## express基础知识(4.17.1)

- 能够使用express.static()快速托管静态资源
- 能够使用express路由精简项目结构
- 能够使用常见的express中间件
- 能够使用express创建API接口
- 能够在express中启用cors跨域资源共享

```
http://127.0.0.1/
```

Express能做什么：

web网站服务器

API接口服务器

```
npm i express@4.17.1
```

```
const express = require('express')
const app = express()      //创建web服务器的实例

//监听端口
app.listen(80,()=>{
    console.log('服务器启动成功')    //启动成功后调用回调函数
})
```

**监听get,post请求**

**查询参数 req.query**

**默认情况下req.query是一个空对象**

```
app.get('请求URL', function(req,res){
    //发送json对象
    res.send({name:"zcs",age:20,gender:"男"})      //express提供的res.send方法
})
app.post('请求地址',function(req,res) {
  //向客户端发送文本内容
  res.send('请求成功')
})

```

**获取URL中的动态参数**

**通过req.params可以获取到URL中通过":"匹配到的动态参数**

**req.params默认也是个空对象**

```
app.get('/user/:id/:name',(req,res)=>{     //http:127.0.0.1/user/123/zhangchengsen
    res.send(req.params.id)
})
```

## express.static()托管静态资源

通过express.static() 我们可以非常方便地创建一个静态资源服务器
通过如下代码就可以将public目录下面的图片,css文件,javascript文件对外开放访问了(向外托管)
注：存放静态资源的文件目录名不会出现在url中

```
app.use(express.static('public'))
//现在可以访问publick下所有文件了
http://127.0.0.1/images/bg.jpg
http://127.0.0.1/css/style.css
```

托管多个静态目录

```
app.use(express.static('public'))
app.use(express.static('files'))
```

访问静态资源时,express.static()函数会根据目录的添加顺寻查找所需的文件
找到了直接返回 没找到就去下一个目录里查找

### 挂载路径前缀

```
app.use('/static',express.static('./public'))
```

想要访问到静态资源需要通过**http:127.0.0.1/static/images/...**访问

## express路由

**路由就是映射关系**
**在express中 路由指的是客户端的请求与服务器处理函数之间的映射关系**
**由三部分组成**

- **请求类型**
- **请求的URL地址**
- **处理函数**

**app.METHOD(PATH,HANDLER)**

路由的匹配过程
按照先后顺序 请求的类型和URL匹配成功 会交给对应的function处理

```js
const express = require('express')
// 创建 Web 服务器，命名为 app
const app = express()

// 挂载路由
app.get('/', (req, res) => { res.send('Hello World.') })
app.post('/', (req, res) => { res.send('Post Request.') })

// 启动 Web 服务器
app.listen(80, () => { console.log('server running at http://127.0.0.1') })
```

1.最简单的挂载(少用)

**2.模块化路由**

**将路由抽离为模块化的步骤**

- **创建路由模块对应.js文件**
- **调用express.Router()函数创建路由对象**
- **向路由实例对象上挂载具体的路由**
- **使用module.exports向外共享路由对象**
- **app.use()函数注册路由模块**

```js
var express = require('express')                    // 1. 导入 express
var router = express.Router()                       // 2. 创建路由对象

router.get('/user/list', function (req, res) {      // 3. 挂载获取用户列表的路由
  res.send('Get user list.')
})
router.post('/user/add', function (req, res) {      // 4. 挂载添加用户的路由
  res.send('Add new user.')
})

module.exports = router                             // 5. 向外导出路由对象
```

```
//=====user.js======
```

```
2
3 var router = express.Router()
4 router.get('/user/list',(req,res)=>{
5     res.send('.....')
6 })
7 router.post
8 //module.exports 本来是一个空对象
9 //如果module.exports = {router} 就是对象中的对象了
10 module.exports = router
11 //👆直接只导出router
```

```
1 //=====app.js======
2 //导入路由模块
3 const userRouter = require('./router/user.js')
4 //注册路由模块
5 app.use(userRouter)
6 //  👆
```

