01 服务器是如何服务前端 把前端变成一个动态网页的

```

const express = require('express')

const app = express ()

app.get('/video',(req,res)=>{
    console.log('有人访问我了/video  索要视频')
    res.send({
        name:'我是视频',
        url:'https://www.baidu.com/',

    })
})



app.get('/car',(req,res)=>{
    console.log('有人访问我了/car 汽车资讯')
    res.send({
        name:'汽车资讯',
        content:'最新新车资讯 。。。。。。。。。。。。。。。。。。。。'
    })
})

app.get('/car/benz',(req,res)=>{
    console.log('有人访问我了/car/benz 汽车资讯 之benz')
    res.send({
        name:'汽车资讯之奔驰最新讯息',
        content:'梅赛德斯-奔驰（Mercedes-Benz），德国豪华汽车品牌，汽车的发明者，被认为是世界上最成功的高档汽车品牌之一，其完美的技术水平、过硬的质量标准、推陈出新的创新能力，以及一系列经典轿跑车款式令人称道。奔驰三叉星已成为世界上最著名的汽车及品牌标志之一'
    })
})

app.get('/car/bmw',(req,res)=>{
```

```
35      console.log('有人访问我了/car/bmw 汽车资讯 之bmw')
36      res.send({
37          name:'汽车资讯之奔驰最新讯息',
38          content:' 宝马汽车，是指宝马汽车公司（Bayerische Motoren Werke AG，简称BMW）生产的汽车，主要的系列车型有1、2、3、4、5、6、7、8等系列。以生产豪华轿车、摩托车和高性能发动机而闻名于世。宝马汽车公司是世界著名的轿车公司'
39      })
40  })
41
42
43
```

02后端路由繁琐 需要配置路由表

原因：如果路由表和业务逻辑写在一起会大量冗余的面条型代码 所以必须有路由表的映射

```
1  const express = require('express')
2
3  const app = express()
4
5  const router = express.Router()
6
7  // 这是第一个路由表
8  router.get('/a',(req,res)=>{
9      res.send('我是get请求的/a')
10 })
11 //这是第二个路由表
12 router.get('/b',(req,res)=>{
13     res.send('我是get请求的/a')
14 })
15
16
17 //路由表只是类似于 wps excel那种文档 想要使用 要明确声明要用
18 app.use(router)
```

```
19
20
21
22 app.listen(8001,()=>{
23     console.log('服务器8001端口启动了')
24 })
25
26
27
28
29
```

03 express.router配置路由表的真正意义用处所在

```
1  //a.js
2  const express = require('express')
3  const goodsRouter = require('./route/goods')
4  const usersRouter = require('./route/users')
5  const app = express()
6
7  app.use( usersRouter)
8  app.use( goodsRouter)
9
10 app.listen(8000,()=>{
11     console.log('服务器启动到8000端口了')
12 })
13
14
15
16
17
18 //routes/goods.js
19 //route/goods    路由表
20
21 // 任务: 配置一个商品相关的路由表
22 const express = require('express')
23
24 //建立表
```

```
const router = express.Router()

// 进行路由表的配置
router.get('/goodsList', (req, res) => {
    res.send('获取商品信息')
  })

router.post('/addGoods', (req, res) => {
    res.send('添加商品')
  })


module.exports = router



//users.js
const express = require('express')
//建立表
const router = express.Router()

router.get('/a',(req,res)=>{
    res.send('我是get /a请求')
})

router.post('/user',(req,res)=>{
    res.send('获取用户信息')
})



module.exports = router


```

04 整合路由表出现的小小的问题

```
核心代码


// 告诉 app 使用这个路由表
// 只有以 /users 开头的请求，会使用这个表
app.use('/users', userRouter)
// 告诉 app 这个表也要使用
app.use('/goods', goodsRouter)
```

