node，npm，express 启动

1. node启动

- 创建 index.js 或 app.js 文件，作为入口文件；
- 使用 npm init 初始化项目，创建package.json文件；
- 使用 npm install express --save 安装需要的模块；
- 使用 node index.js（app.js） 就可以启动node 项目；

2. npm启动

- 创建 index.js 或 app.js 文件，作为入口文件；
- 使用 npm init 初始化项目，创建package.json文件；
- 使用 npm install express --save 安装需要的模块；
- 在 package.json 的 scripts 添加 "start": "node app.js"
- 使用 npm start 就可以启动；

3. express启动

- 使用 npm install express-generator -g ；
- 使用 express projectName 得到一个express 生成的项目结构；
- 使用 npm start 启动

常见问题

1. 使用这个指令，没有出现node_modulesw文件夹，由于是全局安装，所以只会安装在node 的安装目录，需要使用 npm install express --save;
2. npm install express 是默认在package.json 添加依赖库，要想在开发环境下添加（devDependencies）,需要使用npm install express --save--dev

node的基础

1.创建简单的node 服务

创建一个项目文件夹，例如myapp， 创建index.js文件，执行npm init

- 使用var http = require("http");引用http模块；
- 创建服务http.createServer；
- 使用listen监听和启动服务；
- 在回调中设置响应res

```
var http = require("http");

var server = http.createServer(function (req,res) {
    res.writeHead(200,{"Content-type":"text/html;charset=UTF-8"});
    console.log("Hello world");
    res.end("Hello world");

});

server.listen(3000,"localhost");
```

执行 *node index.js*,在浏览器打开 *http://localhost:3000/*

2.req.url 返回不同的数据

- req.url是请求的路径，从端口号后，如：http://localhost:3000/home?name=hello，则 req.url 的值为“/home?name=hello”
- 在回调函数中，可以根据req.url的请求地址，返回不同的内容，如：请求http://localhost:3000/home，返回"主页"

```
var http = require("http");

var server = http.createServer(function (req,res) {
```

```
    console.log("服务器接收到了请求，地址为：" + req.url);

    if(req.url == "/home") {
        res.writeHead(200,{"Content-type":"text/html;charset=UTF-8"});
        res.end("<h1>主页</h1>");
    }else if(req.url == "/nav") {
        res.writeHead(200,{"Content-type":"text/html;charset=UTF-8"});
        res.end("<h1>导航</h1>");
    }}else {
        res.writeHead(404,{"Content-type":"text/html;charset=UTF-8"});
        res.end("<h1>页面不存在</h1>");
    }
});

server.listen(3000,"127.0.0.1");
```

3.静态文件的渲染

- 首先创建一个home.html的文件，并创建home.css的css文件；
- 使用var fs = require("fs"); fs.readFile来获取文件，并返回;
- readFile 函数的第一个参数为文件路径，第二个参数为回调函数，当获取成功后，将回调该方法；

index.js文件:

```
var http = require("http");
var fs = require("fs");

var server = http.createServer(function (req,res) {
    if (req.url == "/home") {
        fs.readFile("./home.html",function(err,data){
            res.writeHead(200,{"Content-type":"text/html;charset=UTF-8"});
            res.end(data);
        });
    }else if(req.url == "/pulic/home.css") {
        fs.readFile("./resource/css.css",function (err,data) {
            res.writeHead(200,{"Content-type":"text/css"});  //不是UTF-8
            res.end(data);
        });
    }else {
        res.writeHead(404,{"Content-type":"text/html;charset=UTF-8"});
        res.end("请求地址资源不存在");
```

```
    }
});

server.listen(3000,"127.0.0.1");
```

上面代码中，由于css文件也是需要请求的，所以必须在使用readFile返回

4. req 相关参数和方法

例如：请求http://localhost:3000/home?name=chase&age=22

- req包含用户请求的相关参数，使用var url = require("url"); 可以将参数字符串转换为对象；
- req.url　输出: /home?name=chase&age=22
- url.parse(req.url).query　输出: name=chase&age=22
- url.parse(req.url, true).query　输出: { name: 'chase', age: '22' }
- url.parse(req.url, true).query.name　输出：chase

