# Node.js

Node.js本质和浏览器一样是一个JavaScript运行环境。浏览器的运行环境是V8引擎+浏览器提供的API，node.js的运行环境是V8引擎+node提供的API。说白了就是基于Chrome浏览器的v8引擎的一JavaScript个运行环境。他可以使JavaScript能够编写后端。同时要注意的是node.js这个运行环境不支持浏览器的webAPI除了console.log和计时器。全局对象是global，或者是globalThis，浏览器的是window。

1. **Buffer**

Buffer是缓冲区类似于数组，JS没有二进制数据类型，而TCP，或文件流要用二进制，所以要用buffer处理二进制。简单的说就是实现二进制的转换用的。

语法： Buffer.alloc(10)，Buffer.allocUnsafe(10)，前者清空了，后者没清空可能混一些前面程序的数据。（也就是内存空间复用了）这个比前者的优点就是快。

Buffer.from(‘hellow’)，Buffer.from([114,110,111,111])会给他转换为二进制。

就是将（）里的数据转换为2进制的意思。只不过最后给我们看的是16进制。



上面基本就是Buffer所有的常用语法了，其实核心就是转换为二进制，进制转换让他能跟上TCP协议或者是文档流的要求。

1. **关于线程和进程**

之前一直在说JS是单线程的这里就做一个解释吧，毕竟node是用来写后端的，对操作系统这块还是得懂点的。进程是程序执行的进程，而线程是进程的一部分。线程至少有一个，JavaScript这个进程就只有一个线程，就相当于一个饭馆只有一个人干所有的活。这样的话上面的异步什么的也能更好理解一点。

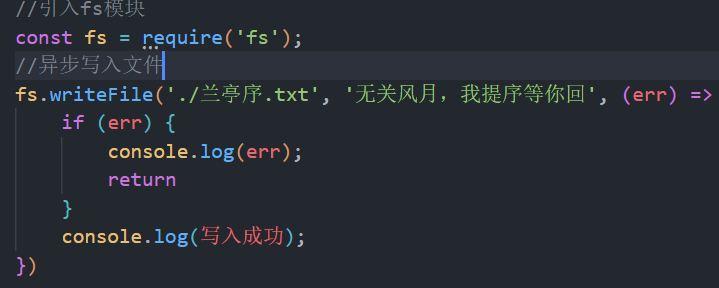
举个实际的例子：把你学编程就当做一个进程，那么node.js就是一个线程，线程进程就是这种关系。

1. **关于fs模块**

fs是一个比较高级的API和global是一个级别的，他的全称为file system，通常作用于文档的读取，写入，增删改查。主要用于操作文档。下面是fs的用法实例与一些解释。这些通常用于后端开发，而node本来就是将JS弄成后端开发的很符合node的初心。

写入文件

通常在安装软件，服务发布，写日志等方面用到这个fs的写入。





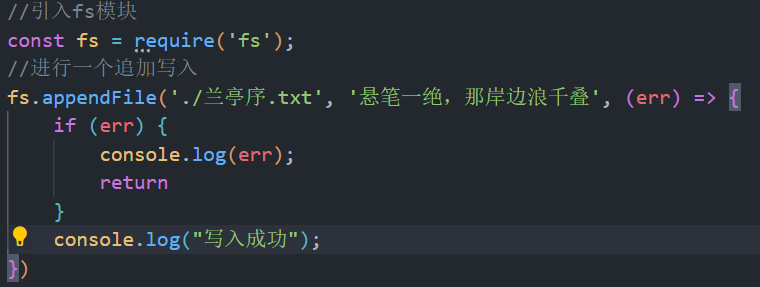
这样就写入成功了，这是异步写入，有一个回调。同步与异步问题在上面提过很多次了，无非就是一个是执行完这个才执行下面代码与这个放在任务队列，下面代码执行完再执行这个的区别。

下面是同步写入的语法，与异步写入做一个区分。



通常我们写入都是异步的，同步用的很少因为他的效率会很低。众所知JS是单线程，一个线程中执行了他就会阻塞下面代码，所以基本都用异步写入。

追加写入





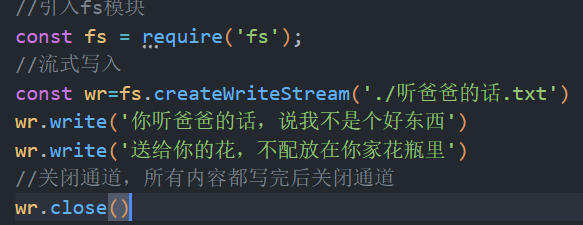
同样也有同步追加写入方法。





流式写入

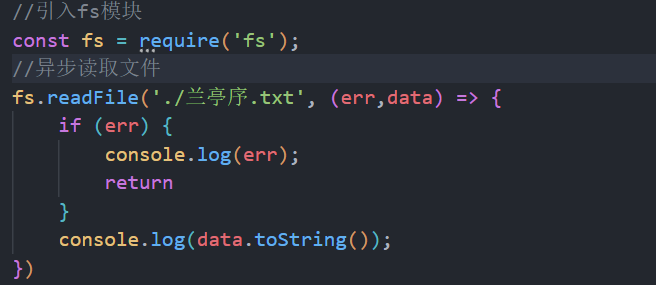
上面的写入中每次写入都要重新打开一次文件，而流式写入一次写一点，但不用重复打开，效率更高，占用内存更小，运行更快。

