导语：Express 是一个自身功能极简，完全是由路由和中间件构成的 web 开发框架：从本质上来说，一个 Express 应用就是在调用各种中间件。

路由

1. 路由的定义：

我们可以将路由定义为三个部分：

第一部分：HTTP请求的方法（get或post）

第二部分：URI路径

第三部分: 回调函数

2. 路由的实现

Express中提供了一系列函数，可以让我们很方便的实现路由：

```
app.<method>(path, callback)
语法解析：
method指的是HTTP请求方法，比如：
app.get()
app.post()
path指要通过回调函数来处理的URI地址
callback参数是应该处理该请求并把响应发回客户端的请求处理程序
```

3. 路由的实例

```
//引入express
var express = require('express')
//创建应用对象
var app = express()
//配置路由
//根路由
app.get('/',function (request,response) {
  response.send('我是根路由返回的数据--get')
})

//根路由
app.post('/',function (request,response) {
  console.log(request.body)
  response.send('我是根路由返回的数据--post')
})

//一级路由
app.get('/demo',function (request,response) {
  response.send('我是demo路由返回的数据')
})

//二级路由
app.get('/demo/test',function (request,response) {
  response.send('我是demo/test路由返回的数据')
```

```
})

//参数路由---可以动态接收参数
app.get('/meishi/:id',function (request,response) {
  console.log(request.params);
  let {id} = request.params
  response.send(`我是变化为${id}的商家`)
})

// 给服务器绑定端口监听
app.listen(3000,function (err) {
  if(!err) console.log('ok')
  else console.log(err)
})
```

request对象和response对象的常用API

1. request对象属性和方法

属性/方法	描述
request.query	获取get请求查询字符串的参数，拿到的是一个对象
request.params	获取get请求参数路由的参数，拿到的是一个对象
request.body	获取post请求体，拿到的是一个对象（要借助一个中间件—body-parser）
request.get(xxxx)	获取请求头中指定key对应的value

2. response对象属性和方法

属性/方法	描述
response.send()	给浏览器做出一个响应
response.end()	给浏览器做出一个响应（不会自动追加响应头）
response.download()	告诉浏览器下载一个文件
response.sendFile()	给浏览器发送一个文件 备注：必须传递绝对路径
response.redirect()	重定向到一个新的地址（url）
response.set(key,value)	自定义响应头内容
response.get(key)	获取响应头指定key对应的value
response.status(code)	设置响应状态码

中间件

1. 概念：

本质上就是一个函数，包含三个参数：request、response、next

2. 作用：

1) 执行任何代码。

2） 修改请求对象、响应对象。

3） 终结请求-响应循环。(让一次请求得到响应)

4) 调用堆栈中的下一个中间件或路由。

3. 分类：

1） 应用(全局)级中间件（过滤非法的请求，例如防盗链）

第一种写法：app.use((*request*,*response*,*next*)=>{})

第二种写法：使用函数定义

2） 第三方中间件，即：不是Node内置的，也不是express内置的（通过npm下载的中间件，例如body-parser）

app.use(bodyParser.urlencoded({extended:true}))

3） 内置中间件（express内部封装好的中间件）

app.use(express.urlencoded({extended:true}))

app.use(express.static('public')) //暴露静态资源

4) 路由器中间件 （Router）

备注：

1. 在express中，定义路由和中间件的时候，根据定义的顺序（代码的顺序），将定义的每一个中间件或路由， 放在一个类似于 数组的容器中，当请求过来的时候，依次从容器中取出中间件和路由，进行匹配，如果匹配成功，交由该路由或中间件处理，如果全局 中间件写在了最开始的位置，那么他就是请求的“第一扇门”。