5. 文件，文件夹的相关操作

- 文件夹的子文件遍历，fs的readdir可以遍历一个指定的文件夹;
- fs.stat 可以读取指定文件的相关信息，stats.isDirectory()可以判断是否是文件夹，stats.isFile()判断是否是文件;

```
例如：photoName 为uploads下的images文件夹 (photoName = "images")

var path = require("path"); 模块的extname函数能获取到文件格式，返回如“.jpg  .png”

fs.readdir("./uploads/" + photoName,function(err,files){
    if(err){
        //callback("没有找到Uploads文件",null);
        return;
    }

    var allImages = [];
    (function iterator(i){    //循环实现全部子文件的遍历
        if(i == files.length){
            //结束了
            //callback(null,allImages);
```

```
                return;//所有的除了报错可以不写也没关系，其他的时候一定记得加上
            }
            //uploads/当前点击的“相册文件夹”/相册文件
            fs.stat("./uploads/" + photoName + "/" +
files[i],function(err,stats){
                if(err){
                    throw err;
                }
                /*
                    如果需要整个全部显示，那么你判断是否是图片或文件夹，如果二
者满足其一，则push到allImages数组，
                */
                if (stats.isDirectory()) {
                    allImages.push({name: files[i],isDire:true,
path:"./uploads/" + files[i]});
                }else {
                    var extname = path.extname(files[i]).toLowerCase();
                    if (extname == ".jpg" || extname == ".png" || extname
== ".jpeg" || extname == ".gif" || extname == ".bmp") {
                        allImages.push({name: files[i], isDire:false,
path:"uploads/" + files[i]});
                    }
                }
                iterator(i+1);
            });
        })(0);

    });
```

6. 模块的创建，require，exports，module.exports

- 创建foo.js后，定义了name,age等相关属性，外界如何获得并使用foo的信息，通过使用exports 来向外部暴露相关的方法和属性即可;

此为foo.js文件：

```
var name = "chase";
var age = 20;

var obj = {
    name: name,
    age: age
}
```

```
console.log("已经执行");
exports.obj = obj;  // 向外暴露信息

------------------------------------------------------------
```

此为index.js 文件进行调用:

```
var foo = require("./public/foo.js");
console.log(typeof foo);      //object      对象
console.log(foo.obj.name);
```

- 创建一个类，并且引用

此为 people.js文件：

```
function People(name,sex,age) {
    this.name = name;
    this.sex = sex;
    this.age = age;
}

People.prototype = {
    sayHello: function () {
        console.log(this.name+","+this.sex+","+this.age);
    }
}

module.exports = People;  //暴露一个方法（类）

------------------------------------------------------------
```

此为index.js 文件进行调用:

```
var people = require("./public/people");
console.log(typeof people);  //function   方法（类）

var chase = new people("chase","男",20);
console.log(chase.name);
```

7. 日期格式化

- 引用require("silly-datetime") 模块，可以对日期进行格式化处理；

```
/*
    format:两个参数
    第一个：时间对象，就是你需要格式化的字符对象
    第二个：传入你需要指定的时间格式

    year：年
    month:月
    day:天
    hour：小时
    Minute：分钟
    second: 秒

    Y：年   M：月   D：日   H：小时  m：分钟（小写），s: 秒（小写）
*/

var sd = require("silly-datetime");

var newDate = sd.format(new Date(),"YYYY-MM-DD HH:mm");
console.log(newDate);  // 输出： 2017-11-27 15:43

var newDate1 = sd.format(new Date(),"YYYY-MM-DD HH:mm:ss");
console.log(newDate1); // 输出： 2017-11-27 15:43:47
```

8. post 数据请求

- 首先需要创建一个upload.html进行post数据的提交，action设置为http://localhost:3000/dopost，method为post；

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>表单提交  动脑学院作业</title>
    <link rel="stylesheet" href="css/master.css"/>
</head>
<body>
    <div class="wrapper">
        <div class="title">注册中心
            <span class="title-status"><a href="index.html">返回相册</a>
</span>
        </div>
        <form class="form"   action="http://localhost:3000/dopost"
method="post">
            <ul>
                <li><input type="text" name="name" placeholder="请输入您的
名字" class="txt"/></li>
                <li><input type="password" name="password" placeholder="请
输入您的密码" class="txt"/></li>
                <li><input type="submit" placeholder="" class="btn"/></li>
            </ul>
            <div class="cl"></div>
        </form>
    </div>
</body>
</html>
```

- 创建form.js进行，实现服务接收数据

```
var http = require("http");
var querystring = require("querystring");
//将参数 & 字符串转换对象，
//如 “name=hello&password=123456” --> {name:"hello",password:"123456"}

var server = http.createServer(function (req, res) {

    if (req.url == "/dopost" && req.method.toLowerCase() == "post") {

        var allData = "";
        req.addListener("data",function(chunk){
            allData += chunk;  //数据是分段接收的
        });
```

