express框架

# vscode自己定义的express代码判断：打开vscode 输入etp回车，就会由下面的代码，前提是需要安装express和配置vscode中javascript.json文件

|  |
| --- |
| let express = require("express");  let port = 8000;  let app = express();  //app.use(express.static(\_\_dirname+'/public'))//静态服务器，根据需要配置  app.get("/", (req, res) => {    res.end("<h1>welcome</h1>");  });  app.listen(port, () => {    console.log("server is ready :http://localhost:" + port + "/");  }); |

# 1.初识express框架

|  |
| --- |
|  |

# 2.express初体验

## 1.）新建一个文件夹learnexpress,然后打开终端定位到这个目录，运行npm init -y，创建包

|  |
| --- |
|  |

## 2）安装express

### 在终端中输入命令：npm i express

|  |
| --- |
|  |

## 3）编写第一个express程序

## 新建一个expressdemo1.js文件，内容如下

|  |
| --- |
| let express = require('express')  //创建应用对象  let app = express()  //创建路由  app.get('/home',(req,res)=>{      res.end("<h1>welcome to home page</h1>")  })  //4.监听端口，启动服务  let port =8000  app.listen(port,()=>{      console.log(`server is ready :http://localhost:${port}/`);  }) |

### 效果

|  |
| --- |
|  |

# 3.express路由

|  |
| --- |
|  |

## 路由小案例：02-expressdemo.js

|  |
| --- |
|  |

### 02-expressdemo.js

|  |
| --- |
| let express = require('express')  //创建应用对象  let app = express()  //创建路由,get请求  app.get('/',(req,res)=>{      res.end("<h1>welcome to Index page</h1>")  })  app.get('/home',(req,res)=>{      res.end("<h1>welcome to home page</h1>")  })  //创建路由,post请求  app.post('/login',(req,res)=>{      res.end("<h1>login page</h1>")  })  // 创建路由,all请求，就是只匹配路径，不管什么请求方式都功能访问到  app.all('/test',(req,res)=>{      res.end("<h1>test page</h1>")  })  //响应404 \*表示如果上面的都匹配不上就会走这里  app.all('\*',(req,res)=>{      res.end("<h1>Not Found</h1>")  })  //4.监听端口，启动服务  let port =8000  app.listen(port,()=>{      console.log(`server is ready :http://localhost:${port}/`);  }) |

# 4、express提取请求报文参数

|  |
| --- |
|  |

## 小案例

## 03-express-get-param.js

|  |
| --- |
| let express = require('express')  //创建应用对象  let app = express()  //创建路由,get请求  app.get('/request',(req,res)=>{      //1.使用nodejs原生方式获取      console.log(req.method);      console.log(req.url);      console.log(req.httpVersion);      console.log(req.headers);      //2.使用express的方式获取，也就是这些方式只能在express里面使用      console.log(req.query); //获取查询字符串      //获取指定的请求头信息      console.log(req.get('host'));      res.end('ok')  })    //.监听端口，启动服务  let port =8000  app.listen(port,()=>{      console.log(`server is ready :http://localhost:${port}/`);  }) |
|  |
|  |

## 其实这里我们可以使用all方法，路由传递‘\*’，因为我们这里是演示获取请求头，应该什么路由都可以

|  |
| --- |
| let express = require('express')  //创建应用对象  let app = express()  //创建路由,get请求  // app.get('/request',(req,res)=>{  //     //1.使用nodejs原生方式获取  //     console.log(req.method);  //     console.log(req.url);  //     console.log(req.httpVersion);  //     console.log(req.headers);  //     //2.使用express的方式获取，也就是这些方式只能在express里面使用  //     console.log(req.query); //获取查询字符串  //     //获取指定的请求头信息  //     console.log(req.get('host'));  //     res.end('ok')  // })    app.all('\*',(req,res)=>{      //1.使用nodejs原生方式获取      console.log(req.method);      console.log(req.url);      console.log(req.httpVersion);      console.log(req.headers);      //2.使用express的方式获取，也就是这些方式只能在express里面使用      console.log(req.query); //获取查询字符串      console.log(req.path); //获取请求路径，这么写是express才有      console.log(req.ip);//获取客户端的ip地址      console.log(req.get('host'));//获取指定的请求头信息      res.end('ok')  })    //.监听端口，启动服务  let port =8000  app.listen(port,()=>{      console.log(`server is ready :http://localhost:${port}/`);  }) |