05 express 配置静态资源 向服务器请求css的服务器加载

```
1.js

const express = require('express')
const viewsRouter = require('./route/view')
const app = express()



// 挂载静态资源
 // 所有的 /public 开头的都会去到 public 文件夹下寻找内容
 // 按照你请求路径后面的内容去寻找
 app.use('/public', express.static('./public'))
 app.use('/views',viewsRouter)
```

```
14
15 app.listen(8080,()=>{
16     console.log('8080启动了')
17 })
18
19
20
21
22
23
24 /view/index.html
25 <!DOCTYPE html>
26 <html lang="en">
27 <head>
28     <meta charset="UTF-8">
29     <meta name="viewport" content="width=device-width, initial-scale=1.
0">
30     <link rel="stylesheet" href="/public/css/index.css">
31     <title>Document</title>
32 </head>
33 <body>
34     <div>
35         <h1>我是view文件脸的index.html</h1>
36         <img src="/public/1.jpg" alt="行情">
37     </div>
38 </body>
39 </html>
40
41
42
43 /route/view.js
44 const router = require('express').Router()
45 const fs = require('fs')
46
47 router.get('/index.html', (req, res) => {
48     fs.readFile('./view/index.html', 'utf8', (err, data) => {
49       if (err) return console.log(err)
50
51       // res.send() 方法如果返回的 buffer 的数据格式, 会自动下载
52       res.send(data)
53     })
```

```
  })


module.exports = router



/public/
下面一个1.jpg 一个index.css 自行配置

```

06 express 解析请求体 query

```
此例子参考 用这个测试
//http://localhost:8080/a?name=zs&password=123456
//http://localhost:8080/a?name=zs


1.js

const express = require('express')
const testRouer = require('./route/test')
 const app = express()
 app.use(testRouer)
 app.listen(8080,()=>{
     console.log('running at port 8080 !!!')
 })


 route/test.js
 // route/test.js

```

```
const router = require('express').Router()


router.get('/a',(req,res)=>{
    const {url,query} = req
    console.log(url)
    console.log(query)

    res.send({
        msg:query
    })

})


module.exports = router
```

07 重点 express 解析JSON数据

```
postman测试body

1.js

const express = require('express')
const testRouer = require('./route/test')
const bodyParser = require('body-parser')
const app = express()
app.use(bodyParser.json());
 //语义 使用urlencoded请求体
 //返回的对象是一个键值对，当extended为false的时候，键值对中的值就为'String'或'Array'形式，为true的则可为任何数据类型。

 app.use(bodyParser.urlencoded({ extended: true }));
 app.use(testRouer)
 app.listen(8080,()=>{
    console.log('running at port 8080 !!!')
```

```
})


route/test.js

const router = require('express').Router()


router.post('/a',(req,res)=>{
    // const {url,query} = req
    // console.log(url)
    // console.log(query)
    console.log(req.body)

    res.send({
        msg:req.body
    })

})


module.exports = router
```

08 node中间件 初识以及理解1

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

const app = express()

app.use(function (req,res,next) {
    console.log('time ' + Date.now())
    console.log(req.method)
    next()
})

app.get('/a',(req,res)=>{
    console.log('有人访问我了/car/bmw 汽车资讯 之bmw')
    res.send({
```

```
        name:'最新讯息',
        content:'最新讯息。。。。。。。。。。。。'
    })
})

app.listen(9999,()=>{
    console.log('9999启动了')
})

```

09 node中间件 加强处理(第三方时间处理 npm包学习使用)

```

const express = require('express')
var moment = require('moment');
const app = express()

app.use(function (req,res,next) {
    moment.locale('zh-cn');
     var time =  moment(Date.now()).format('MMMM Do YYYY, h:mm:ss a');

    console.log('time ' + time)
    console.log(req.method)
    next()
})

app.get('/a',(req,res)=>{
    console.log('有人访问我了/car/bmw 汽车资讯 之bmw')
    res.send({
        name:'最新讯息',
        content:'最新讯息。。。。。。。。。。。。'
    })
})

```

```
app.listen(9999,()=>{
    console.log('9999启动了')
})

```

10 中间件的执行顺序以及作用的深入理解