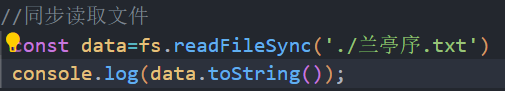




文件读取

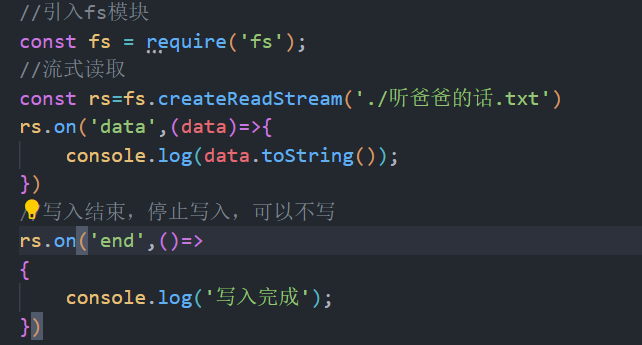
有文件写入肯定有文件读取，通常用于程序运行，查看图片视频，音频这些东西。





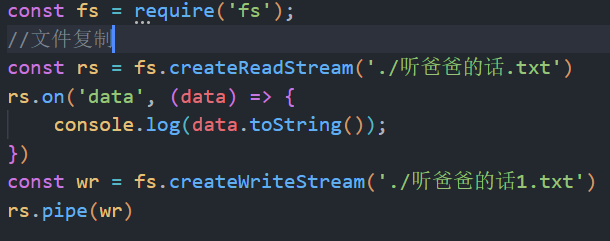
流式读取

流式读取与流式写入一样，是一点一点的读，效率更高，不用频繁打开文件。一次读取最大64K超过64K之后就会放在下一次读取。



文件复制

其实本质就是读取完之后写入，就是上面的代码，但是有一个更好的方法。



文件重命名



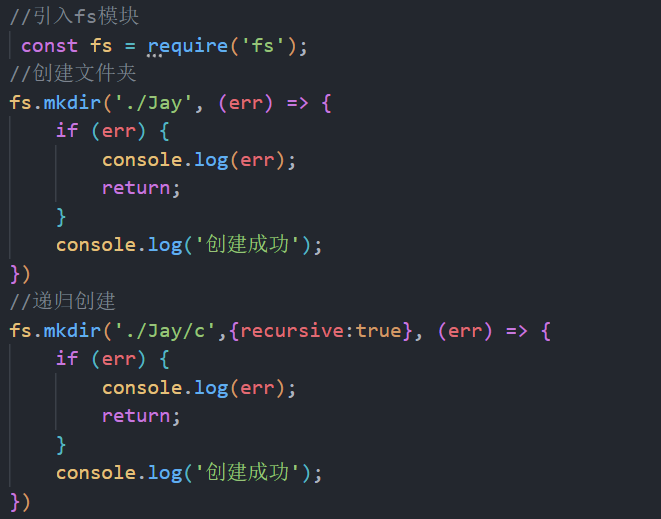
文件的移动也是用的这个原理。

文件删除

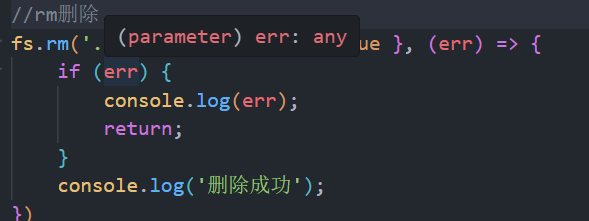


这两种删除效果是一样的。

文件夹操作



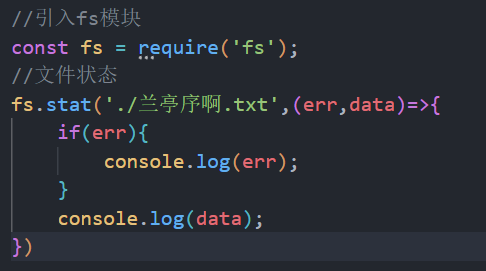




文件状态

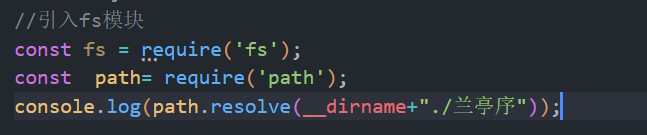
查看文件的位置，创建时间，最后打开时间等，下面这些东西就是从这个里面拿的。





文件路径path

绝对路径与相对路径统一拿到真正的绝对路径，\_\_dirname就是绝对的意思。解决路径冲突的好办法。





总结：fs的作用就是操作文件的，系统中所有文件不管还是程序还是文本都可以通过fs和他的API进行操作，读写复制等等。

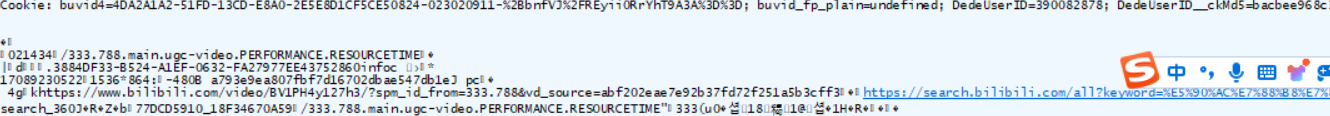
1. **HTTP协议**

HTTP协议本质就是客户端与服务端交互所达成的协议。客户端向服务端发送的叫请求，服务端向客户端发送的叫响应。下面分别对请求和响应做解释。

请求：前端向后端发送的叫请求，请求分为请求行，请求头，请求体。

下面是我用Fiddler拦截的请求，我们对他做一个解析。





**请求行**



POST为请求方法，上面Ajax那块就用了，这里就不多说了

https：这个为协议名。

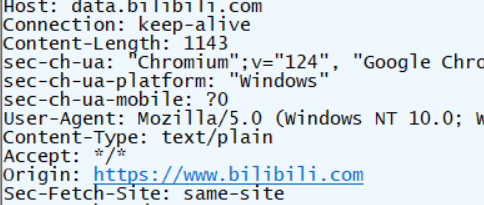
data.bilibili.com主机名

V2/log为路径

后面的一串就是查询字符串，

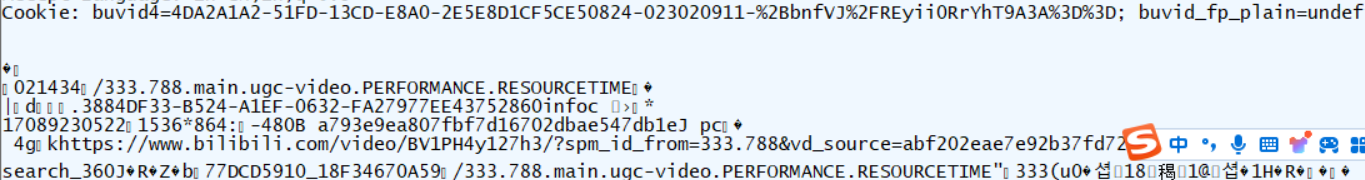
HTTP/1.1就是http协议的版本号，这里用的是1.1实际上已经更新到了3.

**请求头**



请求头是以键值对形式写的，对请求东西的细节进行了规定，这样便于后端操作，便于服务器响应。具体的可以随时到MDN查，再就是背面经的时候背一下。是描述请求体数据的数据而并不是真的数据。

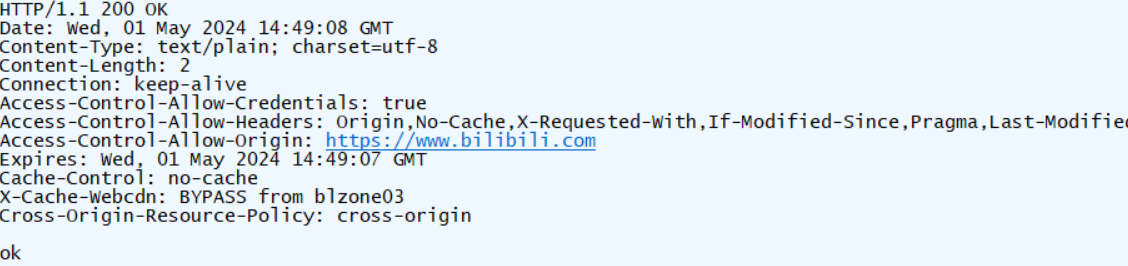
**请求体**



空行下面就是请求体，没有固定格式，与后端约定好就可以。发送的是服务器端需要处理的数据。

响应：后端向前端发送的叫响应，响应分为响应行，响应头，响应体。

下面是拦截的响应，一一介绍一下。



**响应行**



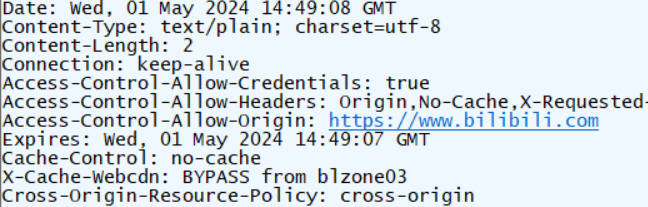
HTTP/1.1是版本号，

200是响应状态码

Ok是状态码的描述。



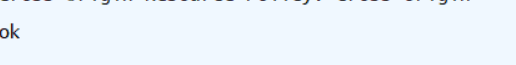
**响应头**



响应头也是键值对形式的，对响应的东西进行了一些细节操作，包含了服务端返回给客户端的一些信息，发给客户端，便于前端操作，便于客户端处理响应。是描述响应体数据的数据而并不是真的数据。