# 5.express获取路由参数

|  |
| --- |
|  |

## 小案例：04-express-get-route-params.js

|  |
| --- |
| let express = require('express')  //创建应用对象  let app = express()  //获取路由参数  app.get('/:id.html',(req,res)=>{ //在路径后面:后面的是路由参数      res.setHeader('Content-type','text/html;charset=utf-8')      res.write(req.params.id) //路由参数响应使用req.params.参数名的方式来获取      res.end()  })        //.监听端口，启动服务  let port =8000  app.listen(port,()=>{      console.log(`server is ready :http://localhost:${port}/`);  }) |
|  |

# 6.路由参数练习

|  |
| --- |
|  |

## 小案例：根据路由响应歌手的信息，项目结构

|  |
| --- |
|  |

## singer.json

|  |
| --- |
| {      "singers":[          {              "singer\_name":"周杰伦",              "singer\_pic" :"https://th.bing.com/th/id/OIP.ncZ7EbJ0O6geJUNv2UnCNgHaJ4?w=133&h=180&c=7&r=0&o=5&pid=1.7",              "other\_name" :"Jay Chou",              "singer\_id":4558,              "id":1            },          {              "singer\_name":"林俊杰",              "singer\_pic" :"https://th.bing.com/th/id/OIP.iL1b5tQgcpqMJSLOoIjbugHaJQ?w=153&h=192&c=7&r=0&o=5&pid=1.7",              "other\_name" :"JJ Lin",              "singer\_id":4286,              "id":2            },          {              "singer\_name":"邓紫棋",              "singer\_pic" :"https://th.bing.com/th/id/OIP.8KgoeStvVSsE7KolSwapiAHaHa?w=176&h=180&c=7&r=0&o=5&pid=1.7",              "other\_name" :"Gloria Tang",              "singer\_id":13948,              "id":3            }      ]  } |

## 06-express-exercise-get-singer.js

|  |
| --- |
| let express = require('express')  //使用require函数导入歌手信息的json文件，导入后自动成为一个对象，  // 不过这里最好使用解构赋值，直接获取里面的singer数组  let {singers} = require('./singer.json')  // console.log(singers);  //创建应用对象  let app = express()  //获取路由参数  app.get('/singer/:id.html',(req,res)=>{ //在路径后面:后面的是路由参数      res.setHeader('Content-type','text/html;charset=utf-8')      let id = req.params.id //这里获取到的是字符串，需要转化为数字，否则下面数组中找不到      // 在数组中使用find方法来查找歌手      let singer = singers.find(item=>item.id === Number(id))      //如果没有找到，需要返回Not Found      if(!singer){          res.statusCode = 404          res.end("<h1>Not Found</h1>")   return      }      // console.log(singer);      res.write(`<!DOCTYPE html>          <html lang="en">          <head>              <meta charset="UTF-8">              <meta name="viewport" content="width=device-width, initial-scale=1.0">              <title>Document</title>          </head>          <body>          <h1>${singer.singer\_name}</h1><br>          <img src=${singer.singer\_pic}>          </body>          </html>`) //路由参数响应使用req.params.参数名的方式来获取      // res.write(singer.singer\_pic) //路由参数响应使用req.params.参数名的方式来获取      res.end()  })      //.监听端口，启动服务  let port =8000  app.listen(port,()=>{      console.log(`server is ready :http://localhost:${port}/`);  }) |

## 效果：

|  |
| --- |
|  |
|  |
|  |
|  |

# 7.express响应设置

|  |
| --- |
|  |

## 小案例：07-express-set-response-params.js

|  |
| --- |
|  |
| let express = require('express')  //创建应用对象  let app = express()  //获取路由参数  app.get('/response',(req,res)=>{      //1.兼容原生nodejs      // res.statusCode=200      // res.statusMessage="OK"      // res.setHeader('author','kenny')      //2.express的写法      // res.status(200)      // res.set("content-type","text/javascript")      // res.send('<a href="https://www.google.com">google</a>')      //3.也可以链式操作      // res.status(200).set("Content-type","text/html;charset=utf-8").send('<a href="https://www.google.com">google</a>')      //4.其他响应      // res.end()//下面的方法不要写，否则没有效果      // res.redirect("http://atguigu.com")      //res.download(\_\_dirname+'/package.json') //浏览器会下载这个文件     res.json({          name:"kenny",          age:50      })      // res.sendFile(\_\_dirname+"/index.html")      })      //.监听端口，启动服务  let port =8000  app.listen(port,()=>{      console.log(`server is ready :http://localhost:${port}/`);  }) |
|  |

