

## 课程目标



实践巩固



全栈应用



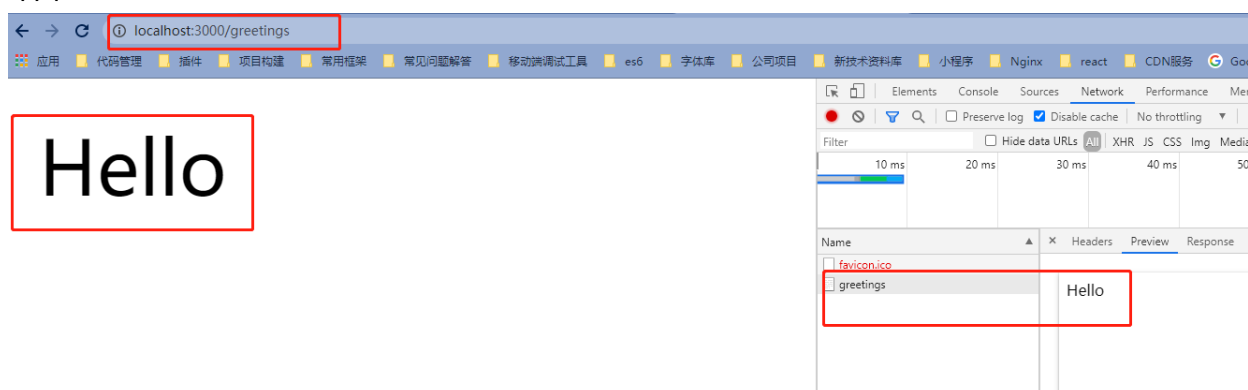
架构思想

## Method和解析Body

### 查询GET /product

```
1 //使用node GET请求
2 const express = require('express')
3 const app = express()
4 app.get('/greetings',(req,res)=>{
5   res.send('Hello')
6 })
7
8 app.listen(3000,(req,res)=>{
9   console.log(23232)
10 })
11 //使用node 开启某个服务可以下载 nodemon
12 //nodemon进行热加载(
13 //不必每次修改都要去重新启动一次)
```

结果:



### 新增POST /product

```
1 //使用node GET请求 Post请求
2 const express = require('express')
3 const app = express()
```

```

4 //get
5 app.get('/greetings',(req,res)=>{
6   res.send('Hello')
7 })
8
9 //post
10 app.post('/product',(req,res)=>{
11   //post在服务器创建一个资源
12   res.set('Content-Type','application/json') //指定返回的资源格式为JSON
13   res.status(201).send(JSON.stringify({data:'资源已经创建,但是未实现'}))
14   //比较规范的做法, 返回201 表示资源已经创建
15 })
16
17 app.listen(3000,(req,res)=>{
18   console.log(23232)
19 })

```

POST product 返回201

使用fetch直接调用(图一) (查看相关结果, 图二, 图三)

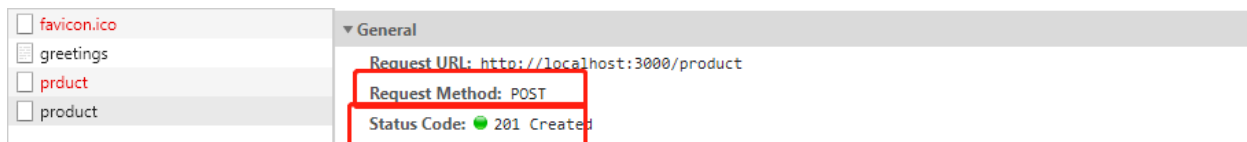
图一

```

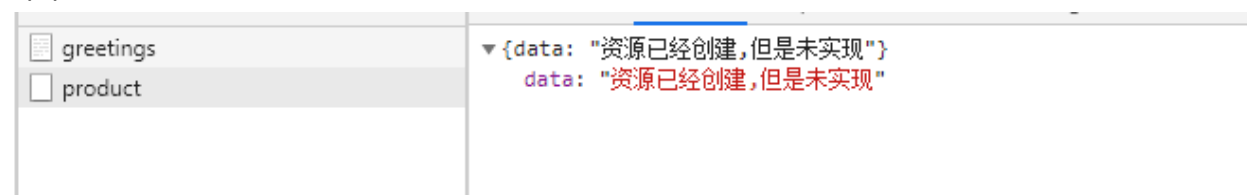
> fetch('/product',{method:'POST'})
< ▶ Promise {<pending>}
> |

