http模块学习

# 1.创建http服务端

|  |
| --- |
| let http = require('http')  let port = 5000  let server = http.createServer((request,response)=>{      response.writeHead(200,{          "content-type":"text/html;charset=utf-8",          "access-control-allow-origin":'\*'      })      // console.log(request.url);      switch (request.url) {          case '/':              response.write('root')              break;          case "/index.html":             response.write("<h1>欢迎来到首页</h1>")             break;          case "/facebook":              response.write("<a href='https://www.facebook.com'>facebook</a>")             break;          case "/google":              response.write("<a href='https://www.google.com'>google</a>")              break;          default:              response.write("not found")              break;      }      response.end()  })  server.listen(port,()=>{      console.log(`visit : http://localhost:${port}/`);  }) |

## 效果

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

# 2.http模块http服务的注意事项

|  |
| --- |
|  |

# 3.查看浏览器报文

|  |
| --- |
|  |

# 4.从request对象中获取请求报文

## 需要使用url模块来解析request对象的url属性

|  |
| --- |
| let http = require('http')  let url = require('url')  let port = 5000  let server = http.createServer((request,response)=>{      response.writeHead(200,{          "content-type":"text/html;charset=utf-8",          "access-control-allow-origin":'\*'      })      // console.log(request.url);      let urlObj = url.parse(request.url)      console.log(urlObj);      // console.log(urlObj.path);      // console.log(urlObj.pathname);      // console.log(urlObj.href);      switch (request.url) {          case '/':              response.write('root')              break;          case "/index.html":             response.write("<h1>欢迎来到首页</h1>")             break;          case "/facebook":              response.write("<a href='https://www.facebook.com'>facebook</a>")             break;          case "/google":              response.write("<a href='https://www.google.com'>google</a>")              break;          case "/json":              response.write('{"name":"Jackline"}')              break;          default:              response.write("not found")              break;      }      response.end()  })  server.listen(port,()=>{      console.log(`visit : http://localhost:${port}/`);  }) |
|  |
|  |
|  |

# 5.从request中提取请求体

|  |
| --- |
|  |

## 为什么后台输出一片空白？

## 因为我们此时发送的是一个get请求。get请求的请求体一般是空的。解决办法就是往这个服务器发送一个post请求

# 6.从request中提取路径和查询字符串

|  |
| --- |
|  |

# 7. 新方式从request中提取路径和查询字符串

|  |
| --- |
|  |
| let http = require('http')  let port = 3500  let site = `http://localhost:${port}/`  // 以/search?kw=webgl&id=5为例  http.createServer((req,res)=>{     let url = new URL(req.url,site) //'http://localhost:3500/search?kw=webgl&id=5'     console.log(url);     res.write("ok")     res.end()  }).listen(port,()=>{      console.log(`vist:${site}`);  }) |
|  |

## 这个URL对象可以把querystring的解析为一个对象，非常好用

## 注意，request对象获取到的路径是没有协议主机名和端口号的，如果在构建URL对象的时候不传递第二个参数程序就会崩溃，第二个参数就是服务器的协议主机名+端口号

## 获取searchParams的值不能直接.而是要使用get方法

|  |
| --- |
|  |

# 8.http请求练习

|  |
| --- |
|  |

## 我的版本

|  |
| --- |
|  |

## ./pages/login.html

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

## ./pages/reg.html

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

## myserver.js

|  |
| --- |
| let http = require('http')  let fs = require('fs')  let port = 3600  http.createServer((req,res)=>{    switch (req.url) {      case '/login':          res.end(fs.readFileSync("./pages/login.html"))          break;      case '/reg':          res.end(fs.readFileSync("./pages/reg.html"))          break;        default:          res.end("<h1>Welcome to index</h1>")          break;    }  }).listen(port,()=>{      console.log(`Server is ready : http://localhost:${port}/`);  }) |

## 可以添加一个index.html页面然后利用超链接跳转

|  |
| --- |
|  |

## ./pages/index.html

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

## 修改myServer.js文件

|  |
| --- |
| let http = require('http')  let fs = require('fs')  let port = 3600  http.createServer((req,res)=>{    switch (req.url) {      case '/':          res.end(fs.readFileSync("./pages/index.html"))          break;      case '/login':          res.end(fs.readFileSync("./pages/login.html"))          break;      case '/reg':          res.end(fs.readFileSync("./pages/reg.html"))          break;        default:          res.end("<h1>Welcome to index</h1>")          break;    }  }).listen(port,()=>{      console.log(`Server is ready : http://localhost:${port}/`);  }) |