**app.use注册 全局中间件**

**为路由模块添加前缀**

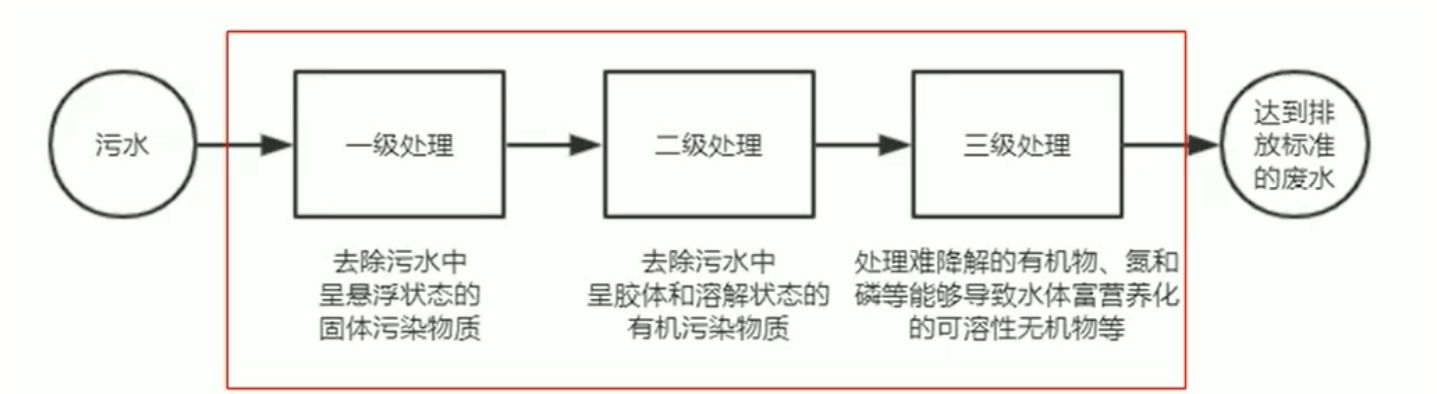
```
1 app.use('/api',userRouter)
```

↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑

访问前需要添加/api地址才可访问

```
1 //如:
2     http://127.0.0.1/api/user
```

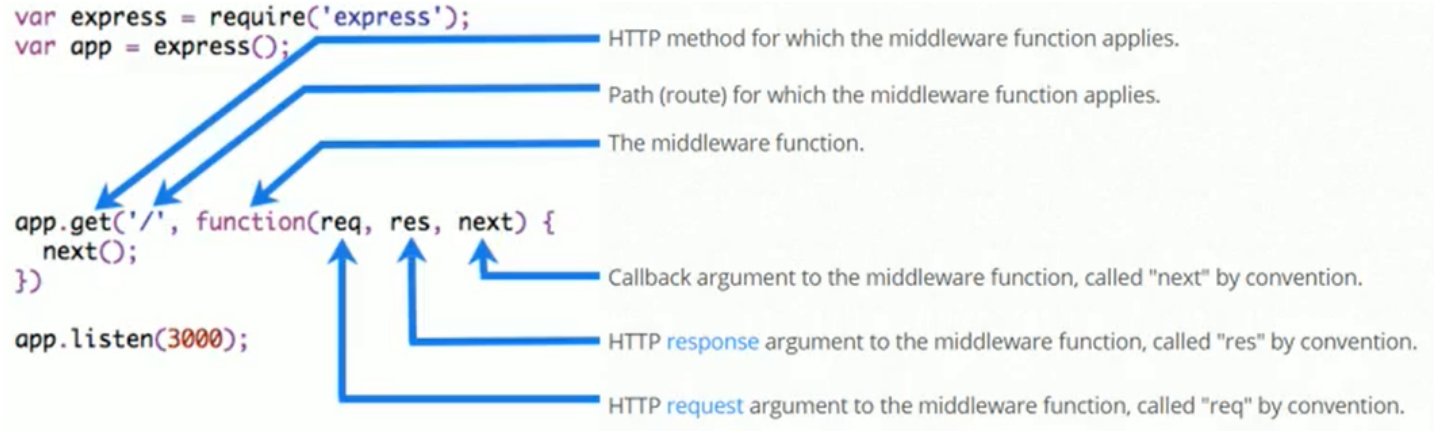
## 中间件(Middleware)