**响应体**

一般包括,HTML,CSS,JS，图片，音频，等等信息。真的要求客户端处理的信息。



响应体也没有具体要求，需要后端与前端进行一些协调。

**扩展计算机网络知识**

IP和端口：IP是设备在互联网中的地址，分为局域网，广域网和本机回环IP。本机回环IP是本设备的，局域网IP是接在路由器上的，广域网IP则是通过联通，移动和电信接在互联网上的。通过IP可以使实现设备与设备之间在互联网上的交互。端口则是IP的一个细化，具体分给某个应用程序，端口是分给设备里的程序的，IP是分给设备的。所以发送的报文一般都是IP加端口内容一起发送。

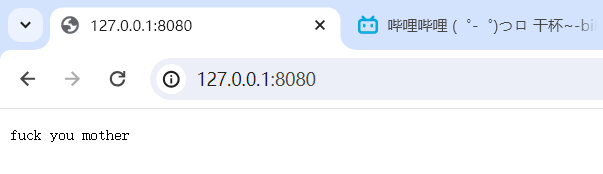
1. **关于http模块**

http模块与fs模块一样是node.js的内置模块，他主要用于实现前后端交互。

下面是用http模块创建服务端的一个小例子。



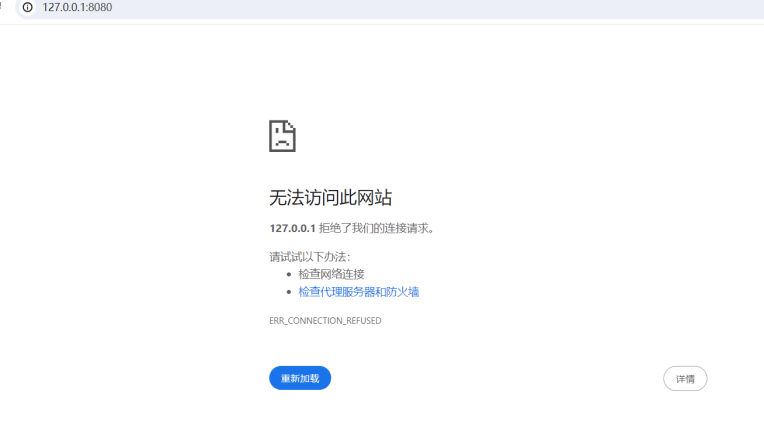
我们node ./路径名就启动成功了，然后就可以去浏览器打开服务。



这样就算是打开成功了。

这里有几个细节要注意一下：

1. **服务启动成功后可以，通过在终端命令行里ctrl+c终止，终止后我们再访问就是这样了。**

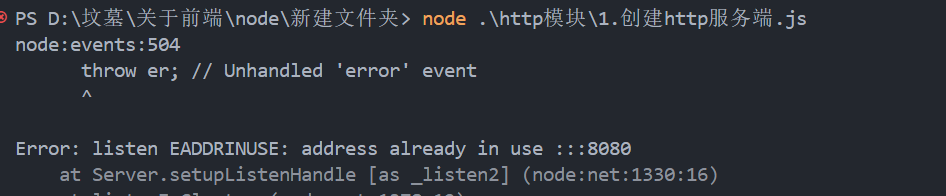


1. **服务启动后代码更新了，这时候要重启一下才能实现客户端的更新。**
2. **上面用response.end写的响应报文如果是中文就会乱码，这时候就要添加一个响应头。**

****

**这样就告诉了浏览器端响应报文的特性，解决乱码问题。**

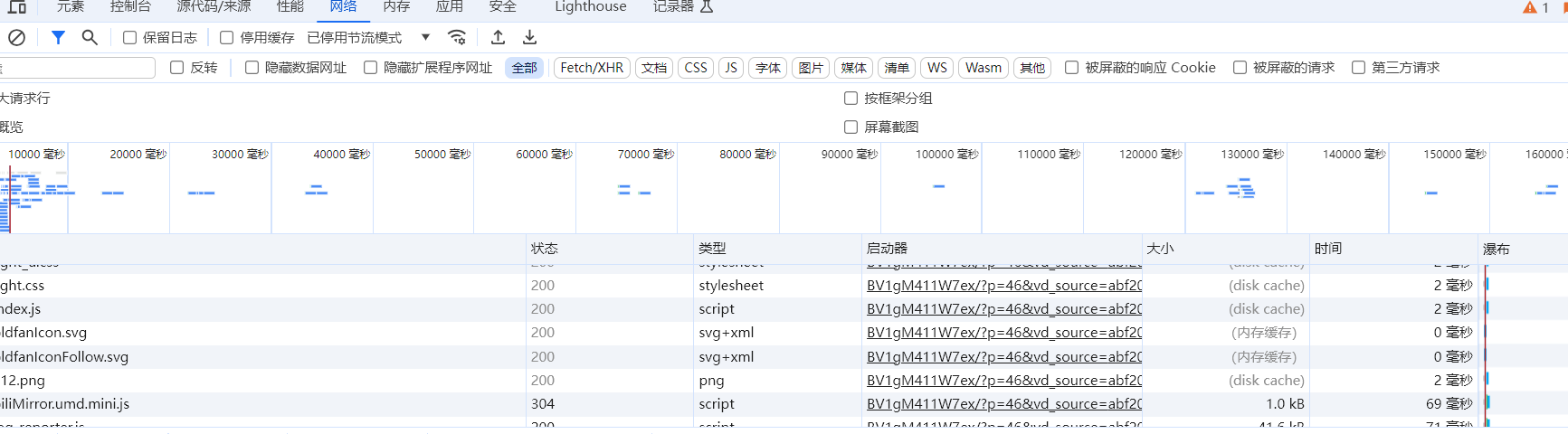
1. **如果我们已经在一个端口启用了一次服务，再次启动是无法在这个端口启动的。这时候有两种解决办法，第一种是终止之前打开的端口，另一种就是换一个端口。**



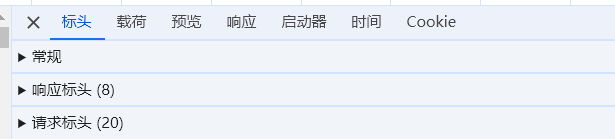
**上面是端口被占用的报错，另外补充一点。http协议有默认端口。80,8080,9000等等。用这些端口默认就不加端口号了。百度就是用这样的方法。**