## 老师版本。只是显示文本并没有加载对应的页面

|  |
| --- |
|  |

## 上面的代码会有问题，不能连续调用2次response.end()方法，除非你是一个分支语句否则报错：在end之后还写入！修改如下

|  |
| --- |
| server.listen()... |

## 还有问题，需要设置编码

|  |
| --- |
|  |

## 还有问题，就是当上面两个条件都不满足，服务器不会结束，解决办法如下

|  |
| --- |
|  |

# 9.设置响应报文

|  |
| --- |
|  |
|  |

## 注意：如果不是在条件分支语句里面的话，在整个回调函数里面只能由一次end方法的调用

# 10，响应练习

|  |
| --- |
|  |

## 分析：要实现这个功能，响应使用css样式表和js功能，我们可以新建一个html文件，写好所有代码，然后在js服务器文件在把这个文件读取进来然后渲染到页面上

## 我们需要一个html文件和一个js文件，如图

|  |
| --- |
|  |

## table.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>表格</title>      <style>          table{              border: 1px;              width: 60%;              text-align: center;          }           tr:nth-child(odd){             background-color: deeppink;          }           tr:nth-child(even){              background-color:  steelblue;          }          /\* td:hover{              color: orangered;            } \*/      </style>  </head>  <body>      <table>          <tr>              <td>row1col1</td>              <td>row1col2</td>              <td>row1col2</td>          </tr>          <tr>              <td>row2col1</td>              <td>row2col2</td>              <td>row2col3</td>          </tr>          <tr>              <td>row3col1</td>              <td>row3col2</td>              <td>row3col3</td>          </tr>          <tr>              <td>row4col1</td>              <td>row4col2</td>              <td>row4col3</td>          </tr>      </table>      <script>          let tds = document.querySelectorAll('td')          tds.forEach(item =>{              item.onclick = function () {                  item.style.background = "#ccc"              }          })      </script>  </body>  </html> |

## response-exercise.js

|  |
| --- |
| let http = require('http')  let fs = require('fs')  let tabledata = fs.readFileSync('./table.html')  let port = 3700  http.createServer((req,res)=>{       res.setHeader("content-type","text/html;charset=utf-8")       res.write(tabledata)       res.end()  }).listen(port,()=>{      console.log(`server is ready : http://localhost:${port}`);  }) |

### 效果

### 点击前

|  |
| --- |
|  |

### 点击后

|  |
| --- |
|  |

# 11网页资源加载的基本过程

## 浏览器加载页面的时候，它会依次逐行读取文件，发现需要某个资源，就会发送一次请求去获取这个资源，直到把所有需要的资源都请求回来为止，可见并不是说一个页面就是一个请求，有时候加载一个页面需要发送n多次请求，这样子才能获取所有资源，这个过程是异步的

# 12.响应练习扩展，我们把上面10的练习的样式和js代码都放到外部文件，然后在html文件中把他们引入。

|  |
| --- |
|  |

## 其实我们直接这么做是不行的。因为这么做浏览器请求回来的css文件和js文件里面的内容都是html，因为这个请求发送过去，服务器的回调函数就把html返回

|  |
| --- |
|  |

## 解决办法，其实是服务器的问题，我们需要调整回调函数中的代码

## response-exercise-extended.js

|  |
| --- |
| let http = require('http')  let fs = require('fs')  let port = 3700  let site = `http://localhost:${port}/`  http.createServer((req,res)=>{      // console.log(req.url)      //  res.setHeader("content-type","text/html;charset=utf-8")//这个不要写因为它把所有的东西都指定为html这里是不对的       let url = new URL(req.url,site)         console.log(url)       let {pathname} =url      //  console.log(pathname);       if(pathname ==='/'){          let tabledata = fs.readFileSync(\_\_dirname+'/table.html')          res.end(tabledata)       } else if(pathname==='/css/style.css'){          let cssdata = fs.readFileSync(\_\_dirname+'/css/style.css')          res.end(cssdata)       } else if(pathname === '/js/index.js'){          let jsdata = fs.readFileSync(\_\_dirname+'/js/index.js')          res.end(jsdata)       } else {          res.statusCode =404          res.end("<h1>not found</h1>")       }  }).listen(port,()=>{      console.log(`server is ready : ${site}`);  }) |