```

图二



图三



指定返回资源的格式res.set('Content-Type','application/json'), 就可以查看response的Headers



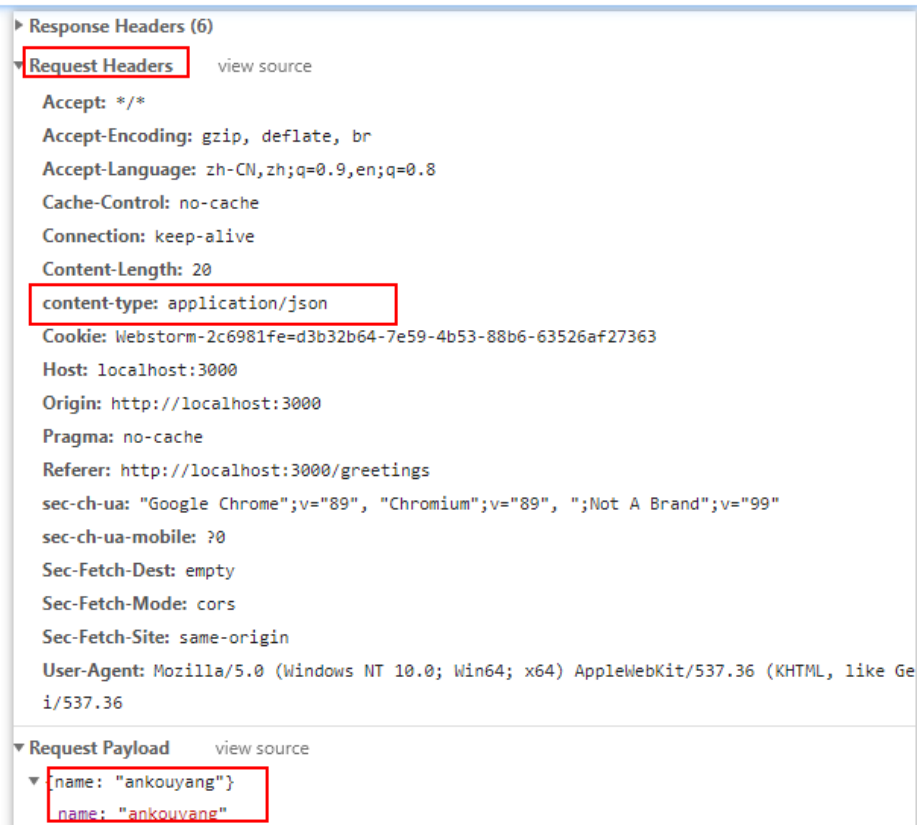
在客户端，可以获取到res的headers中Content-Type

```
> fetch('/product',{method:'POST'}).then(res=>console.log(res.headers.get('Content-type')))  
< ▶ Promise {<pending>}  
application/json; charset=utf-8
```

## POST请求body传参，服务端如何解析body

客户端body传参

```
fetch('/product',{method:'POST',headers:{'content-type':'application/json'},body:JSON.stringify({name:'ankouyang'})})  
▶ Promise {<pending>}
```