**通过浏览器查看响应报文和请求报文**

响应报文和请求报文都可以通过浏览器查看，f12后在控制台的网络里面按ctrl+R就能看到

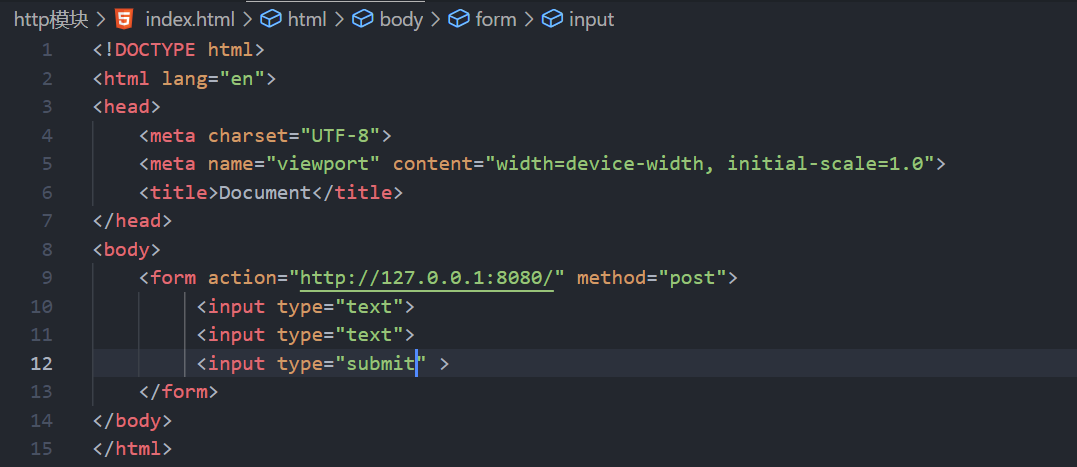


这就是请求和响应，我们随便点开一个



常规一般是请求行的内容，响应标头和请求标头，分别是请求头和响应头的内容。

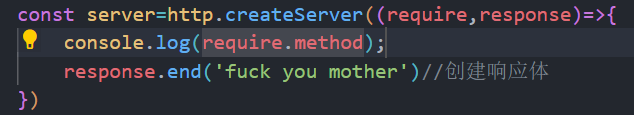
响应体点击响应就可以查看，请求体看载荷就行。



只有post方法有请求体，我们的表单元素就是一个请求体的很好载体。上面的就是一种带请求体的方法。还有一种是Ajax的请求，这里不细说了。

**获得请求头请求体的方法：**

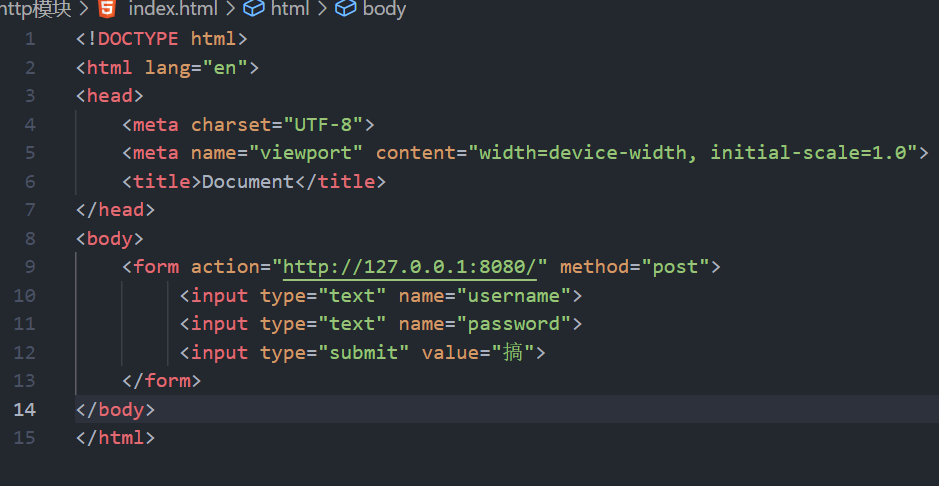
这些就是方法不细细一个个写了，注意写在我们创建的服务里面就行了。如下面这样。



获取请求体的方式有些类似于我们上面的流式读取，具体如下。

，

一般get请求是没有请求体的所以我们配合表单元素来实现。

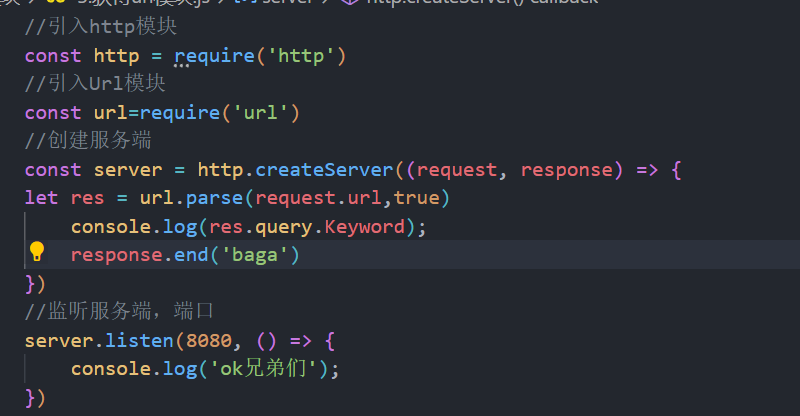


通过这个表单就可以实现请求体的传输。

**获得路径和查询参数的方法：**

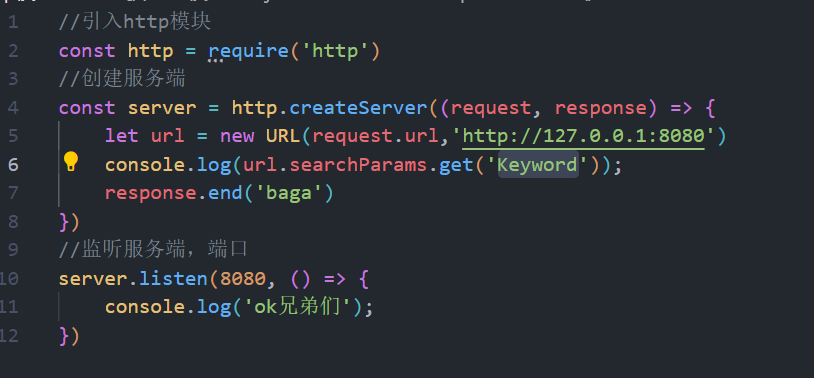
上面的获得请求地址上面是有获得URL地址的方式的，但是获得的都是混合起来的地址，不方便操作。但不是不能操作，用split这种方法分开了也是能操作的。但是开发讲究效率所以提供两种方法操作。

1.引入url模块，通过URL模块的parse方法解析解析出路径，代码如下。



Parse参数里的true其实表示的是query是对象的形式，如果不写query这个查询参数就是一个字符串形式，也就不好调用了。写了之后就可以直接用.来调用对象属性。