上一级的输出为下一级的输入

当一个请求到达Express的服务器之后,可以连续调用多个中间件,从而对这次请求进行**预处理。**

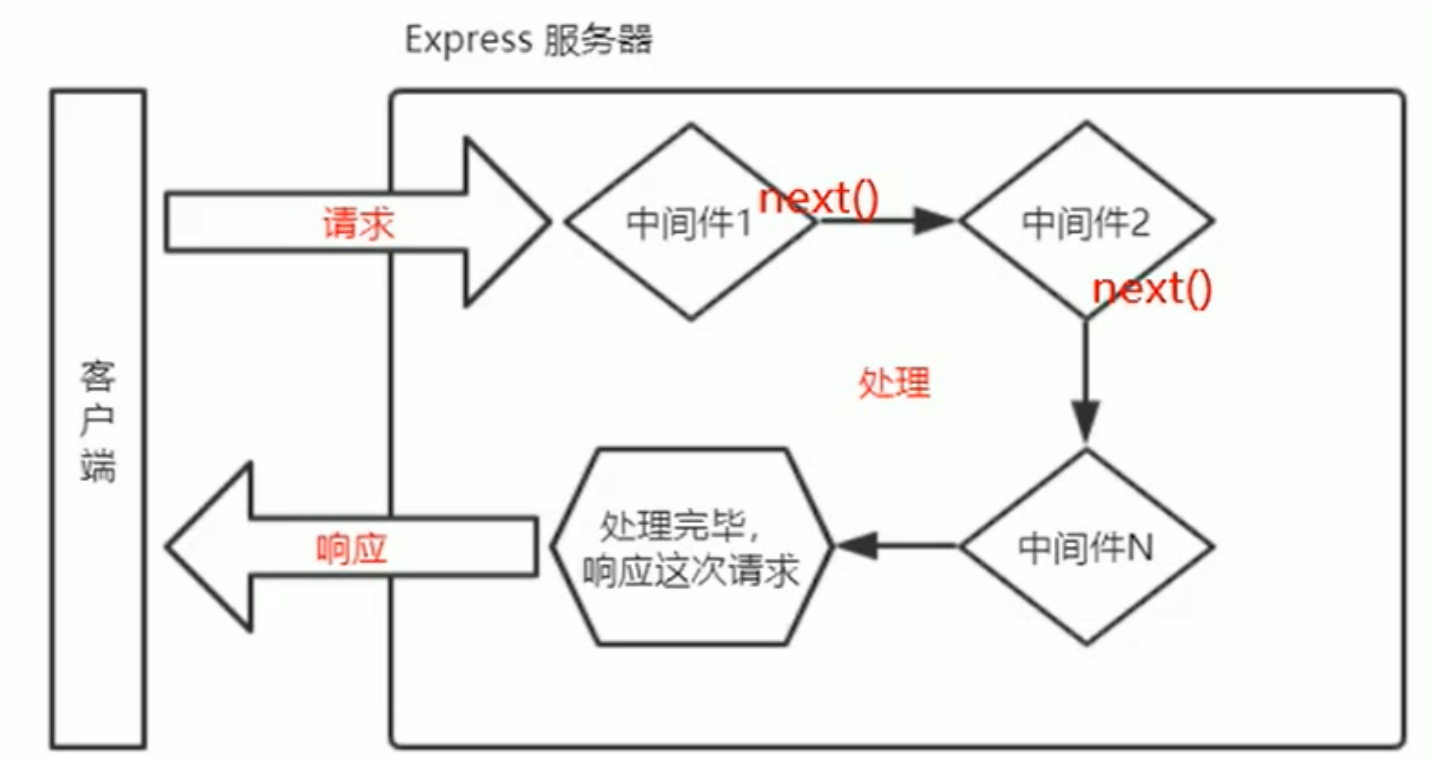
本质是一个function处理函数




包含next说明它是一个中间件

**next函数**

**next是实现多个中间件连续调用的关键,他表示把流转关系转交给下一个中间件或路由**




**定义中间件函数**

```
const mw = function(req,res,next) {
    console.log('这是一个最简单的中间件函数')
    next()
}
```

**全局生效的中间件**

客户端发起的**任何请求**,到达服务器之后,**都会触发的中间件**,叫做全局生效的中间件

通过调用app.use(中间件函数),即可定义一个全局生效的中间件函数

```
app.use(mw)
```

**全局中间件简化形式**

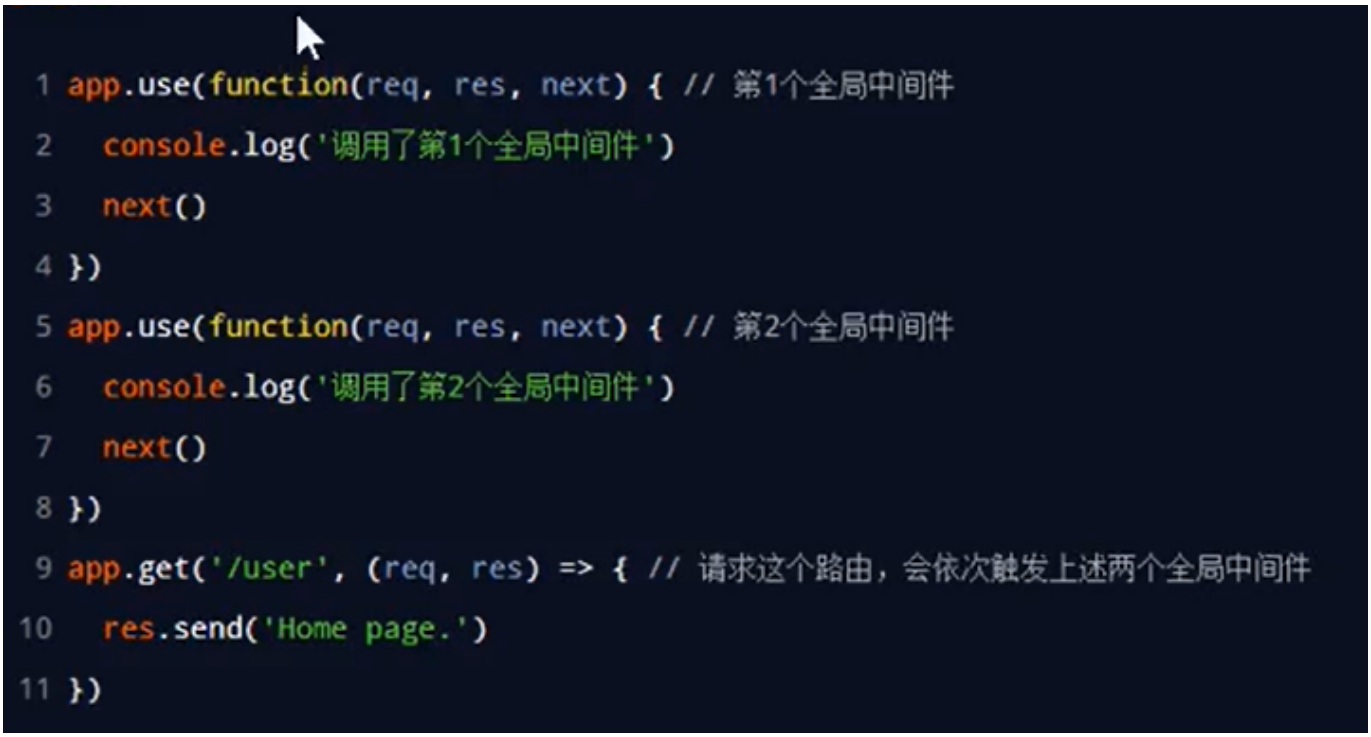
```
app.use(function(req,res,next){
    console.log('abcd')
    next()
})
```

**中间件的作用**

- 多个中间件之间,**共享同一份req和res**。基于这样的特性,我们可以在**上游**中间件中,统一为req和res对象添加自定义的属性或方法,供下游的中间件或路由进行使用。

```
req.a = 10                    //添加自定义属性或方法
```

**连续定义多个中间件**




**局部生效的中间件**

**不适用app.use定义的中间件**

```
const mw = function(req,res,next) {
    if(req.url == "/user/userList")
    {
            res.end('404notFound')
    }
    next()
}
```

```
app.get('/',mw,function(req,res)=>{
    res.send('home page.')
    //  局部调用同
})
//  不会影响下面
app.get('/user')...
```

### 定义多个局部中间件

两种等价写法

```
app.get('/',  [m1,m2]  ,(req,res)=>{
    res.send("666")