# 8.中间件

## 1）中间件介绍

|  |
| --- |
|  |

## 2）全局中间件：发送有请求过来，全局中间件都会执行

## 小案例：记录每一个请求的url和ip地址

## 使用全局中间件的步骤：

### 1，需要创建一个全局中间件函数

|  |
| --- |
| let express = require('express')  let fs = require('fs')  let path =require('path')  //创建应用对象  let app = express()  //1.定义中间件函数  function logMiddleWare(req,res,next) {      //获取url和ip      let {url,ip} = req      //保存信息到access.log文件      fs.appendFileSync(path.resolve(\_\_dirname,"./access.log"),`${url} -- ${ip} time:${new Date().toLocaleString()}\r\n`)      //调用next函数      next()  } |

### 2.需要在app对象中注册(也叫使用)全局中间件函数

|  |
| --- |
| let express = require('express')  let fs = require('fs')  let path =require('path')  //创建应用对象  let app = express()  //1.定义中间件函数  function logMiddleWare(req,res,next) {      //获取url和ip      let {url,ip} = req      //保存信息到access.log文件      fs.appendFileSync(path.resolve(\_\_dirname,"./access.log"),`${url} -- ${ip} time:${new Date().toLocaleString()}\r\n`)      //调用next函数      next()  }  //2.使用中间件函数  app.use(logMiddleWare)  //获取路由参数  app.get('/home',(req,res)=>{  //    logMiddleWare(req,res,next)     res.send("前台首页")  })  app.get('/admin',(req,res)=>{     res.send("后台首页")  })  app.all('\*',(req,res)=>{     res.send("<h1>Not Found</h1>")  })      //.监听端口，启动服务  let port =8000  app.listen(port,()=>{      console.log(`server is ready :http://127.0.0.1:${port}/`);  }) |

### 效果：由于这是一个全局中间件，无论你输入什么路径都会有日志记录

|  |
| --- |
|  |

## 3）路由中间件：只有和中间件指定的路由匹配时，路由中间件才会执行

## 小案例：9-express-route-middleware.js

|  |
| --- |
|  |

## 使用路由中间件的步骤

### 1）定义中间件函数

|  |
| --- |
| let express = require('express')  //创建应用对象  let app = express()  //获取路由参数  //前台  app.get('/home',(req,res)=>{     res.send("前台首页")  })  //定义路由中间件函数  let checkCodeMiddleWare = (req,res,next)=>{     //判断url是否有code并且等于521   if(req.query.code === '521'){        next()     } else {        res.send('暗号错误')     }  } |

#### 其实定义和写法大同小异，只是使用的地方不一样，就不能用app对象来注册了

## 2）在需要使用的地方调用，app.get方法其实可以有3个参数，第二个参数就是路由中间件，如果没有，可以不传

|  |
| --- |
| let express = require('express')  //创建应用对象  let app = express()  //获取路由参数  //前台  app.get('/home',(req,res)=>{     res.send("前台首页")  })  //定义路由中间件函数  let checkCodeMiddleWare = (req,res,next)=>{ //使用路由中间件     //判断url是否有code并且等于521   if(req.query.code === '521'){        next()     } else {        res.send('暗号错误')     }  }  //后台  app.get('/admin',checkCodeMiddleWare,(req,res)=>{ //路由中间件只在有需要的路由里面使用       res.send("后台首页")  })  //后台设置  app.get('/setting',checkCodeMiddleWare,(req,res)=>{     res.send("设置页面")  })  app.all('\*',(req,res)=>{     res.send("<h1>Not Found</h1>")  })      //.监听端口，启动服务  let port =8000  app.listen(port,()=>{      console.log(`server is ready :http://127.0.0.1:${port}/`);  }) |
|  |
|  |
|  |
|  |

# 9.express静态资源中间件

|  |
| --- |
|  |

## 小案例，为了方便演示，我们先创建一个10-express-static-middleware文件夹，在文件夹里面创建一个public文件夹，里面新建一个index.html，还可以有其他资源

|  |
| --- |
|  |

## 10-static-middleware.js

|  |
| --- |
| let express = require('express')  let app = express()  let port = 8000  app.use(express.static('public'))  app.get("/",(req,res)=>{      res.send('<h1>index page</h1>')  })  app.listen(port,()=>{      console.log(`Server: http://127.0.0.1:${port}/`);  }) |

### 效果

|  |
| --- |
|  |
|  |

### 注意：这样子一个简单的静态服务器就做好了，此时如果你添加一个非静态路由，它也能够正常工作

|  |
| --- |
| let express = require('express')  let app = express()  let port = 8000  app.use(express.static(\_\_dirname+'/public'))  app.get("/home",(req,res)=>{      res.send('<h1>welcome home</h1>')  })  app.listen(port,()=>{      console.log(`Server: http://127.0.0.1:${port}/`);  }) |
|  |