1. 通过URL实例化对象解决：代码如下



url会自动将URL解析成对象的形式，但有一点要注意URL中的 searchParams是map对象所以拿到他的内容需要用get方法。很显然。

Map对象是ES6提供的一个可迭代对象。具体的去看ES6就行了，这里就不细说了。

下面用一个简单的例子来实现以下http请求。

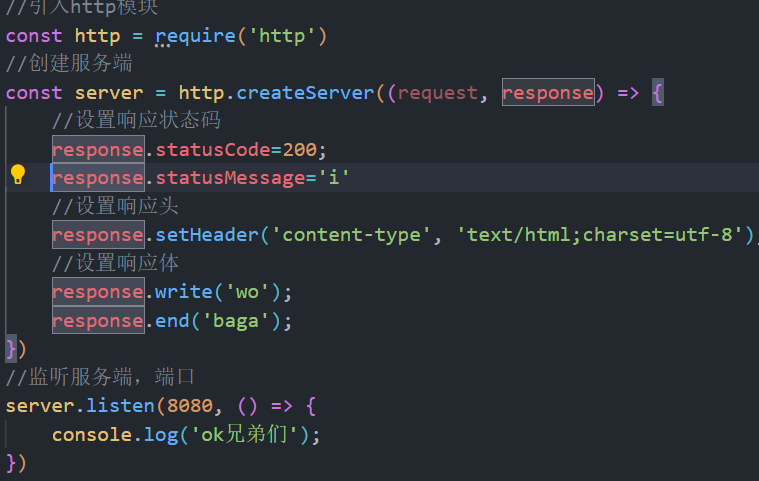


代码如下：

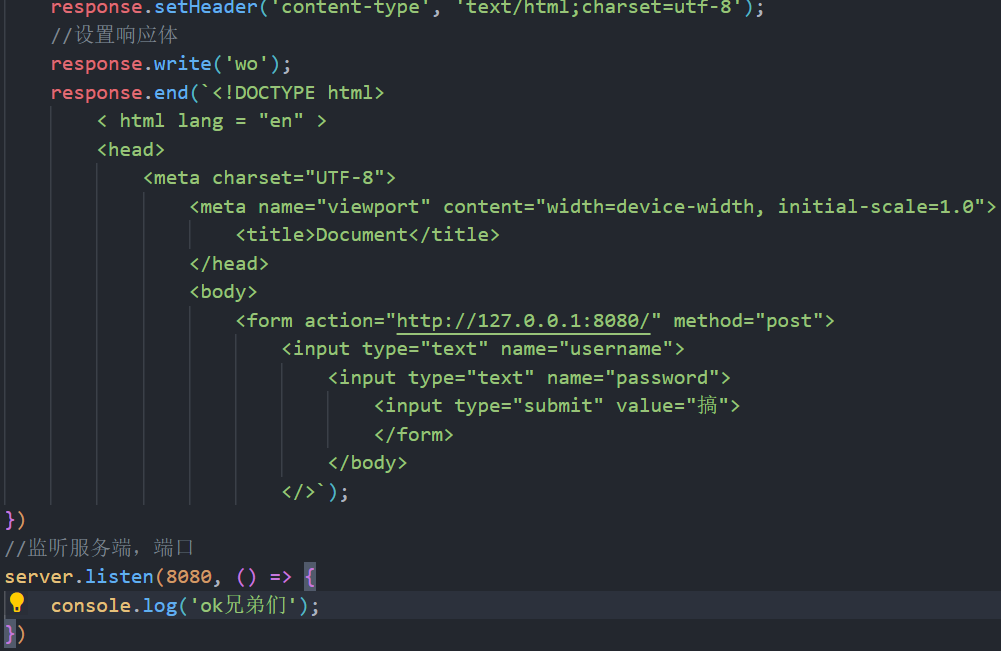


本质其实就是一个通过路径来设置响应体，并没有什么特别的地方。就是结合了一下上面的语法。像获得路径，请求方式之类的。

**用http设置响应**



注意点就是write和end这里，end不能写两个，end之后就不能再写入了。如果硬要写就会报错。End前写write会将两个叠加起来。通常写了write就不会写end了。

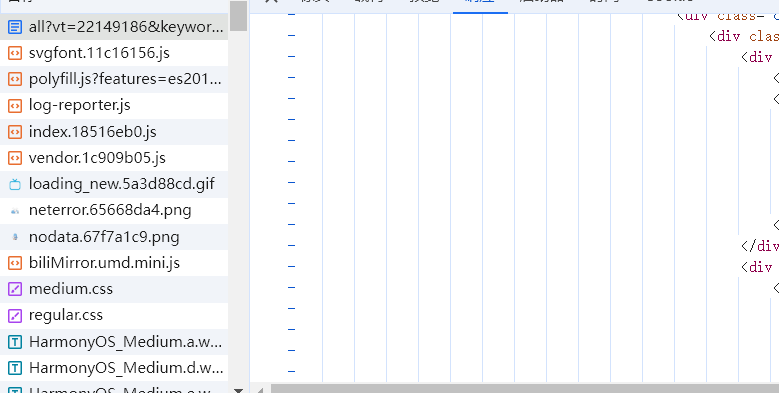


这样写响应太多且不会给你报错写错了也不知道，为了简单省事。我们选择直接用fs模块从外部引入。这就和上面的fs联系起来了。

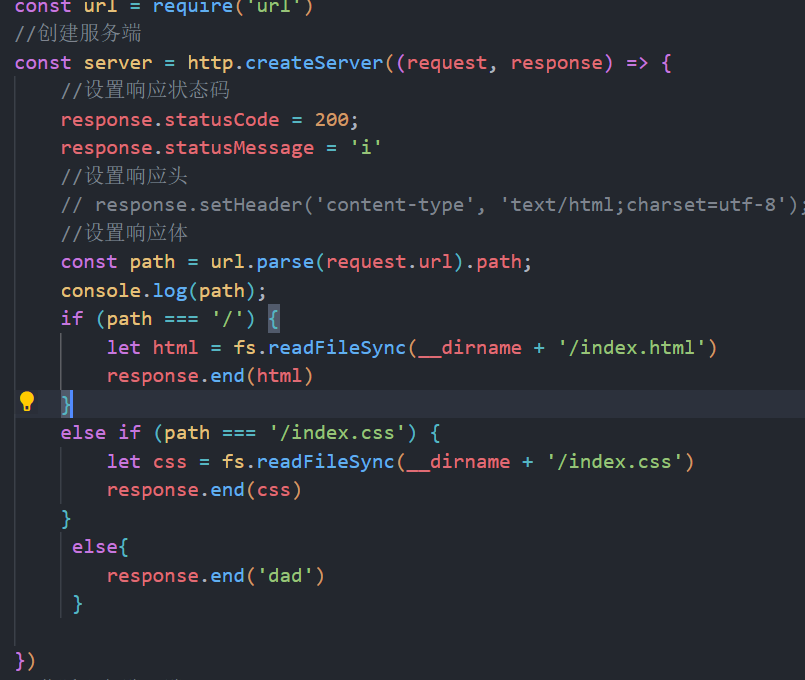


**网页资源请求与响应的过程：**

这个就是我们一般上网的请求和响应的顺序就是拿到报文一点点请求，看见css请求css看见JS请求js看见文件请求文件，这个过程是异步的，以我们的bilibili为例。

请求的就是这些东西。没什么好说的。

基于上面的这个有一个很显著的问题。请求是由我们http的createserver的回调执行的，我们每次执行的都是HTML，也就导致了样式和script从外部引入时无法加载，这时候就需要根据路径来分别对HTML，css和JS进行请求。代码如下



这样写一个两个可以，多了之后操作复杂程度必将爆炸，所以我们就找寻其中的规律，发现他的请求绝对路径\_\_dirname＋pathname进行一个拼接就是他的请求路径。代码如下：可以省去if，else判断。



其实这些都是静态资源，静态资源就是项目上线后一般不怎么变的资源，这里的图片，html,css，js都是这个意思，与之对应的是动态资源，动态资源是随时刷新的。像新浪上面的新闻就是动态资源。