```
var express = require('express');

var app = express();

function middlewareA(req, res, next) {
    console.log('middlewareA before next()');
    const start = new Date()

    next();
     console.log('middlewareA after next()');
     const delta = new Date() - start;
     console.log(`请求耗时: ${delta}ms`);

}

function middlewareB(req, res, next) {
    console.log('middlewareB before next()');

    next();
    console.log('middlewareB after next()');

}

function middlewareC(req, res, next) {
    console.log('middlewareC before next()');

    next();
    console.log('middlewareC after next()');

}

```

```
app.use(middlewareA);
app.use(middlewareB);
app.use(middlewareC);
app.listen(3000, function () {
    console.log('listen 3000...');
});

```

项目准备工作

```
自己配置的expess 有着千奇百怪的写法 我们为了 更快速 更好的开发 我们用express脚手架
npm install -g express-generator



介绍项目开发方式
1 旧模式 前后端不分离 比如express ejs java jsp
2 新模式 以及未来趋势 前后端分离 比如 vue expesss 好用 逻辑清晰
此项目介绍前后端不分离的开发方式 express ejs 为了了解曾经的编程界的开发 了解曾经的模板引擎
页面模板是什么:在静态页中插入后端的数据变量 逻辑语句 通过后端程序执行 得到真实的页面
express --view=ejs myapp 启动脚手架 脚手架搭建起来



```

以下是这个项目的核心代码　体验下如何给模板返回数据　EJS前后端不分离的 古老开发方法　了解为主　我们主要学习express后端的node代码　前后端非分离是未来的主流 现在知识了解下返回数据的思维 以及以前曾经程序员走过的老路

```
routes/index.js

var express = require('express');
var router = express.Router();

/* GET home page. */
router.get('/', function(req, res, next) {
  res.render('index', {
    title: 'Express',
     num:123456,
     arr:['北京','上海','深圳','广州'],
     flag:true,
     html:"<h1>我是h1标签</h1>"
  });
});

module.exports = router;




ejs 引擎学习代码 views/index.ejs

<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <%- include("./header.ejs") %>
    <h1><%= title %></h1>
    <p>Welcome to <%= title %></p>
    <p>num  <%=num %></p>
```

```
    <ul>
      <%for(var i = 0; i<arr.length; i++) { %>
          <li><%=arr[i]%></li>
        <% }%>
    </ul>
    <p>
      <% if(flag){%>
          <h1>好</h1>
          <% }else {%>
        <h1>不好</h1>
        <%}%>
    </p>
    <p>
      <%=html%>
      <%-html%>
    </p>


  </body>
</html>
```

我们用这个做 前端的模板 管理系统模板 曾经很火 现在都是前后端分离 vue系列 react系列 adminLTE现在只做辅学 主要学express的后端

3. AdminLTE Star：35.5k

https://github.com/almasaeed2010/AdminLTE

AdminLTE后台管理模板也是基于Bootstrap 4.4框架以及JS / jQuery插件的。

最大的特点就是高度可定制而且容易上手使用，方便快捷。

并且支持从小型移动设备到大型台式机的多种屏幕分辨率，兼容性强。

效果地址：https://adminlte.io/themes/v3/

还好我们还有Font Awesome的中文官网：http://www.fontawesome.com.cn/

使用很简单 Easy to use

整合过的 bug少的版本AdminLTE

然后把views.zip 解压后放到 express 脚手架的 views里面

辅助核心代码 依次改掉

```
routes/index.js

var express = require('express');
var router = express.Router();
router.get('/', function(req, res, next) {
   res.render('index',{
     index:0
   })
});

module.exports = router;


app.js 导入 路由表


var indexRouter = require('./routes/index');

```

```
20
21 app.use('/', indexRouter);
22
23
24
25
26
27
28
29
30
31 第二个 routes/pro.js
32 var express = require('express');
33 var router = express.Router();
34
35 router.get('/', function(req, res, next) {
36    res.render('pro',{index:1})
37 });
38
39 module.exports = router;
40
41
42 var proRouter = require('./routes/pro');
43
44 app.use('/pro', proRouter);
45
46
47
48
```