服务端解析Body, 这里用原生的写一下，其实是可以用到express的中间件去实现的

```
1 const express = require('express')  
2 const app = express()  
3 //get  
4 app.get('/greetings',(req,res)=>{
```

```

5   res.send('Hello')
6 })
7
8 //post
9 app.post('/product',(req,res)=>{
10   //使用application/json
11   const contentType = req.headers['content-type']
12   let requestText = ""
13   let body =null
14   req.on('data',(buffer)=>{
15     //8bit -byte 把流转化为utf-8数据格式
16     requestText +=buffer.toString('utf-8')
17   })
18   //数据传输完毕后
19   req.on('end',()=>{
20     switch (contentType) {
21       //通过客户端的请求头content-type
22       case 'application/json':
23         body = JSON.parse(requestText) //解析body
24         res.set('Content-Type','application/json')
25         res.status(201).send(JSON.stringify({data:body}))
26         break
27       }
28     })
29
30 })
31
32 app.listen(3000,(req,res)=>{
33   console.log(23232)
34 })

```

修改PUT /product/:id (restful风格)

服务端：

```

//put
app.put('/product/:id',(req,res)=>{
  res.status(204).send() //204成功了 但是无内容返回给你
})

```

客户端请求：

```
> fetch('/product/123',{method:'PUT',headers:{'content-type':'application/json'},body:JSON.stringify({name:'A'})})
< ▶ Promise {<pending>}
```

服务端返回：

Name	▲	×	Headers	Preview	Response	Initiator	Timing	Cookies
<input type="checkbox"/> 123			▼ General					
<input type="checkbox"/> greetings			Request URL: http://localhost:3000/product/123					
			Request Method: PUT					
			Status Code: 204 No Content					
			Remote Address: [::1]:3000					
			Referrer Policy: strict-origin-when-cross-origin					
			▶ Response Headers (3)					
			▶ Request Headers (18)					
			▼ Request Payload view source					
			▼ {name: "A"}					
			name: "A"					

删除DELETE /product/:id (resful风格)

服务端：

```
//delete
app.put('/product/:id',(req,res)=>{
  res.status(204).send() //204成功了 但是无内容返回给你
})
```

客户端请求：

```
fetch('/product/123',{method:'DELETE',headers:{'content-type':'application/json'},body:JSON.stringify({name:'A'})})
▶ Promise {<pending>}
```

服务端返回：

<input type="checkbox"/> 123			▼ General					
<input type="checkbox"/> 123			Request URL: http://localhost:3000/product/123					
<input checked="" type="checkbox"/> 123			Request Method: DELETE					
<input type="checkbox"/> greetings			Status Code: 204 No Content					
			Remote Address: [::1]:3000					
			Referrer Policy: strict-origin-when-cross-origin					
			▶ Response Headers (3)					
			▶ Request Headers (18)					
			▼ Request Payload view source					
			▼ {name: "A"}					
			name: "A"					