## 注意：上面这种写法还是笔记麻烦的，我们应该使用静态资源服务器

# 13.静态资源与动态资源

|  |
| --- |
|  |

## 注意：所谓的内容长时间不改变是指项目上线以后不经常改变而不是开发过程不常改变

# 14.搭建静态资源服务器

## 上面的扩展案例虽然解决了问题，但是如果我们的网站由很多静态资源，那么这个代码就会变得非常繁琐，为了解决这个问题，我们来学习静态资源服务开发，比如我们有如下需求

|  |
| --- |
|  |

## 根据需求搭建项目，结构如下

|  |
| --- |
|  |

### 我的版本

### st-server.js

|  |
| --- |
| let http = require('http')  let fs = require('fs')  let port = 9000  let site = `http://localhost:${port}/`  http.createServer((req,res)=>{      // 静态资源服务器是根据用户的输入来匹配本地资源，如果有就返回内容给用户，如果没有就返回not found      //  res.setHeader("content-type","text/html;charset=utf-8")//这个不要写因为它把所有的东西都指定为html这里是不对的      //从路径获取资源文件名称      let url = new URL(req.url,site)      //    console.log(url)       let {pathname} =url //获取路径      //  console.log(pathname);         if(fs.existsSync('.'+pathname)){ //匹配本地文件路径需要把'/index.html'变为'./intex.html'          console.log(true);          let data = fs.readFileSync(\_\_dirname+'/page'+pathname) //注意此时输入/index.html就会变为/page/index.html          res.end(data)       } else {          res.statusCode =404          res.end("<h1>not found</h1>")       }    }).listen(port,()=>{      console.log(`server is ready : ${site}`);  }) |

### ./page/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>      <h1>欢迎来到首页</h1>  </body>  </html> |

### ./page/css/app.css

|  |
| --- |
| \*{      margin: 0;      padding: 0;  }  /\* 下面是body的样式 \*/  body{      height: 100vh;      padding: 20px;      box-sizing: border-box;      background: linear-gradient(to right bottom,rgb(148,213,201),rgb(167,151,104));  } |

### ./page/images/logo.png

|  |
| --- |
|  |

## 老师的版本只是服务器文件不一样

### st-server2.js

|  |
| --- |
| let http = require('http')  let fs = require('fs')  let port = 9000  let site = `http://localhost:${port}/`  http.createServer((req,res)=>{      // 静态资源服务器是根据用户的输入来匹配本地资源，如果有就返回内容给用户，如果没有就返回not found      //  res.setHeader("content-type","text/html;charset=utf-8")//这个不要写因为它把所有的东西都指定为html这里是不对的      //从路径获取资源文件名称      let url = new URL(req.url,site)      //    console.log(url)       let {pathname} =url //获取路径      //  console.log(pathname);      let filePath = \_\_dirname+'/page' + pathname //注意此时输入/index.html就会变为/page/index.html      //异步方法读取文件      fs.readFile(filePath,(err,data)=>{        if(err){           res.end('<h1>not found</h1>')           return        }         res.end(data)      })    }).listen(port,()=>{      console.log(`server is ready : ${site}`);  }) |

## 注意，我们以及对page做了处理，输入：<http://localhost:9000/index.html> 就能访问./page/index.html, 输入：<http://localhost:9000/images/logo.png> 就能访问./page/images/logo.png

# 15.静态资源目录与网站根目录

|  |
| --- |
|  |

## 比如我们上面的静态资源服务器中的page目录就是网站的静态资源目录同时也是网站的根目录

## 我们可以把代码修改为下面的写法

|  |
| --- |
| let http = require('http')  let fs = require('fs')  let path = require('path')  let port = 9000  let site = `http://localhost:${port}/`  http.createServer((req,res)=>{      // 静态资源服务器是根据用户的输入来匹配本地资源，如果有就返回内容给用户，如果没有就返回not found      //  res.setHeader("content-type","text/html;charset=utf-8")//这个不要写因为它把所有的东西都指定为html这里是不对的      //从路径获取资源文件名称      let url = new URL(req.url,site)      //    console.log(url)       let {pathname} =url //获取路径      //  console.log(pathname);      //定义网站的根目录      // let rootPath =\_\_dirname+'/page'      let rootPath =path.resolve(\_\_dirname+'/page')        let filePath =rootPath + pathname //注意此时输入/index.html就会变为/page/index.html      //异步方法读取文件      fs.readFile(filePath,(err,data)=>{        if(err){           res.end('<h1>not found</h1>')           return        }         res.end(data)      })    }).listen(port,()=>{      console.log(`server is ready : ${site}`);  }) |