})
app.get('/', m1 , m2 ,(req,res)=>{
    res.send("666")
})
```

### 中间件注意事项

- **一定要在路由之前注册中间件**
- **客户端发送的请求可以连续调用多个中间件进行处理**
- **执行完中间件的业务代码之后,不要忘记调用next()函数**
- **为了防止代码逻辑混乱,调用next()函数后不要再写额外的代码**
- **连续调用多个中间件时,多个中间件,共享req和res对象**

### 中间件的分类

**5大类**

- **应用级别的中间件 app.use() app.get() app.get()绑定到app实例上的中间件 全局**
- **路由级别的中间件 绑定到router实例上 router.use()**
- **错误级别的中间件 专门捕获项目中发生的异常错误,从而防止项目异常崩溃的情况 注册在所有路由之后**

```
throw new Error("服务器内部发生了错误!")      //认为制造
// 发生错误会立即捕获
const mw = function(err,req,res,next) {
            //      ↑
}
```

- **Express内置的中间件 4.16.0**
  - **express.static 快速托管静态资源的中间件**
  - **express.json 解析JSON格式的请求体数据(有兼容性 4.16.0)**
  - **express.urlencoded 解析URL-encoded 格式的请求体数据(有兼容性 4.16.0)**

```
// body raw Json
//不配置解析表单的数据 req.body默认  = undefined
```

```
app.use(express.json())
//x-www-form-urlencoded
app.use(express.urlencoded({extended:false}))
```

- **第三方的中间件 第三方 按需下载并配置**

```
//例如之前使用的body-parser
npm install
require
app.use
```

```
app.use((req,res,next)=>{
    let str = ''
    req.on('data',chunk=> {
            str += chunk
    })
    // 使用中间件 监听data数据传入事件
})
```