## 跳转Header和3XX状态码

## 实战-重定向

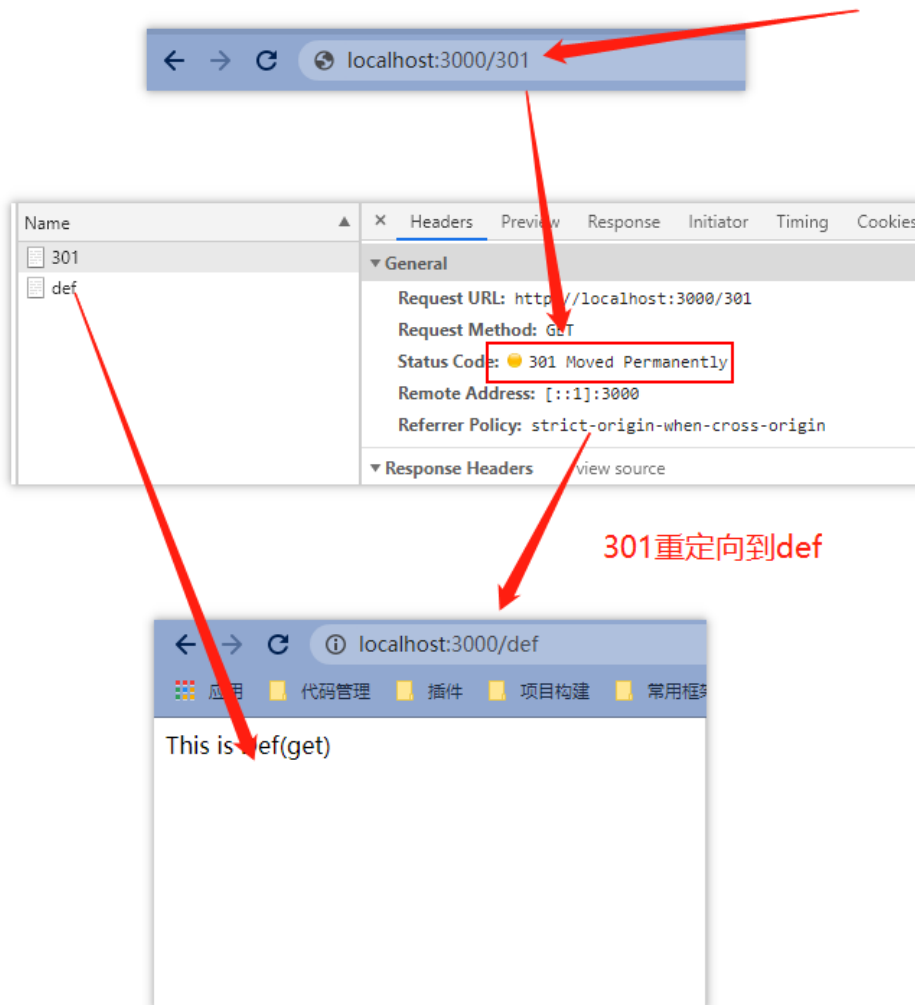
- 301
- 302
- 303
- 307
- 308

## 重定向301

服务端：

```
1  const express = require('express')
2  const app = express()
3  //get
4  app.get('/301', (req, res) => {
5    res.redirect(301, '/def')
6  })
7  //重定向接口
8  app.get('/def', (req, res) => {
9    res.send('This is Def(get)')
10 })
11
12
13 app.listen(3000, (req, res) => {
14   console.log(23232)
15 })
```

客户端：



注意：这个重定向是浏览器干的，我们可以通过curl http://localhost:3000/301了可以看出，只是提示Redirecting to /def

```
C:\Users\yangke>curl http://localhost:3000/301
Moved Permanently. Redirecting to /def
C:\Users\yangke>
```

## 重定向302/303/307

服务端:

```
1 const express = require('express')
2 const app = express()
3 //get post
4 //302/303/307
5 app.get('/301', (req, res) => {
```

```
6   res.redirect(301, '/def')
7 })
8 app.post('/301', (req, res) => {
9   res.redirect(301, '/def')
10 })
11 //302
12 app.get('/302', (req, res) => {
13   res.redirect(302, '/def')
14 })
15 app.post('/302', (req, res) => {
16   res.redirect(302, '/def')
17 })
18 //303
19 app.get('/303', (req, res) => {
20   res.redirect(303, '/def')
21 })
22
23 app.post('/303', (req, res) => {
24   res.redirect(303, '/def')
25 })
26
27 //get 直接报错 307
28 app.get('/307', (req, res) => {
29   res.redirect(307, '/def')
30 })
31 //post 307
32 app.post('/307', (req, res) => {
33   res.redirect(307, '/def')
34 })
35
36 //重定向接口
37 app.get('/def', (req, res) => {
38   res.send('This is Def(get)')
39 })
40
41 app.post('/def', (req, res) => {
42   res.send('This is Def(post)')
43 })
44
45 app.listen(3000, (req, res) => {
```



```
46 console.log(23232)
47 })
```

客户端：302 303方式,不管是GET还是POST请求方式，重定向都是GET返回。  
307方式，只能支持post方式，重定向方式也是POST 如何所示:

```
> fetch('/307',{method:'POST'})
< ▶ Promise {<pending>}
```

<input type="checkbox"/> 307 <input type="checkbox"/> def	Request URL: http://localhost:3000/307 Request Method: POST Status Code: 🟡 307 Temporary Redirect Remote Address: [::1]:3000 Referrer Policy: strict-origin-when-cross-origin
<input type="checkbox"/> 307 <input type="checkbox"/> def	Request URL: http://localhost:3000/def Request Method: POST Status Code: 🟢 200 OK Remote Address: [::1]:3000 Referrer Policy: strict-origin-when-cross-origin