### 效果是一样的

|  |
| --- |
|  |

## 建议这么写，因为这样子可以很方便的更换网站的根目录

# 16.网页中的URL绝对路径和相对路径

## 1）绝对路径，项目中用的多

|  |
| --- |
|  |

### 除了中间那个用的比较少，另外两个用的都比较多，最后应该有应该好处，就是以后可以很方便的换主机名称，只要这个目录还在，就能够正常工作

## 2）相对路径，学习阶段用的多

|  |
| --- |
|  |

## 注意：网页的相对路径是不可靠的，尽量不要在项目中使用

# 17.页面URL的使用场景

|  |
| --- |
|  |

## 我们完善一下上面的服务器练习的html文件的代码，给他添加应该引入css文件的链接

### ./page/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>      <link rel="stylesheet" href="./css/app.css">  </head>  <body>      <h1>欢迎来到首页</h1>  </body>  </html> |

### 发现是好使的，样式生效了

|  |
| --- |
|  |

## 也可以这么写

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>Document</title>      <!-- <link rel="stylesheet" href="./css/app.css"> --> <!-- ok -->      <link rel="stylesheet" href="/css/app.css"><!-- ok -->  </head>  <body>      <h1>欢迎来到首页</h1>  </body>  </html> |

## 还有一种写法，就是使用网站的完整路径，这种写法不推荐，因为以后万一我们的域名改了，可能就会引起很多问题，最好就是这种：<link rel="stylesheet" href="/css/app.css">。因为相对路径不可靠。

# 18.http设置mime类型

|  |
| --- |
|  |

## 我们可以继续完善上面的静态资源服务器类型，为他设置mime类型我们修改st-server3.js,使我们的代码更加规范

|  |
| --- |
| let http = require('http')  let fs = require('fs')  let path = require('path')  let port = 9000  let site = `http://localhost:${port}/`  //定义应该对象来保存常见mime类型  let mimes ={      html:"text/html",      css:"text/css",      js:"text/javascript",      png:"image/png",      jpg:"image/jpeg",      gif:"image/gif",      mp4:"video/mp4",      mp3:"audio/mpeg",      json:"application/json"  }  http.createServer((req,res)=>{      // 静态资源服务器是根据用户的输入来匹配本地资源，如果有就返回内容给用户，如果没有就返回not found      //  res.setHeader("content-type","text/html;charset=utf-8")//这个不要写因为它把所有的东西都指定为html这里是不对的      //从路径获取资源文件名称      let url = new URL(req.url,site)      //    console.log(url)       let {pathname} =url //获取路径      //  console.log(pathname);      //定义网站的根目录      // let rootPath =\_\_dirname+'/page'      let rootPath =path.resolve(\_\_dirname+'/page')        let filePath =rootPath + pathname //注意此时输入/index.html就会变为/page/index.html      //异步方法读取文件      fs.readFile(filePath,(err,data)=>{        if(err){           res.end('<h1>not found</h1>')           return        }         //获取文件的后缀名,需要处理一下，因为我们不需要.         let ext = path.extname(filePath).slice(1)         console.log(ext);         //设置网站的mime类型，这里不能写死，因为这里的类型会改变，需要根据用户请求的文件的扩展名来确定mime类型         //获取对应的类型         let type = mimes[ext]         //需要看看type是否有值，如果没有就是匹配不到         if(type){          //匹配到了          res.setHeader("Content-type",type)         } else{          // 没有匹配到，如果都不是上面的类型，就可以设置为下载类型，浏览器会把这个资源下载到本地          res.setHeader("Content-type","application/octet-stream") //这种就是供下载的文件的mime类型         }           res.end(data)      })    }).listen(port,()=>{      console.log(`server is ready : ${site}`);  }) |

## 增加了else里面的类型后，我们把应该zip文件放入网页根目录page下面，然后我们去请求这个zip文件，发现浏览器会把它下载下来，注意。需要确定st-server3.js

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

# 19.http解决乱码问题

## 1)我们把上面的案例的js目录下面的app.js的内容修改一下，如

|  |
| --- |
| let img = document.querySelector('img')  img.onclick = function () {      alert("图片被点击了...")  } |