```
        req.addListener("end",function () {
            console.log(typeof allData);   //输出 string
            console.log(allData); //输出 name=hello&password=123456

            var dateString = allData.toString();
            var dataObj = querystring.parse(dateString)
            console.log("name: "+dataObj.name);  // 输出 name: hello

            res.end("success");
        });
    }
});

server.listen(3000,"127.0.0.1");
```

9. 上传文件或图片

- 引用 var formidable = require("formidable");
- 创建 IncomingForm对象；var form = new formidable.IncomingForm();
- 设置 uploadDir属性，即文件接收收保存的目标文件夹, form.uploadDir = "./static/images;
- form.parse 实现接收成功回调；
- 可以使用fs.rename 来进行文件的重命名和移动;

```
var formidable = require("formidable");
var http = require("http");
var util = require("util");
var path = require("path");
var sd = require("silly-datetime");
var fs = require("fs");

http.createServer(function (req, res) {
    if (req.url == "/upload" && req.method.toLowerCase() == "post") {
        var form = new formidable.IncomingForm(); //创建IncomingForm对象
        form.uploadDir = "./static/images";  //保存的路径

       /*
          fields: 提交的基本数据（name： username, age）
          files: 提交的文件数据（name： upload)
       */

        form.parse(req, function (err, fields, files) {
```

```
            res.writeHead(200, {"Content-type":"text/plain"});
            res.write("received upload:");

            var ran = parseInt((Math.random()*10000).toFixed(0));
            var newDate = sd.format(new Date(), "YYYYMMDDHHmmss");
            var extname = path.extname(files.upload.name);
            var oldpath = __dirname + "/" + files.upload.path;
            var newpath = __dirname + "/static/images/" + newDate + ran +
extname;

            console.log("__dirname: " + __dirname); //
/Users/chase/Pictures/MyCodes/codes
            console.log("oldpath: " + oldpath);     //
/Users/chase/Pictures/MyCodes/codes/static/images/upload_ce6324572d05a4270
d3cf20259f45f27


            //不同路径下的文件更名 + 移动（这里文件已经在/static/images下，只
是对其进行重命名）
            fs.rename(oldpath,newpath,function (err) {
                if (err) {
                    throw Error("失败");
                }
                res.writeHead(200, {"Content-type": "text/plain"});
                res.end("success");
            });

            res.end(util.inspect({fields:fields, files: files}));
        });

        return;
    }

    //表示get请求时，获取渲染html文件
    res.writeHead(200, {"Content-type":"text/html"});
    res.end(
        '<form action="/upload" enctype="multipart/form-data"
method="post">'+
        '<input type="text" name="username"><br>'+
        '<input type="text" name="age"><br>'+
        '<input type="file" name="upload" multiple="multiple"><br>'+
        '<input type="submit" value="Upload">'+
        '</form>'
    )

}).listen(3000);
```

10. ejs的使用

- ejs 是能动态渲染到html，使html看起来更生动和实用
- 实用 npm install ejs --save 安装ejs
- 引用 var ejs = require("ejs");
- 创建 index.ejs 文件,用来显示;
- 创建 ejs.js 设置index.ejs数据；

index.ejs文件：

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
          content="width=device-width, user-scalable=no, initial-
scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>


</head>
<body>

    <div> num的值 <%= num %></div>
    <ul>

        <%
            for(var i=0;i<list.length;i++){
                if(list[i].age < 20){

        %>
                <li>
                    <span style="color:red;"><%= list[i].title %></span>
                    <span style="color:green;"><%= list[i].age %></span>
                </li>
        <%
                }
            }
        %>

    </ul>

</body>
</html>
```

ejs.js 文件：

```
var http = require("http");
var fs = require("fs");
var ejs = require("ejs");

http.createServer(function (req, res) {

    fs.readFile("./views/index.ejs", function (err,stats) {

        var template = stats.toString();
        var data = {
            num: 1,
            list:[
                {
                    title:"Hello1",
                    age:18
                },
                {
                    title:"Hello2",
                    age:19
                },
                {
                    title:"Hello3",
                    age:20
                }
            ]
        }

        var html = ejs.render(template,data);

        console.log("stats: " + typeof stats);
        console.log("template: " + typeof template);
        console.log("html: " + typeof html);

        console.log(stats);
        console.log(html);

        res.writeHead(200,{"Content-type":"text/html;charset=UTF-8"});
        res.end(html);

    });

}).listen(3000,"127.0.0.1");
```

