const express=require('express'); *//主体*const body=require('body-parser'); *//接收普通POST数据*const multer=require('multer'); *//接收文件POST数据*let server=express();  
server.listen(8000);  
  
server.use(body.urlencoded({extended: false}));  
  
let multerObj=multer({dest: './upload/'});  
server.use(multerObj.any());  
  
*//处理请求*server.post('/upload', (req, res)=>{  
 if(req.headers['origin']=='null' || req.headers['origin'].startsWith('http://localhost')){  
 res.setHeader('Access-Control-Allow-Origin', '\*');  
 }  
  
 res.send("OK");  
  
 console.log(req.body); *//普通POST数据* console.log(req.files); *//文件POST数据*});  
  
server.use(express.static('./public/'));

## 3 ajax2.0监控文件上传进度

使用upload对象可以查看上传进度：

ajax.upload.onprogress = function(ev){

ev.loaded 完成

ev.total 总共

};

（如果出现下列BUG--有待考证：

使用了upload对象后，服务端除了需要处理正常的文件上传POST请求外，需要额外处理一个请求OPTIONS，我们只需要将后台的对应 post改为use即可。）

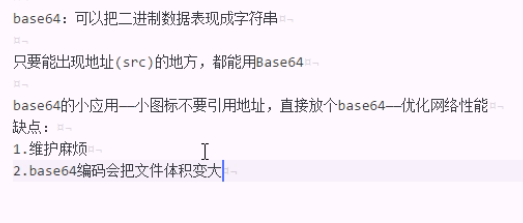
进度监控代码：注意这段代码要放在ajax的send方法之前

*//监控进度*ajax.upload.onprogress = function (ev) {  
 *// console.log(ev);* console.log(ev.loaded/ev.total);  
};

官方推荐使用事件绑定形式添加：



## 4 关于base64



## 5 后台跨域限制

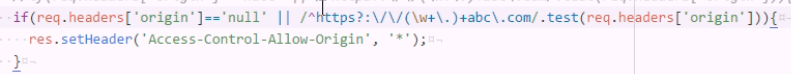
在后台，其实跨域是不存在的，无论什么样的域名请求，浏览器都会收到后台的结果，只是发生跨域后，浏览器不展示给请求的用户看而已。

node后台设置跨域与否：



\*指任意域都可以访问，通常只会写某些地址。

通常的写法：



在ajax1.0中可以给服务器发送一个自定义头，让服务器解析：