## 2）然后我们在浏览器中请求这个资源，会程序乱码

|  |
| --- |
|  |

## 3）然后我们需要解决一下这个乱码问题，解决办法很简单，在type后面+编码类型说明

### st-server3.js

|  |
| --- |
| let http = require('http')  let fs = require('fs')  let path = require('path')  let port = 9000  let site = `http://localhost:${port}/`  //定义应该对象来保存常见mime类型  let mimes ={      html:"text/html",      css:"text/css",      js:"text/javascript",      png:"image/png",      jpg:"image/jpeg",      gif:"image/gif",      mp4:"video/mp4",      mp3:"audio/mpeg",      json:"application/json"  }  http.createServer((req,res)=>{      // 静态资源服务器是根据用户的输入来匹配本地资源，如果有就返回内容给用户，如果没有就返回not found      //  res.setHeader("content-type","text/html;charset=utf-8")//这个不要写因为它把所有的东西都指定为html这里是不对的      //从路径获取资源文件名称      let url = new URL(req.url,site)      //    console.log(url)       let {pathname} =url //获取路径      //  console.log(pathname);      //定义网站的根目录      // let rootPath =\_\_dirname+'/page'      let rootPath =path.resolve(\_\_dirname+'/page')        let filePath =rootPath + pathname //注意此时输入/index.html就会变为/page/index.html      //异步方法读取文件      fs.readFile(filePath,(err,data)=>{        if(err){           res.end('<h1>not found</h1>')           return        }         //获取文件的后缀名,需要处理一下，因为我们不需要.         let ext = path.extname(filePath).slice(1)         console.log(ext);         //设置网站的mime类型，这里不能写死，因为这里的类型会改变，需要根据用户请求的文件的扩展名来确定mime类型         //获取对应的类型         let type = mimes[ext]         //需要看看type是否有值，如果没有就是匹配不到         if(type){          //匹配到了          res.setHeader("Content-type",type+";charset=utf-8")         } else{          // 没有匹配到，如果都不是上面的类型，就可以设置为下载类型，浏览器会把这个资源下载到本地          res.setHeader("Content-type","application/octet-stream") //这种就是供下载的文件的mime类型         }           res.end(data)      })    }).listen(port,()=>{      console.log(`server is ready : ${site}`);  }) |

### 效果：

|  |
| --- |
|  |
|  |

### 注意：在HTML文件在可以使用meta标签来设置字符编码，如：<meta charset="UTF-8">，那么问题来了，究竟meta标签和服务器的setHeader方法那个的优先级高？

### 我们来做关于试验，在setHeader方法里面把编码改为gbk，而meta标签的编码为utf-8

|  |
| --- |
|  |
|  |

### 效果：

|  |
| --- |
|  |

### 出乱码了

## 我们可以把他们的编码互换

|  |
| --- |
|  |
|  |

### 效果：

|  |
| --- |
|  |

### 可见setHeader方法的优先级比meta标签要高，

### 不过我们建议把两个的编码设置为一致

### 另外，css，js和图片是没有必要设置字符集编码的,即使他们有乱码，也不影响使用，因为我们确实不会直接去请求js文件，我们把html文件修改一下，在里面链接logo.png图片和app.js文件

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <!-- <meta charset="gbk"> -->      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <title>Document</title>      <!-- <link rel="stylesheet" href="./css/app.css"> --> <!-- ok -->      <link rel="stylesheet" href="/css/app.css"><!-- ok -->  </head>  <body>      <h1>欢迎来到首页</h1>      <img src="/images/logo.png" alt="" >      <script src="/js/app.js"></script>  </body>  </html> |

### 效果：

|  |
| --- |
|  |

#### 可见，的确不影响使用

## 所以，我们把上面的代码再完善一下

### st-server3.js

|  |
| --- |
| let http = require('http')  let fs = require('fs')  let path = require('path')  let port = 9000  let site = `http://localhost:${port}/`  //定义应该对象来保存常见mime类型，以后我们有更好的方法  let mimes ={      html:"text/html",      css:"text/css",      js:"text/javascript",      png:"image/png",      jpg:"image/jpeg",      gif:"image/gif",      mp4:"video/mp4",      mp3:"audio/mpeg",      json:"application/json"  }  http.createServer((req,res)=>{      // 静态资源服务器是根据用户的输入来匹配本地资源，如果有就返回内容给用户，如果没有就返回not found      //  res.setHeader("content-type","text/html;charset=utf-8")//这个不要写因为它把所有的东西都指定为html这里是不对的      //从路径获取资源文件名称      let url = new URL(req.url,site)      //    console.log(url)       let {pathname} =url //获取路径      //  console.log(pathname);      //定义网站的根目录      // let rootPath =\_\_dirname+'/page'      let rootPath =path.resolve(\_\_dirname+'/page')        let filePath =rootPath + pathname //注意此时输入/index.html就会变为/page/index.html      //异步方法读取文件      fs.readFile(filePath,(err,data)=>{        if(err){           res.end('<h1>not found</h1>')           return        }         //获取文件的后缀名,需要处理一下，因为我们不需要.         let ext = path.extname(filePath).slice(1)         console.log(ext);         //设置网站的mime类型，这里不能写死，因为这里的类型会改变，需要根据用户请求的文件的扩展名来确定mime类型         //获取对应的类型         let type = mimes[ext]         //需要看看type是否有值，如果没有就是匹配不到         if(type){          //匹配到了          if(ext==='html'){              res.setHeader("Content-type",type+";charset=utf-8") //只有html才需要字符集编码          } else {            res.setHeader("Content-type",type)//css,js,图片影片音乐都不需要设置字符集编码          }         } else{          // 没有匹配到，如果都不是上面的类型，就可以设置为下载类型，浏览器会把这个资源下载到本地          res.setHeader("Content-type","application/octet-stream") //这种就是供下载的文件的mime类型         }           res.end(data)      })    }).listen(port,()=>{      console.log(`server is ready : ${site}`);  }) |