其实上面的pathname之前的部分，全部都是跟目录。跟目录是资源的一个父亲，就好比说VScode的liveserver插件。打开的跟目录就是通过VScode打开的文件夹



大概就是这样的。应该很容易就看出来跟目录就是打开的文件夹，显示出来的都是require.url，请求的url，也就是pathname。

**关于URL路径**

都说道这了就顺便说说路径吧，url路径分为相对路径和绝对路径。

绝对路径：项目中比较常用。



相对路径：比较不好用，因为他会随着查询参数变化而变化，具体在vue中的路由那块再细说。

相对路径：我们练习用这个，用他的时候我们通常和当前网页的协议，域名，端口URL进行一个计算得到当前的路径。

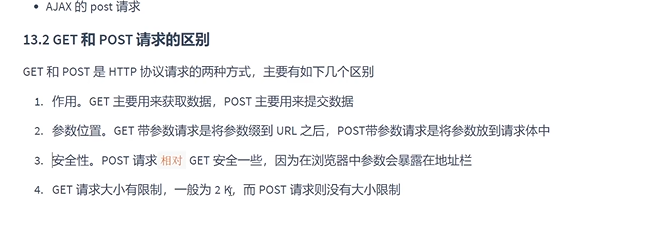


URL通常用于一下场景。



关于协议的细节：我们经常会遇到乱码问题，媒体类型（mime）等问题。通常用content-type来这个请求头来解决这些问题，不细说了。面试时候背背得了，还有一个是请求错误的问题，我们会通过请求的statusCode根据不同错误设置响应状态码，这个node的官网有这个的。有一个专门的error模块解释这些错误。根据介绍设置状态码就行了。

最后出了Ajax和form的post请求外剩下的基本都是get请求，二者的区别如下：



**关于模块化**

提高代码复用性

提高代码可维护性

防止变量命名冲突

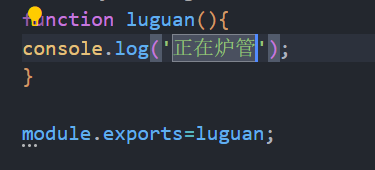
与vue的组件化是一个意思。之前

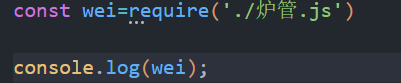
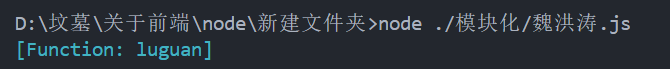
语法为：

导出：module.export={}，exports.value=’’以对象形式导出的。

导入：require(路径)

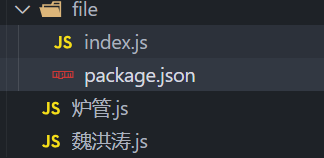
下面是一个例子：

创建了一个撸管方法用module.exports导出暴露。再用require通过相对路径引入，与内置模块进行一个区分，内置模块直接引入就行了不用./，../这种相对路径。引用代码如下

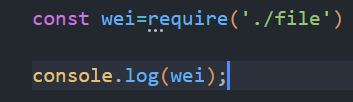
，require用的是相对路径但是有一点是要注意的./不能不写但是后面的后缀可以不写，他会给你默认了，如果名字一样那就默认先认js再认JSON，所以实际开发中还是加着比较好不然重名很难维护。

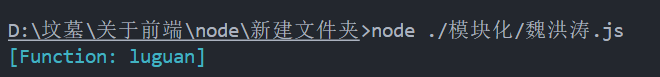
这样就引入成功了。其实还有一种暴露数据的方法就是exports.value=值，其实mode.export=exports={}是一个对象形式，这样就很清除明了了。

如果require导入的是一个文件夹那就是另一个故事了，**他会先去找package.json文件下的main里面的内容，如果没有那就会再去找index.js和index.json，如果都没有就会报错，下面是代码示例：**









这里文件夹下有package.json所以就找到他里面main的文件并导入了，在控制台打印我们require的结果也是正常的，能够看见我们的撸管方法。

当没有package.json或者package.json里面的main不正常，不存在时，就会去找index.js和index.json这里就不演示了。很简单的道理。

**Common.js**

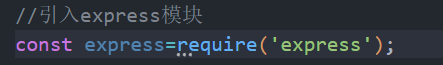
这个其实就是一个规范，里面提供了module.exports,exports，require等方法，其实他与node的关系可以理解为JS和ECMAscript之间的关系，一个是规范。一个是负责执行这个规范的。

express框架

express框架是node.js的一个框架，他是基于http模块的封装，提供了比http更方便的方法。用它比写原生的http会方便许多下面来记录一下express框架。



我们通过npm包安装一下express框架。



通过require将express引入文件中。。。。。

下面是我用express写的简单的服务：里面写了express路由的常用方法：get，post为请求，all为两个请求都能用。\*表示所有，就是没有的时候响应的是\*。这里还有一个细节，express自动生成了mime模型，所以即使我们写汉语也不会出现乱码的情况。



这里要介绍一个路由的概念，其实我们之前用原生的http模块时就写过路由，那个路由里我们先开始用if else判断。之后简化了用路径拼接，虽然简单了但没完全简单。Express框架直接简化到了极致。就像上面写的那样。那么路由是什么呢，其实就是我们浏览器IP：端口/路径这个格式的路径，这里的login就是路由。其实就是

URL里面的load，我们通过不同的load实现了不同页面的跳转，这就是路由。形象点说就是一个分配位置的东西，一个IP里面存了很多东西，路由就相当于导航，你给他一个路径，路由就会请求你想要路由里的东西。路由不止在这里会用即使到vue的工程化里面也是一个很重要的东西。所以务必弄懂。

关于请求报文

express里面获得请求报文的方法与原生http模块是兼容的也就是原生http的获得报文方法在express里面都能用，另外他还有自己的一些更方便的方法来拿到报文内容。

这是express提供的更方便的内容。

其实一个大项目中的路由是很多的。以京东为例，他每个页面都有一个特定的路由，每个商品都有路由，这时候其实就是路由参数发力了。下面就是路由参数的例子。



不要误会:id不是主体，主体是：后面的值可以随便写，写什么parms查什么就行了。

**关于响应设置**

express在兼容原生http的响应写法的基础上提供了很多响应设置方法：最提供方便的一点便是他可以支持链式调用，经常写代码的朋友都知道链式调用非常的方便我们不必一直在前面写对象，最典型的链式调用可以类比一下promise的链式调用。

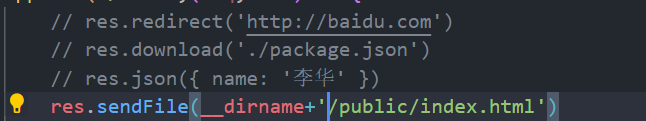
其中status()这个替代的是statusCode这个东西，set替代的是setHeader这个东西。Send替代的是end。另！他还有其他的一些特别的响应设置。

其中：redirect：是指的重定向。

Download：指的是下载响应，我们的所有下载都是通过download这个响应实现的。

JSON：我们用这个方法响应一个对象他会自动转义为JSON这个方法也有典型的，在我们的openlayer里的一些地图参数如果你用浏览器查询，以百度为例，你去单独查他的城市参数，他就是以JSON类型响应的。就是可能回犯密集恐惧症，后面用到再说。