# 10.静态资源中间件注意事项

|  |
| --- |
|  |

# 11.express静态资源中间件练习（略）

## 需求

|  |
| --- |
|  |

## 做法和上面的案例一样，这里省略

# 12.express获取请求体的数据

|  |
| --- |
|  |

## 课堂小案例：

## 1）新建一个文件夹：12-express-get-rquest-body在里面新建一个public文件夹里面新建一个login.html,然后在public外面新建一个12-get-req-body.js

|  |
| --- |
|  |

### 注意这里我们学习express，使用的依赖包都安装在learnexpress文件夹里面，这样子，所有的子文件夹都可以使用依赖

## 2）先安装body-parser : npm i body-parser

|  |
| --- |
|  |

## 3)使用body-parser来获取数据，在public里面添加两个文件index.html和reg.html

|  |
| --- |
|  |

### index.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>Index Page</title>  </head>  <body>      <h1>首页</h1>         <p> <a href="/login.html">登录</a></p>      <p><a href="/reg.html">注册</a></p>    </body>  </html> |

### reg.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>用户注册</title>  </head>  <body>      <form action="#" method="POST">          user: <input type="text" name="username"><br>          pwd: <input type="text" name="password"><br>          <input type="submit" value="注册">      </form>    </body>  </html> |

### 12-get-req-body.js

|  |
| --- |
| let express = require('express')  let bodyParser = require('body-parser')  let port = 8000  let app = express()  //设置静态资源服务  app.use(express.static(\_\_dirname+'/public'))  //获取urlParser  let urlParser = bodyParser.urlencoded({extended:false})  //处理json格式的请求体，这里用不到  // let jsonParser = bodyParser.json()  //配置login路由,urlParser是作为路由中间件来使用的  app.post('/login',urlParser,(req,res)=>{      console.log(`         表单数据：         用户名：${req.body.username}         密码：${req.body.password}      `);      res.send('ok')  })  app.listen(port,()=>{      console.log(`Ready : http://127.0.0.1:${port}/`);  }) |

### 效果：

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

### 注意：body-parser既可以作为全局中间件来使用，也可以作为路由中间件来使用，我们更推荐作为路由中间件来使用，当body-parser中间件执行完成后，它会给req请求对象添加一个body属性，里面由表单数据，我们需要从req身上获取数据

# 13.express-防盗链

## 防盗链可以防止用户盗窃网站资源

## 防盗链的原理就是利用请求头里面的referer属性来区分是否是自己的网站，如果不上，会响应404，那么他们就拿不到数据

## 我们可以自己写一个小案例来实现防盗链功能

### 1.我们新建一个文件夹：13-anti-steal,然后把10-文件夹里面的public拷贝过来，新建一个13-anti-steal.js结构如下

|  |
| --- |
|  |

### 2.使用上面的代码判断创建一个最基本的express服务，然后改动一点点，代码如下

|  |
| --- |
| let express = require("express");  let port = 8000;  let app = express();  app.use(express.static(\_\_dirname+'/public'))//静态服务器，根据需要配置  // app.get("/", (req, res) => {  //   res.end("<h1>welcome</h1>");  // });  app.listen(port, () => {    console.log("server is ready :http://localhost:" + port + "/");  }); |

### 3.先测试一下，发现正常运行

|  |
| --- |
|  |

#### 此时无论我们使用<http://127.0.0.1:8000/images/dogcr.gif>还是<http://localhost:8000/images/dogcr.gif>都能够访问到资源，我们下面可以做一个限定，规定只有<http://127.0.0.1:8000/>形式的主机才能访问我们的页面，<http://localhost:8000/>形式的不能访问我们的页面，我们写一个中间件来完成这个功能

|  |
| --- |
| let express = require("express");  let port = 8000;  let app = express();  //限制特定网址的中间件,需要在静态服务中间件之前调用否则没有任何意义  app.use((req,res,next)=>{      //检测请求头的referer属性值是否为127.0.0.1      //1.获取url完整路径      let referer = req.get('referer')      //2.利用URL实例来获得主机名      if(referer){          let hostname = new URL(referer).hostname          if(hostname !=='127.0.0.1'){             //响应404             res.status(404).send("<h1>Not Found</h1>")             return          }      }      next()      // console.log(referer);     })  app.use(express.static(\_\_dirname+'/public'))//静态服务器，根据需要配置  app.listen(port, () => {    console.log("server is ready :http://localhost:" + port + "/");  }); |