# 20.http完善错误处理

## error对象的错误代码可以在nodejs的官网找到,结果如下

|  |
| --- |
| 这是编写 Node.js 程序时经常遇到的系统错误列表。 有关完整的列表，请参阅 [errno(3) 手册页](http://url.nodejs.cn/32d7Vh)。   * EACCES（权限被拒绝）：试图以文件访问权限禁止的方式访问文件。 * EADDRINUSE（地址已被使用）：尝试将服务器（[net](https://nodejs.cn/api-v12/net.html)、[http](https://nodejs.cn/api-v12/http.html)、或 [https](https://nodejs.cn/api-v12/https.html)）绑定到本地地址失败，因为本地系统上的另一台服务器已经占用了该地址。 * ECONNREFUSED（连接被拒绝）：由于目标机器主动拒绝，无法建立连接。 这通常是由于尝试连接到在外部主机上处于非活动状态的服务。 * ECONNRESET（对等方重置连接）：连接被对等方强行关闭。。 这通常是由于超时或重新启动导致远程套接字上的连接丢失造成的。 通常通过 [http](https://nodejs.cn/api-v12/http.html) 和 [net](https://nodejs.cn/api-v12/net.html) 模块遇到。 * EEXIST（文件存在）：现有文件是要求目标不存在的操作的目标。 * EISDIR（是目录）：操作期望文件，但给定的路径名​​是目录。 * EMFILE（系统中打开的文件太多）：已达到系统上允许的[文件描述符](http://url.nodejs.cn/4xWwJy)的最大数量，并且在至少一个描述符被关闭之前无法满足对另一个描述符的请求。 同时打开多个文件时会遇到这种情况，尤其是在进程的文件描述符限制较低的系统（特别是 macOS）上。 要弥补低限制，则在将运行 Node.js 进程的同一个 shell 中运行 ulimit -n 2048。 * ENOENT（无此文件或目录）：通常由 [fs](https://nodejs.cn/api-v12/fs.html) 操作引发，以指示指定路径名的组件不存在。 给定路径找不到任何实体（文件或目录）。 * ENOTDIR（不是目录）：给定路径名的组件存在，但不是期望的目录。 通常由 [fs.readdir](https://nodejs.cn/api-v12/fs.html#fs_fs_readdir_path_options_callback) 引起。 * ENOTEMPTY（目录不为空）：有条目的目录是需要空目录的操作目标，通常是 [fs.unlink](https://nodejs.cn/api-v12/fs.html#fs_fs_unlink_path_callback)。 * ENOTFOUND（域名系统查找失败）：表示 EAI\_NODATA 或 EAI\_NONAME 的域名系统失败。 这不是标准的 POSIX 错误。 * EPERM（不允许操作）：试图执行需要提升权限的操作。 * EPIPE（断开的管道）：对没有进程读取数据的管道、套接字或 FIFO 的写操作。 通常发生在 [net](https://nodejs.cn/api-v12/net.html) 和 [http](https://nodejs.cn/api-v12/http.html) 层，表示正在写入的流的远程端已关闭。 * ETIMEDOUT（操作超时）：连接或发送请求失败，因为连接方在一段时间后没有正确地响应。 通常发生在 [http](https://nodejs.cn/api-v12/http.html) 或 [net](https://nodejs.cn/api-v12/net.html)。 通常表明 socket.end() 没有被正确地调用。 |