307get方式 直接404

```
> fetch('/307',{method:'GET'})
< ▶ Promise {<pending>}
```

✖ GET http://localhost:3000/307 404 (Not Found)

307get方式，直接是404

(遵守这个协议的话，可以有利于seo优化)

## 错误处理4XX和5XX

为下列场景返回不同的错误码

- 用户没有登录
- 服务器报错
- 内容没有找到
- 不支持POST方法

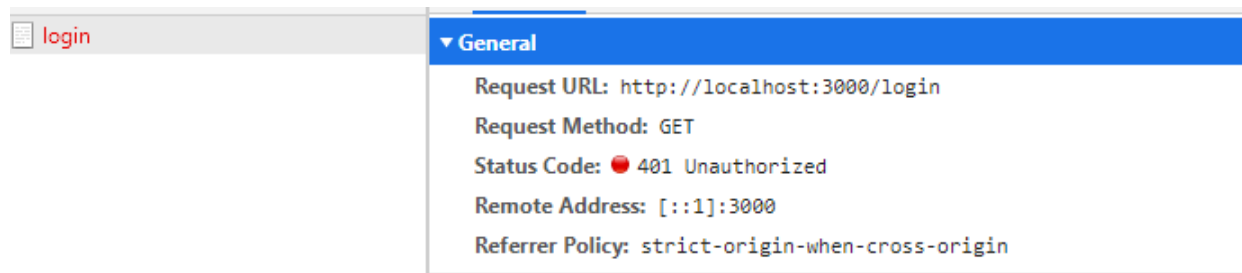
用户没有登录

服务端:

```
1 // 前面省略
```

```
2 app.get('/login',(req,res)=>{
3   res.sendStatus(401)//未授权
4   // res.sendStatus(403) 禁止访问
5   // res.sendStatus(404) 未找到资源，内容未找到
6   // res.sendStatus(405) 方法不被允许
7 })
```

客户端：

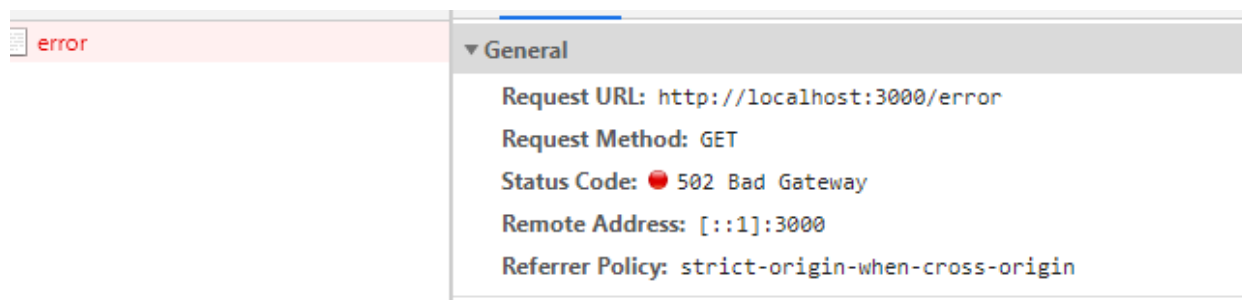


## 服务器报错

服务端：

```
1 app.get('/error',(req,res)=>{
2   res.sendStatus(502) //网关错误
3   // res.sendStatus(500) 服务错误
4   // res.sendStatus(504) 网关超时
5   // res.sendStatus(501) 没有实现，这个接口已经存在
6   // res.sendStatus(503) 服务不可用
7 })
```

客户端：



。。。。等等

## 课程小结

Body解析过程中的协商技巧很有用

认真对待3XX状态码（谁开发,谁运维，前端工程师要配置反向代理，负载均衡）

错误处理遵循HTTP协议