### 效果：此时如果用<http://localhost:8000/index.html> 是没有图片的。

|  |
| --- |
|  |

### 用<http://127.0.0.1:8000/index.html>是可以正常显示图片的

|  |
| --- |
|  |

## express有一个叫做express-anti-leech的防盗链插件，可以参考下面文档

|  |
| --- |
| 安装 npm install express-anti-leech 使用 var express = require('express'),  path = require('path'),  app = express();    var AntiLeech = require('express-anti-leech');  // 允许引用的域名白名单  var hosts = ['localhost', 'localhost:8004'];  // 反盗链类型  var exts = ['.png', '.jpg', '.jpeg', '.gif', '.swf', '.flv'];  // 盗链默认指向图片  var pictrue = "/images/default.png";  app.use(AntiLeech({  allow: hosts,  exts: exts,  log: console.log, // 你也可以使用自己的方法来记录  default: pictrue  }));  // 请在调用静态资源之前先使用反盗链模块  app.use(express.static(path.join(\_\_dirname, 'public')));  app.set('port', process.env.PORT || 8004);  app.get('/', function(req, res) {  res.redirect("/index.html");  });  app.listen(app.get('port'), function() {  console.log("Express test server listening on http://localhost:" + app.get('port'));  }); |

# 14，路由模块化

## 我们以前写的代码都是在一个文件里面处理路由的，如果以后项目比较复杂这样子比较难维护，我们可以使用路由模块化的方法来解决这个问题

## 案例：

### 1.新建一个14-route-modulize文件夹，然后在里面新建一个routes文件夹在里面新建一个homeRouter.js文件，内容如下

|  |
| --- |
| let express = require('express')  //创建路由对象  let router = express.Router()  //创建路由规则  //前台  router.get('/home',(req,res)=>{      res.send("前台首页")   })   //搜索   router.get('/search',(req,res)=>{      res.send("内容搜索")   })   module.exports = router |

### 2.在14-route-modulize.js中导入这个router，然后需要调用app.use()方法把这个router作为参数传递进来

|  |
| --- |
| let express = require('express')  let homeRoter = require('./routes/homeRouter') //导入自己写的路由文件  //创建应用对象  let app = express()  //获取路由参数  app.use(homeRoter) //使用路由文件  //后台  app.get('/admin',(req,res)=>{       res.send("后台首页")  })  //后台设置  app.get('/setting',(req,res)=>{     res.send("设置页面")  })  app.all('\*',(req,res)=>{     res.send("<h1>Not Found</h1>")  })      //.监听端口，启动服务  let port =8000  app.listen(port,()=>{      console.log(`server is ready :http://127.0.0.1:${port}/`);  }) |

### 测试一下，效果如下：

|  |
| --- |
|  |
|  |

### 3我们可以把14-route-modulize.js里面剩余的路由也抽取出来，放到一个./routes/adminRouter.js文件中

|  |
| --- |
| let express = require('express')  let router = express.Router()  //后台  router.get('/admin',(req,res)=>{        res.send("后台首页")   })     //后台设置   router.get('/setting',(req,res)=>{      res.send("设置页面")   })   //导出roter   module.exports = router |

### 4. 在14-route-modulize.js中导入这个router，然后需要调用app.use()方法把这个router作为参数传递进来

|  |
| --- |
| let express = require('express')  let homeRouter = require('./routes/homeRouter') //导入自己写的路由文件  let adminRouter = require('./routes/adminRouter')  //创建应用对象  let app = express()  //获取路由参数  app.use(homeRouter) //使用路由文件  app.use(adminRouter) //使用路由文件  app.all('\*',(req,res)=>{     res.send("<h1>Not Found</h1>")  })      //.监听端口，启动服务  let port =8000  app.listen(port,()=>{      console.log(`server is ready :http://127.0.0.1:${port}/`);  }) |

### 也是好使的

|  |
| --- |
|  |
|  |

# 15.express模板引擎-ejs

## 1.ejs模板引擎

|  |
| --- |
|  |

### 需要先把ejs安装好

|  |
| --- |
|  |

## 2.ejs基本使用

### 01-ejs-first-app.js

|  |
| --- |
| let ejs = require('ejs')  //字符串拼接  let china = '中国'  //使用ejs渲染  let result = ejs.render('我爱你 <%= data%>',{data:china})  console.log(result); |
|  |

### 我们把代码修改一下

|  |
| --- |
| let ejs = require('ejs')  //字符串拼接  let china = '中国'  let str='我爱你 <%= data%>'  //使用ejs渲染  let result = ejs.render(str,{data:china})  console.log(result); |

### 效果是一样的

|  |
| --- |
|  |

### 然后我们在相同目录下面创建一个01-html.html文件，把str粘贴到里面

|  |
| --- |
|  |

### 然后用fs把这个html文件读取进来

|  |
| --- |
| let ejs = require('ejs')  let fs = require('fs')  //字符串拼接  let china = '中国'  let str =fs.readFileSync('./01-html.html').toString()  //使用ejs渲染  let result = ejs.render(str,{data:china})  console.log(result); |