## 我们以前把所有的错误都归为一类，这样子是不对的，我们需要根据错误对象的code来分别处理

## 我们的服务器有一个小细节，当我们只是访问/，就会出现一个错误：EISDIR,说期望访问文件但是得到目录，我们把它处理一下，就是当pathname===/,我们就在他的后面+”index.html”

|  |
| --- |
| let http = require('http')  let fs = require('fs')  let path = require('path')  let port = 9000  let site = `http://localhost:${port}/`  //定义应该对象来保存常见mime类型  let mimes ={      html:"text/html",      css:"text/css",      js:"text/javascript",      png:"image/png",      jpg:"image/jpeg",      gif:"image/gif",      mp4:"video/mp4",      mp3:"audio/mpeg",      json:"application/json"  }  http.createServer((req,res)=>{      // 静态资源服务器是根据用户的输入来匹配本地资源，如果有就返回内容给用户，如果没有就返回not found      //  res.setHeader("content-type","text/html;charset=utf-8")//这个不要写因为它把所有的东西都指定为html这里是不对的      //从路径获取资源文件名称      let url = new URL(req.url,site)      //    console.log(url)       let {pathname} =url //获取路径       //处理/,相当于重定向,但是又不是真正意义上的重定向       if(pathname==='/'){          pathname +='index.html'       }      //  console.log(pathname);      //定义网站的根目录      // let rootPath =\_\_dirname+'/page'      let rootPath =path.resolve(\_\_dirname+'/page')        let filePath =rootPath + pathname //注意此时输入/index.html就会变为/page/index.html      //异步方法读取文件      fs.readFile(filePath,(err,data)=>{        if(err){          //   console.log(err);            res.setHeader("Content-type","text/html;charset=utf-8")           switch(err.code){              //找不到资源              case 'ENOENT': //错误代码可以在nodejs官网找到                  res.statusCode = 404                  res.end('<h1>not found</h1>')              break;              //没有权限              case 'EPERM':                  res.statusCode = 403                  res.end('<h1>没有权限</h1>')              break;              default:                  break;           }             return        }         //获取文件的后缀名,需要处理一下，因为我们不需要.         let ext = path.extname(filePath).slice(1)         console.log(ext);         //设置网站的mime类型，这里不能写死，因为这里的类型会改变，需要根据用户请求的文件的扩展名来确定mime类型         //获取对应的类型         let type = mimes[ext]         //需要看看type是否有值，如果没有就是匹配不到         if(type){          //匹配到了          if(ext==='html'){              res.setHeader("Content-type",type+";charset=utf-8") //只有html才需要字符集编码          } else {            res.setHeader("Content-type",type)          }         } else{          // 没有匹配到，如果都不是上面的类型，就可以设置为下载类型，浏览器会把这个资源下载到本地          res.setHeader("Content-type","application/octet-stream") //这种就是供下载的文件的mime类型         }           res.end(data)      })    }).listen(port,()=>{      console.log(`server is ready : ${site}`);  }) |

## 还有一点，我们的静态资源服务器一般只响应get请求，如果用户使用post或者其他请求，我们需要给他返回一个错误并且停止程序的执行，这个处理必须在回调函数的开始就要处理

|  |
| --- |
| let http = require('http')  let fs = require('fs')  let path = require('path')  let port = 9000  let site = `http://localhost:${port}/`  //定义应该对象来保存常见mime类型  let mimes ={      html:"text/html",      css:"text/css",      js:"text/javascript",      png:"image/png",      jpg:"image/jpeg",      gif:"image/gif",      mp4:"video/mp4",      mp3:"audio/mpeg",      json:"application/json"  }  http.createServer((req,res)=>{      //注意我们的静态资源服务器是处理get请求的，我们需要首先判断请求方式是否正确，如果用户发送post请求，我们需要返回一个错误      if(req.method!=='GET'){          res.setHeader("Content-type","text/html;charset=utf-8")          res.statusCode = 405          res.end('<h1>请求方式错误</h1>')          return      }      // 静态资源服务器是根据用户的输入来匹配本地资源，如果有就返回内容给用户，如果没有就返回not found      //  res.setHeader("content-type","text/html;charset=utf-8")//这个不要写因为它把所有的东西都指定为html这里是不对的      //从路径获取资源文件名称      let url = new URL(req.url,site)      //    console.log(url)       let {pathname} =url //获取路径       //处理/,相当于重定向       if(pathname==='/'){          pathname +='index.html'       }      //  console.log(pathname);      //定义网站的根目录      // let rootPath =\_\_dirname+'/page'      let rootPath =path.resolve(\_\_dirname+'/page')        let filePath =rootPath + pathname //注意此时输入/index.html就会变为/page/index.html      //异步方法读取文件      fs.readFile(filePath,(err,data)=>{        if(err){          //   console.log(err);            res.setHeader("Content-type","text/html;charset=utf-8")           switch(err.code){              //找不到资源              case 'ENOENT': //错误代码可以在nodejs官网找到                  res.statusCode = 404                  res.end('<h1>not found</h1>')              break;              //没有权限              case 'EPERM':                  res.statusCode = 403                  res.end('<h1>没有权限</h1>')              break;              default:                  break;           }             return        }         //获取文件的后缀名,需要处理一下，因为我们不需要.         let ext = path.extname(filePath).slice(1)         console.log(ext);         //设置网站的mime类型，这里不能写死，因为这里的类型会改变，需要根据用户请求的文件的扩展名来确定mime类型         //获取对应的类型         let type = mimes[ext]         //需要看看type是否有值，如果没有就是匹配不到         if(type){          //匹配到了          if(ext==='html'){              res.setHeader("Content-type",type+";charset=utf-8") //只有html才需要字符集编码          } else {            res.setHeader("Content-type",type)          }         } else{          // 没有匹配到，如果都不是上面的类型，就可以设置为下载类型，浏览器会把这个资源下载到本地          res.setHeader("Content-type","application/octet-stream") //这种就是供下载的文件的mime类型         }           res.end(data)      })    }).listen(port,()=>{      console.log(`server is ready : ${site}`);  }) |