- 使用fs.readFile获取文件，将文件转化为字符串var template = stats.toString()模版；
- ejs.render 方法是将数据渲染到模版中，第一个参数为文件模版字符串，第二个参数为数据对象；
- 然后设置响应返回 res.end(html);

express 学习

1. express use 和 get 的区别

- 同一个路由，例如apple，use可以访问apple/news....下面所有的路由，而get不行
- use 可以设置静态资源文件app.use(express.static("./public"));
- use 不考虑http的请求方法；

```
如果path 设置为’ /’ ,则
GET /
PUT /foo
POST /foo/bar
均是中间件的作用范围
```

koa 学习

1. koa 和 express 的区别

```
官网：http://www.koacn.com/
开发团队：Express原班人马
Express弱点：不能忍的callback（函调函数）
优点：可以免除重复繁琐的回调函数嵌套，并极大地提升常用错误处理效率
node版本建议：v7.6.0以上
特点：是为ES6而生的，并非ES5，
```

需要使用npm install koa --save 来安装koa

2. koa 的基础应用

```
var Koa = require("koa");
var app = new Koa();

//context
var home = function (ctx) {
    //ctx.request  表示请求
    //ctx.response 表示http 响应

    //ctx.response.body属性就是发送给用户得内容

    console.log(ctx);
    ctx.response.type = "html";
    ctx.response.body = "hello"
}

//模糊匹配   和 express .use 相似
app.use(home);
app.listen(3000);
```

- 首先引用koa模块require("koa")；
- 创建koa对象；
- 使用use进行回调，与express .use 方法相似，都是模糊匹配；
- 调用listen进行监听和开始服务；
- 回调方法中有两个参数，ctx(上下文)，next(下一个中间件)；
- ctx.response.body 属性就是发送给用户得内容，将会显示在浏览器中；
- ctx.response.type 设置响应的文本类型；

3.koa 读取静态文件

使用 fs.createReadStream 流的方式进行读取

```
var Koa = require("koa");
var app = new Koa();
var fs = require("fs");

var main = function (ctx) {
    //返回文件
    /*
       不能使用fs.readFile
```

```
    */
    ctx.response.type = "html";
    ctx.response.body = fs.createReadStream("./template.html");
}

app.use(main);

app.listen(3000);
```

4.koa route绝对匹配

使用koa-route 模块，可以进行get，post 等方式的路由绝对匹配，如果不实用route模块，直接进行use调用，则为相对匹配。

```
var Koa = require("koa");
var app = new Koa();
var route = require("koa-route");

// 实现 get 路由

var home = function (ctx) {
    ctx.response.type = "html";
    ctx.response.body = "welcome to home";

}

var other = function (ctx) {
    ctx.response.type = "html";
    ctx.response.body = "welcome to other";
}


// 绝对匹配
app.use(route.get("/", home));
app.use(route.get("/other", other));

// 相对匹配
// app.use(path, callback);

app.listen(3000);
```

5. koa 静态资源文件

使用koa-static 模块，把需要的设置的静态资源文件路径作为参数，然后设置use就可以了。

```
var Koa = require("koa");
var app = new Koa();
var koastatic = require("koa-static");
var path = require("path");

//设置静态资源
var main = koastatic(path.join(_dirname));

// 就可以访问这么目录下的所有文件
app.use(main);

app.listen(3000);
```

6. koa 请求重定向

在回调的响应属性redirect，设置重定向的路由；

```
var Koa = require("koa");
var app = new Koa();
var route = require("koa-route");

var login = function (ctx) {
    ctx.response.body = "login page"
}

var user = function (ctx) {
    ctx.response.body = "user page"
    ctx.response.redirect("/login");
    console.log("hello login");
    //重定向到 /login
}

app.use(route.get("/login",login));
app.use(route.get("/user", user));

app.listen(3000);
```

7. koa 中间件

koa 中间件 之间的处理是通过next()来现实的，执行next()时，将会跳转到匹配的下一个中间价，执行结束后，在回到原处，往下执行。例如以下:

```
var Koa = require("koa");
var app = new Koa();

var login = function (ctx, next) {

    /*
      ctx.response.body = "login";
      next();
      浏览器显示  main


      next();
      ctx.response.body = "login";
      浏览器显示  login


      总结： 说明在执行next()时，将会跳转到匹配的下一个中间价，执行结束后，在回到原处，往下执行

     */

    next();
    ctx.response.body = "login";
    //next();
}

var main = function (ctx) {
    ctx.response.body = "main";
}

app.use(login);
app.use(main);

app.listen(3000);
```