### 然后运行，效果是一样的

|  |
| --- |
|  |

### 然后我们把01-html.html的内容修改一下

|  |
| --- |
|  |

### 效果如下

|  |
| --- |
|  |

### 然后我们把01-html.html改为一个完整的html文件

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>Document</title>  </head>  <body>      <h2>我爱你 <%= data %></h2>  </body>  </html> |

### 运行程序，效果如下

|  |
| --- |
|  |

### 我们可以在js文件里面添加一个新的变量。然后在ejs的render函数的对象里面设置

### 01-ejs-first-app.js

|  |
| --- |
| let ejs = require('ejs')  let fs = require('fs')  //字符串拼接  let china = '中国'  //新内容  let comment = "今天天气很好，我们出去郊游吧~~"  let str =fs.readFileSync('./01-html.html').toString()  //使用ejs渲染  let result = ejs.render(str,{data:china,comment})  console.log(result); |

### 然后中html文件中指定渲染的位置

|  |
| --- |
|  |

### 运行，效果如下

|  |
| --- |
|  |

## 3.ejs列表渲染

### 我们把15-express-template-engine改名为15-express-template-engine-ejs，然后新建一个01-ejs-basic文件夹，把上节课的代码移动到这个文件夹里面，然后新建一个02-ejs-list-rendering文件夹，在文件夹里面新建一个02-render-list.js文件和02-html.html文件

|  |
| --- |
|  |

### 02-html.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>Document</title>  </head>  <body>      <ul>        <% for(let i of xiyou){ %>          <li><%= i %></li>        <% } %>      </ul>    </body>  </html> |

### 02-render-list.js

|  |
| --- |
| let ejs = require('ejs')  let fs = require('fs')  let xiyou = ['唐僧','孙悟空','猪八戒','沙僧']  let str = fs.readFileSync('./02-html.html').toString()//读取的内容在Buffer对象中需要转化为字符串  let result = ejs.render(str,{xiyou:xiyou})  console.log(result); |

### 效果

|  |
| --- |
|  |

## 老师的写法，用数组的forEach方法

|  |
| --- |
|  |

## 4.ejs条件渲染

## 小案例

|  |
| --- |
|  |

### 新建一个03-condition-rendering文件夹在里面新建一个03-html.html和一个03-condition-rendering.js文件

### 3-condition-rendering.js

|  |
| --- |
| let ejs = require('ejs')  let fs = require('fs')  /\*   练习：利用变量isLogin来显示内容  true 输出<span>欢迎回来</span>  false 输出 <button>登录</button><button>注册</button>  \*/  let isLogin = true  //获取模板文件的内容  let html = fs.readFileSync('./03-html.html').toString()  let result = ejs.render(html,{isLogin:isLogin})   console.log(result); |

### 03-html.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>Document</title>  </head>  <body>      <% if(isLogin){ %>          <span>欢迎回来</span>       <% }else{ %>          <button>登录</button><button>注册</button>       <% } %>  </body>  </html> |

### 效果

|  |
| --- |
|  |

## 5.express中使用ejs

## 我的小案例

### 1.在learnexpress文件夹下面新建一个16-use-ejs-in-express文件夹，在里面新建一个js文件和一个public目录，里面有一个index.html文件应该cjs目录里面有应该test.html,然后还有images目录和css目录和js目录ejs/test.html文件里面有ejs模板语法

#### 16-express-ejs-demo.js

|  |
| --- |
| let express = require('express')  let ejs = require('ejs')  let fs = require('fs')  let port = 8000  let app = express()  app.use(express.static(\_\_dirname+'/public'))//静态服务器，根据需要配置  //读取test.html的内容  let html = fs.readFileSync(\_\_dirname+'/public/ejs/test.html').toString()  let result = ejs.render(html,{data:["cpp",'python','golang','js']})  app.get('/test',(req,res)=>{       res.send(result)  })  app.listen(port,()=>{       console.log('server is ready :http://localhost:'+port+'/');  }) |

#### ./public/index.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>Document</title>  </head>  <body>      <a href="/test">测试ejs</a>  </body>  </html> |

#### ./public/ejs/test.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>Document</title>  </head>  <body>      <ul>          <% data.forEach(item=>{%>              <li><%= item  %></li>          <% })%>      </ul>  </body>  </html> |