## 我们还要添加服务器内部错误的处理

|  |
| --- |
| let http = require('http')  let fs = require('fs')  let path = require('path')  let port = 9000  let site = `http://localhost:${port}/`  //定义应该对象来保存常见mime类型  let mimes ={      html:"text/html",      css:"text/css",      js:"text/javascript",      png:"image/png",      jpg:"image/jpeg",      gif:"image/gif",      mp4:"video/mp4",      mp3:"audio/mpeg",      json:"application/json"  }  http.createServer((req,res)=>{      //注意我们的静态资源服务器是处理get请求的，我们需要首先判断请求方式是否正确，如果用户发送post请求，我们需要返回一个错误      if(req.method!=='GET'){          res.setHeader("Content-type","text/html;charset=utf-8")          res.statusCode = 405          res.end('<h1>请求方式错误</h1>')          return      }      // 静态资源服务器是根据用户的输入来匹配本地资源，如果有就返回内容给用户，如果没有就返回not found      //  res.setHeader("content-type","text/html;charset=utf-8")//这个不要写因为它把所有的东西都指定为html这里是不对的      //从路径获取资源文件名称      let url = new URL(req.url,site)      //    console.log(url)       let {pathname} =url //获取路径       //处理/,相当于重定向       if(pathname==='/'){          pathname +='index.html'       }      //  console.log(pathname);      //定义网站的根目录      // let rootPath =\_\_dirname+'/page'      let rootPath =path.resolve(\_\_dirname+'/page')        let filePath =rootPath + pathname //注意此时输入/index.html就会变为/page/index.html      //异步方法读取文件  fs.readFile(filePath,(err,data)=>{  //完善错误处理        if(err){          //   console.log(err);            res.setHeader("Content-type","text/html;charset=utf-8")           switch(err.code){              //找不到资源              case 'ENOENT': //错误代码可以在nodejs官网找到                  res.statusCode = 404                  res.end('<h1>not found</h1>')              break;              //没有权限              case 'EPERM':                  res.statusCode = 403                  res.end('<h1>没有权限</h1>')              break;              default:                  //如果都不是上面的错误，就是服务器内部错误                  res.statusCode =500                  res.end('<h1>服务器内部错误</h1>')                  break;           }             return        }         //获取文件的后缀名,需要处理一下，因为我们不需要.         let ext = path.extname(filePath).slice(1)         console.log(ext);         //设置网站的mime类型，这里不能写死，因为这里的类型会改变，需要根据用户请求的文件的扩展名来确定mime类型         //获取对应的类型         let type = mimes[ext]         //需要看看type是否有值，如果没有就是匹配不到         if(type){          //匹配到了          if(ext==='html'){              res.setHeader("Content-type",type+";charset=utf-8") //只有html才需要字符集编码          } else {            res.setHeader("Content-type",type)          }         } else{          // 没有匹配到，如果都不是上面的类型，就可以设置为下载类型，浏览器会把这个资源下载到本地          res.setHeader("Content-type","application/octet-stream") //这种就是供下载的文件的mime类型         }           res.end(data)      })    }).listen(port,()=>{      console.log(`server is ready : ${site}`);  }) |

# 21.get方法和post方法的应用场景与区别

## 1. 应用场景

|  |
| --- |
|  |

## 2. 区别

|  |
| --- |
|  |