最后是一个sendFile他与send非常类似，区别在于他可以理解成一个fs的read的封装他只需要一个绝对路径就可以直接把路径文件响应给客户端。这是上面的几种响应的语法设置。



下面是我在网上截的一个大概的表。



**关于中间件**

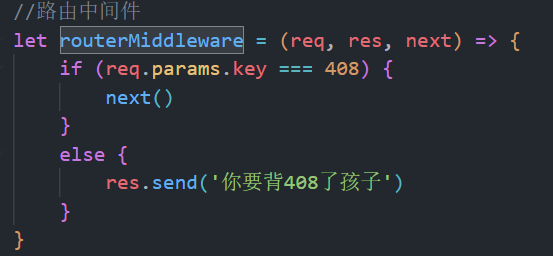
中间件的话他和我们vue的一个全局路由前置导航守卫是非常类似的。但不能说他和全局路由前置导航守卫是一个东西。只能说他的一个思路很像。他的一个开发初心其实就是为了使我们浏览器查询更加有序。可以说是起到了一个过滤作用。就好比是我们去车站。进车站要刷一次身份证，上车也要刷一次身份证。前者可类比于全局中间件，就是任何经过我们路由的东西都要进行这个。而后者，只是对特定的路由进行一个这个中间件的过滤。他也是可以访问我们的请求（req）和响应(res)。其实目的就是函数封装参数检查代码以此过滤。因此造成的必将是语法的不同。

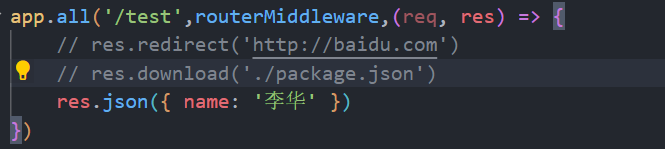
全局中间件：**代码示例如下：这里实现了一个简单请求日志。这也是很常见的东西他记录了网站的访问，当网站被攻击，会留下IP和访问方式。警察叔叔就是通过这个抓人的。**



以上是一个全局中间件的代码示例，其主要业务逻辑是封装函数，通过访问我们路由的req和res对我们的路由进行一个过滤，这个过滤的意思包含但不限于删除，可以类比于数组的一个map方法。只能所是加工。加工完毕后，调用next来进行下面路由的一个读取。

路由中间件：路由中间件来说的话和全局中间件是非常的相似的，但在部分地方也有一些差异，下面是一个路由中间件的代码示范。

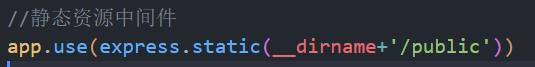




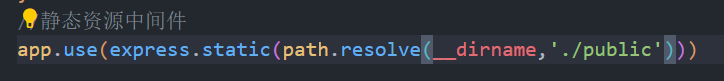
以上代码为路由中间件的用法。与全局中间件的区别在于next的位置以及使用方法，全局中间件用的是app.use(中间件名称)。而路由中间件跟他的名字一样直接插入我们要管的路由之中了。其中还有一个明显的顺序问题 。

静态资源中间件

静态资源中间件的用法可以类似于全局中间件，示例代码如下：



他的原理，可以理解为把IP地址换成资源的跟目录，我们可以根据根目录的结构来访问我们的静态资源。如图片，CSS，JavaScript,视频，音频等等吧。他用的是express的一个内置的函数static。后面跟的是绝对路径。既然是绝对路径path.resolve依然可用。

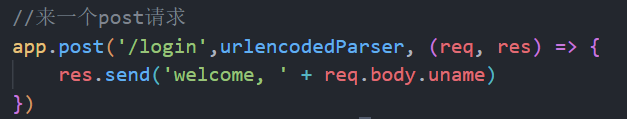


其实默认的静态资源他都是index.html。他启动时默认找的也是index.html

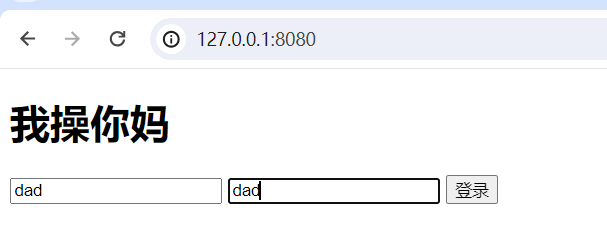
获得请求体数据

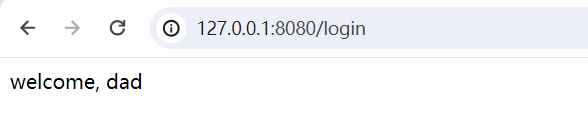
请求头请求行都在上面提供了获取方法，但请求体的获取有点特别，我们express获取需要一个npm包来实现，body-parser。我们先npm安装body-parser这个插件。





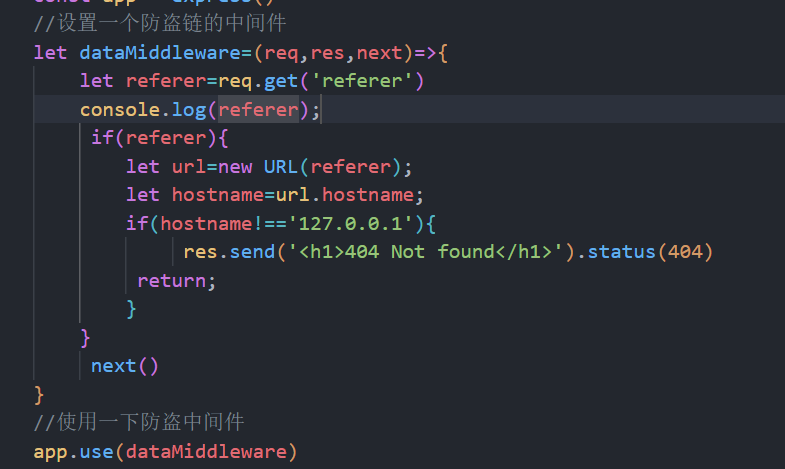
上面代码我们用表单发送了post请求，经过中间件后就可以通过这个方法拿到请求体。



这是我们的表单，点击登录提交后他会提交给login这个路由并response响应我们的请求体也就是账号密码内容，登录就是通过这个方法实现的。。

关于防盗链

他的初心其实就是为了防止，Ip内部的数据被盗用，导致服务器的负载变大，运行效率降低的问题。其实本质就是一个响应头referer，我们通过这个响应头配合中间件，就可以实现防盗链效果。下面是防盗链的代码实例。

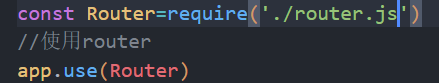


其原理就是通过referer这个参数拿到host，通过host判断Ip是不是需要的Ip，如果是就可以请求，不是就不能请求。就是这个原理。

路由模块化

模块化的思想一样可以用到路由里。这里我们用路由模块化把代码放到一个模块里，再通过引入模块实现原来的效果。代码如下：



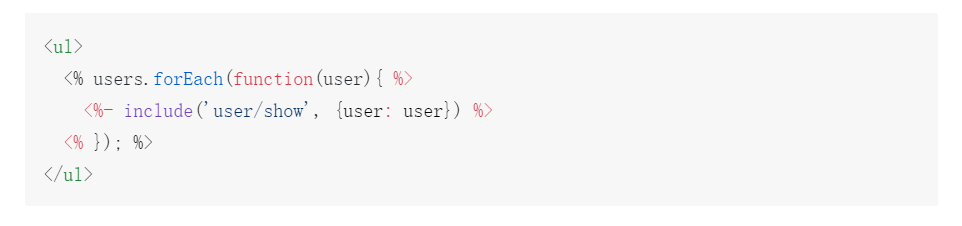


这里把部分代码放到了模块里并导出了。然后在index.js里面引入了，当中间件使用。就可以正常使用了。在app.use这里可以加个路由前缀，每个路由会默认加上去，如app.use(“/home”,Router)这样就在所有的路由前都加了/home这个前缀