## 2.效果

|  |
| --- |
|  |
|  |

## 课堂讲解

## 1.新建应该文件夹：17-use-ejs-with-express-teacher，里面有一个17-teacher-ejs-server.js文件和一个public目录和一个views目录（前提是已经安装了express和ejs）

|  |
| --- |
|  |

## 2.先在17-teacher-ejs-server.js中添加代码,注意：在views文件夹里面的模板文件的扩展名必须是ejs，否则找不到文件

|  |
| --- |
| let express = require('express')  let path = require('path')  let port = 8000  let app = express()  //1.设置模板引擎  app.set("view engine",'ejs')  //2.设置模板文件的存放位置  app.set("views",path.resolve(\_\_dirname,'./views'))  app.use(express.static(\_\_dirname+'/public'))//静态服务器，根据需要配置  app.get('/home',(req,res)=>{      let title = '尚硅谷 - 让天下没有难学的技术'       //3.调用render函数       res.render('home',{title})   //需要在views文件夹里面创建一个home.ejs,后缀名一定是ejs，否则找不到文件  })  app.listen(port,()=>{       console.log('server is ready :http://localhost:'+port+'/');  }) |

## 3.需要在views文件夹里面新建一个home.ejs

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>Home</title>  </head>  <body>      <h2><%= title></h2>  </body>  </html> |

### 效果：

|  |
| --- |
|  |

## 以后都使用老师的方式

# 16.express-generator工具

## 1.介绍

|  |
| --- |
|  |

## 2.安装，这里我们使用全局安装：npm i -g express-generator

|  |
| --- |
|  |

## 3.测试是否安装成功：打开一个终端，输入express -h，有输出如图，说明安装成功

|  |
| --- |
|  |

## 4.创建项目：express -e 项目名称，注意-e表示添加ejs模板引擎支持，如：express -e 18a-express-generator-demo

|  |
| --- |
|  |

### 出现上面的信息，上面项目创建成功

## 5.它默认帮我们引入一些依赖声明，我们需要注入项目目录：cd 18a-express-generator-demo，然后运行npm install

|  |
| --- |
|  |

### 依赖安装完成

|  |
| --- |
|  |

## 6.运行项目：npm start，它会启动bin目录下面的www文件，有一点需要注意，如果你的public里面没有index.html文件，它会运行views目录下面的index.ejs文件，如果你的public目录下面有index.html文件，会运行public目录下面的index.html文件，我这里添加了index.html,效果如下：

|  |
| --- |
|  |

### 注意：为了方便调试，如果你全局安装了nodemon，你可以把start命令改为：nodemon ./bin/www

## 使用express-generator可以快速生成有网站结构，然后我们根据需要来修改内容

# 文件上传

# 17、查看文件上传报文

## 我们来学习文件上传功能，还是使用18a项目，给他增加一个文件上传功能，我们先把public目录下面的index.html改名为index2.html，这样子它就会执行views/index.ejs

|  |
| --- |
|  |

## 我们给routes/index.js文件添加两个路由

|  |
| --- |
| var express = require('express');  var router = express.Router();  /\* GET home page. \*/  router.get('/', function(req, res, next) {    res.render('index', { title: 'Express' });  });  //显示上传表单  router.get('/portrait',(req,res)=>{    res.render('portrait')  })  //处理文件上传  router.post('/portrait',(req,res)=>{    res.send('文件上传成功...')  })  module.exports = router; |

## 测试

|  |
| --- |
|  |

## 我们编辑一下views/portrait.ejs.给他添加一个表单

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>文件上传</title>  </head>  <body>      <h2>文件上传</h2>      <hr>      <form action="/portrait" method="post" enctype="multipart/form-data"> <!--文件上传需要指定类型-->        用户名：<input type="text" name="username"/><br/>          头像：<input type="file" name="portrait"/><br/>          <hr>          <button>提交</button>      </form>  </body>  </html> |

## 测试一下

|  |
| --- |
|  |
|  |

## 说明路由没有问题是可以跳转过去的，我们看看报文是什么样子的

|  |
| --- |
|  |
|  |

# 18.处理文件上传

## 上面我们是可以上传文件了，但是我们还没有处理使用没有拿到数据，现在我们来处理文件上传

## 这里我们需要用的一个库：formidable

|  |
| --- |
|  |

## 安装：npm i formidable，完成后如下图：

|  |
| --- |
|  |

## 然后在index.js里面使用，注意在导入的时候需要使用解构表达式，如果不是会报错，这个和官方例子不一样，注意一下，这里其实是有两种写法的

## 写法1

|  |
| --- |
| var express = require('express');  var router = express.Router();  //导入文件上传处理库  let {formidable} = require('formidable')  /\* GET home page. \*/  router.get('/', function(req, res, next) {    res.render('index', { title: 'Express' });  });  //显示上传表单  router.get('/portrait',(req,res)=>{    res.render('portrait')  })  //处理文件上传  router.post('/portrait',(req,res,next)=>{     const form = formidable({multiples:true});    form.parse(req, (err, fields, files) => {      if (err) {        next(err);        return;      }      // res.json({ fields, files });      console.log(fields);      console.log(files);      res.send('文件上传成功...')    });    })  module.exports = router; |