2. 对于服务器来说，一次请求，只有一个请求对象，和一个响应对象，其他任何的request和response都是对二者的引用。

4. 中间件实例

```
const express = require('express')
//引入body-parser，用于解析post参数
//const bodyParser = require('body-parser')
const app = express()

//【第一种】使用应用级(全局)中间件------所有请求的第一扇门-------所有请求都要经过某些处理的时候用此种写法
```

```
/*app.use((request,response,next)=>{
  //response.send('我是中间件给你的响应')
  //response.test = 1 //修改请求对象
  //图片防盗链
  if(request.get('Referer')){
    let miniReferer = request.get('Referer').split('/')[2]
    if(miniReferer === 'localhost:63347'){
      next()
    }else{
      //发生了盗链
      response.sendFile(__dirname+'/public/err.png')
    }
  }else{
    next()
  }
  //next()
})*/

//【第二种】使用全局中间件的方式------更加灵活，不是第一扇门，可以在任何需要的地方使用。
function guardPic(request,response,next) {
  //防盗链
  if(request.get('Referer')){
    let miniReferer = request.get('Referer').split('/')[2]
    if(miniReferer === 'localhost:63347'){
      next()  //如果不调用next方法，下面路由将不起作用
    }else{
      //发生了盗链
      response.sendFile(__dirname+'/public/err.png')
    }
  }else{
    next()
  }
}

//使用第三方中间件bodyParser

//解析post请求请求体中所携带的urlencoded编码形式的参数为一个对象，随后挂载到request对象上
//app.use(bodyParser.urlencoded({extended: true}))

//使用express内置中间件解析post请求请求体中所携带的urlencoded编码形式的参数为一个对象，随后挂载到request对象上
app.use(express.urlencoded({extended: true}))

//使用内置中间件去暴露静态资源 ---- 一次性把你所指定的文件夹内的资源全部交出去。
app.use(express.static(__dirname+'/public'))

app.get('/',(request,response)=>{
    console.log(request.demo)
    response.send('ok')
})

app.get('/demo',(request,response)=>{
    console.log(request.demo)
    console.log(request.query)
    response.send('ok2')
})

app.get('/picture',guardPic,(request,response)=>{
```

```
  response.sendFile(__dirname+'/public/demo.jpg')
})

app.post('/test',(request,response)=>{
  console.log(request.body)
  response.send('ok')
})


app.listen(3000,function (err) {
  if(!err) console.log('ok')
  else console.log(err)
})
```

路由器

1. Router是什么?

Router 是一个完整的中间件和路由系统，也可以看做是一个小型的app对象。

2. 为什么使用Router?

为了更好的分类管理route，路由可以分为UI路由和业务路由等，可以使用路由器按类别拆分路由模块。

3. Router的使用实例

```
/*专门用于管理展示界面的UI路由*/

//引入express
const express = require('express')
//创建一个Router实例（路由器就是一个小型的app）
const router = express.Router()
//引入path模块----Node中内置的一个专门用于解决路径问题的库
let {resolve} = require('path')

//用于展示登录界面的路由，无其他任何逻辑 ----- UI路由
router.get('/login',(req,res)=>{
  let url = resolve(__dirname,'../public/login.html')
  res.sendFile(url)
})

//用于展示注册界面的路由，无其他任何逻辑 ----- UI路由
router.get('/register',(req,res)=>{
  let url = resolve(__dirname,'../public/register.html')
  res.sendFile(url)
})

module.exports = function () {
  return router
}
```

EJS模板

1. EJS是什么?

EJS是一个高效的 JavaScript 模板引擎。模板引擎是为了使用户界面与业务数据（内容）分离而产生的。简单来说，使用EJS模板引擎就能动态渲染数据。

2. EJS的使用

1) 下载安装： npm i ejs --save

2) 配置模板引擎： app.set("view engine" , "ejs")

3) 配置模板的存放目录： app.set("views","./views")

4) 在views目录下创建模板文件： xxx.ejs

5) 使用模板，通过response.render来渲染模板： response.render('模板名称', 数据对象)

3. EJS的语法

```
ejs语法：
    1.< % % >    里面能写任意js代码，但是不会向浏览器输出任何东西。
    2.< %- % >   能够将后端传递过来对象指定key所对应的value渲染页面
    3.< %= % >   能够将后端传递过来对象指定key所对应的value渲染页面（推荐使用）
```

4. EJS的使用实例

```
const express = require('express')

const app = express()
//让你的服务器知道你在用哪一个模板引擎-----配置模板引擎
app.set('view engine','ejs')
//让你的服务器知道你的模板在哪个目录下，配置模板目录(只有'./haha'可以改动)
app.set('views','./haha')

//如果在express中基于Node搭建的服务器，使用ejs无需引入。
app.get('/show',function (request,response) {
  let personArr = [
    {name:'peiqi',age:4},
    {name:'suxi',age:5},
    {name:'peideluo',age:6}
  ]
  response.render('person',{persons:personArr,a:1})
}) //创建person.ejs文件才能生效。

app.listen(3000,function (err) {
  if (!err) console.log('服务器启动成功了')
  else console.log(err)
})
```