## 写法2

|  |
| --- |
| var express = require('express');  var router = express.Router();  //导入文件上传处理库  let formidable = require('formidable')  /\* GET home page. \*/  router.get('/', function(req, res, next) {    res.render('index', { title: 'Express' });  });  //显示上传表单  router.get('/portrait',(req,res)=>{    res.render('portrait')  })  //处理文件上传  router.post('/portrait',(req,res,next)=>{    const form = new formidable.IncomingForm()    form.parse(req, (err, fields, files) => {      if (err) {        next(err);        return;      }      // res.json({ fields, files });      console.log(fields);      console.log(files);      res.send('文件上传成功...')    });    })  module.exports = router; |

### 注意：fields字段只能保存：文本框，单选按钮，复选框，下拉列表等等的值不能保存文件上传控件的内容。formidable可以完成一些body-parser所能做的事情

## 数据是接收到了。现在我们需要把文件保存起来，需要设置一些选项，我们修改一下代码，2种写法

## 写法1

|  |
| --- |
| var express = require('express');  var router = express.Router();  //导入文件上传处理库  let {formidable} = require('formidable')  /\* GET home page. \*/  router.get('/', function(req, res, next) {    res.render('index', { title: 'Express' });  });  //显示上传表单  router.get('/portrait',(req,res)=>{    res.render('portrait')  })  //处理文件上传  router.post('/portrait',(req,res,next)=>{     const form = formidable({      multiples:true,      //设置上传文件的保存目录      uploadDir:\_\_dirname + '/../public/images',      //保留文件后缀      keepExtensions:true    });    form.parse(req, (err, fields, files) => {      if (err) {        next(err);        return;      }      // res.json({ fields, files });      console.log(fields);      console.log(files);      res.send('文件上传成功...')    });    })  module.exports = router; |

## 写法2

|  |
| --- |
| var express = require('express');  var router = express.Router();  //导入文件上传处理库  let formidable = require('formidable')  /\* GET home page. \*/  router.get('/', function(req, res, next) {    res.render('index', { title: 'Express' });  });  //显示上传表单  router.get('/portrait',(req,res)=>{    res.render('portrait')  })  //处理文件上传  router.post('/portrait',(req,res,next)=>{       const form = new formidable.IncomingForm({      multiples:true,      //设置上传文件的保存目录      uploadDir:\_\_dirname + '/../public/images',      //保留文件后缀      keepExtensions:true    })    form.parse(req, (err, fields, files) => {      if (err) {        next(err);        return;      }      // res.json({ fields, files });      console.log(fields);      console.log(files);      res.send('文件上传成功...')    });    })  module.exports = router; |

### 效果：是可以上传的

|  |
| --- |
|  |
|  |

## 还有一步，需要把图片的路径保存到一个变量，方便用户访问，注意：有时候我们的代码和老师的不一样，可能是现在用的是较高版本的缘故，需要小心

|  |
| --- |
| var express = require('express');  var router = express.Router();  //导入文件上传处理库  // let {formidable} = require('formidable')  let formidable = require('formidable')  /\* GET home page. \*/  router.get('/', function(req, res, next) {    res.render('index', { title: 'Express' });  });  //显示上传表单  router.get('/portrait',(req,res)=>{    res.render('portrait')  })  //处理文件上传  router.post('/portrait',(req,res,next)=>{    //  const form = formidable({    //   multiples:true,    //   //设置上传文件的保存目录    //   uploadDir:\_\_dirname + '/../public/images',    //   //保留文件后缀    //   keepExtensions:true    // });     const form = new formidable.IncomingForm({      multiples:true,      //设置上传文件的保存目录      uploadDir:\_\_dirname + '/../public/images',      //保留文件后缀      keepExtensions:true    })    form.parse(req, (err, fields, files) => {      if (err) {        next(err);        return;      }      // res.json({ fields, files });      // console.log(fields);      console.log(files);      //保存图片路径      //http://localhost:3000/images/4a62e1fd78ddf822235643400.jpg      // console.log("portrait:",files.portrait);      let imgUrl = '/images/'+files.portrait[0].newFilename      // console.log(imgUrl);      res.send(imgUrl) //这里直接返回给用户，等我们学习了数据库，我们需要把它保存到数据库    });    })  module.exports = router; |

### 上传成功后把url返回给用户，以后我们学习了数据库，需要把它保存到数据库

|  |
| --- |
|  |