介绍一下模板引擎：

这个东西怎么说呢，这个模板引擎主要是解耦合用的，也就是说将JS代码和HTML代码完全分离开而不是用${}模板字符串直接进行一个写。但是现在多是前后端分离所以不怎么用了。他用到了一个ejs包。我们通过这个包可以实现ejs实现解耦合。具体看npm的文档介绍就行了。其实现在很少用了，现在都是前后端分离，很少有这种问题。下面是我从npm包的文档里找的一些基本用法。到时候用到什么去包里面cv就行了。

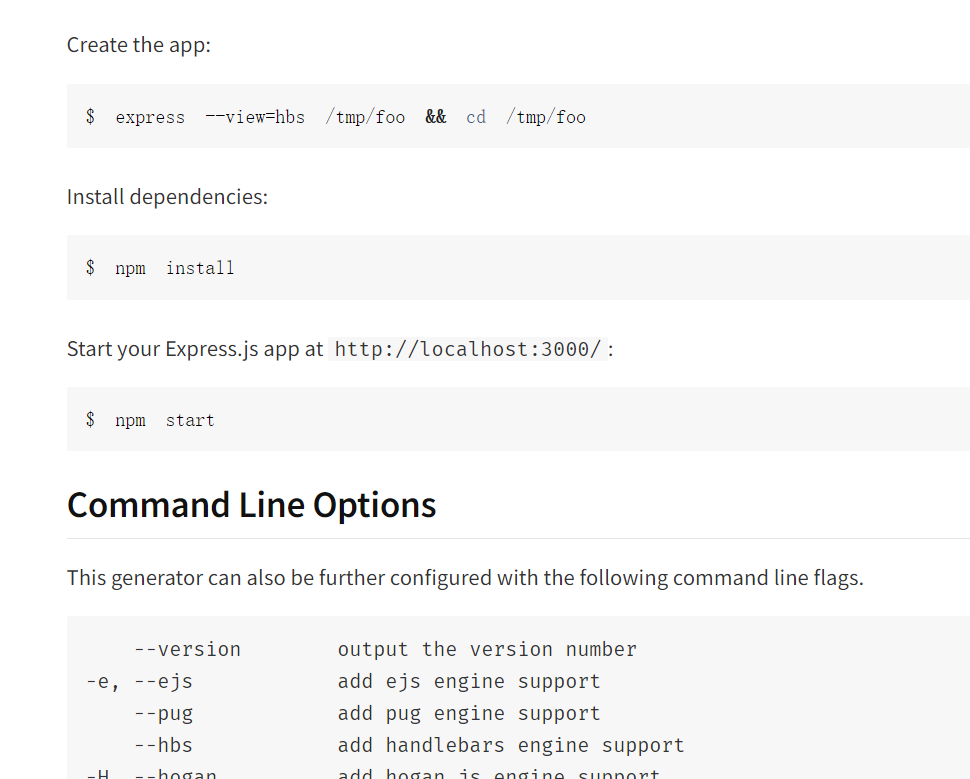




其实他的语法就是把JS代码通过<% %>这个东西包裹起来。然后就可以通过他提供的render方法来解耦合。在express里面使用按照以下方法使用就行了。

配置完毕后用res.render就能正常使用。

其实通过express-generator这个npm包能实现express的快速构建，能省去很多事。具体看包的文档，文档会告诉你怎么做。通过这个其实就实现了。



。最后文件的一个上传需要解析请求体，我们用formidable这个npm包，具体去npm文档里找就能看。

关于接口的问题

作者李某在路由与接口这个问题上纠结了很久。其实就是分不清路由和接口的一个区别，这个问题。后来又是查文章又是问大佬，总算是搞明白了。

我们用Ajax请求的url其实就是接口的一个端点，也可以理解成是接口的一部分。接口其实是前后端交互的一个桥梁。我们后端暴露数据给前端，前端可以通过Ajax请求拿到后端暴露的数据，这其实就是接口的工作原理。

而路由虽然与接口很像，但他其实是一个前端页面跳转的，他决定的更多是前端渲染什么样的数据的问题，可以理解为在前端页面的跳转。

二者的差别就在于接口是后端写的，用于给前端提供数据，或是处理前端传过来的数据。

是一个前后端的桥梁，主要用于实现前后端的交互。路由是前端写的主要是在各种页面文件之间进行跳转。只能给自己玩，和后端没有关系。

Restful其实就是一个接口规范：其url的路径表示资源，且不能有动词。操作资源与http请求方法对应。结果与响应状态码对应。

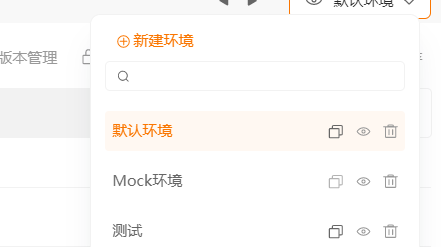


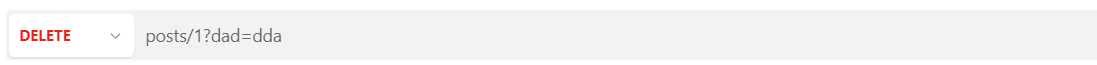
前端可以用json-server这个npm包npm安装后将文件夹在集成终端中打开然后--watch db.json

或者mock模拟接口，具体看文档就行了。没有什么难度，就是一个调包。

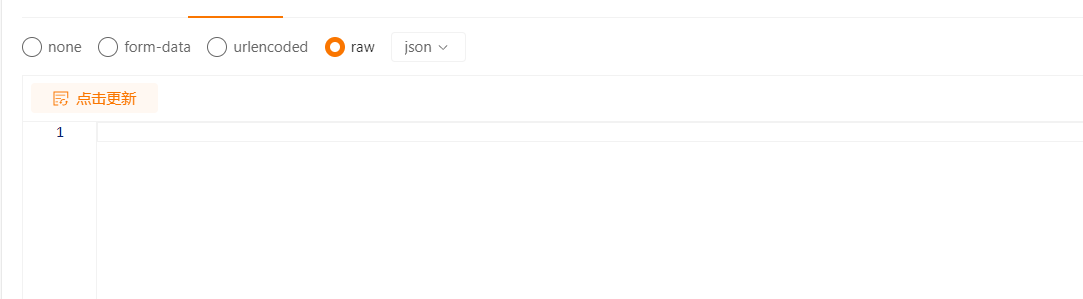
关于接口测试工具

接口好不好用测试过了才知道，这个通常是后端干的，我们用apipost或者postman进行测试，同时还可以通过这俩软件弄出来接口文档，下面操作一下。

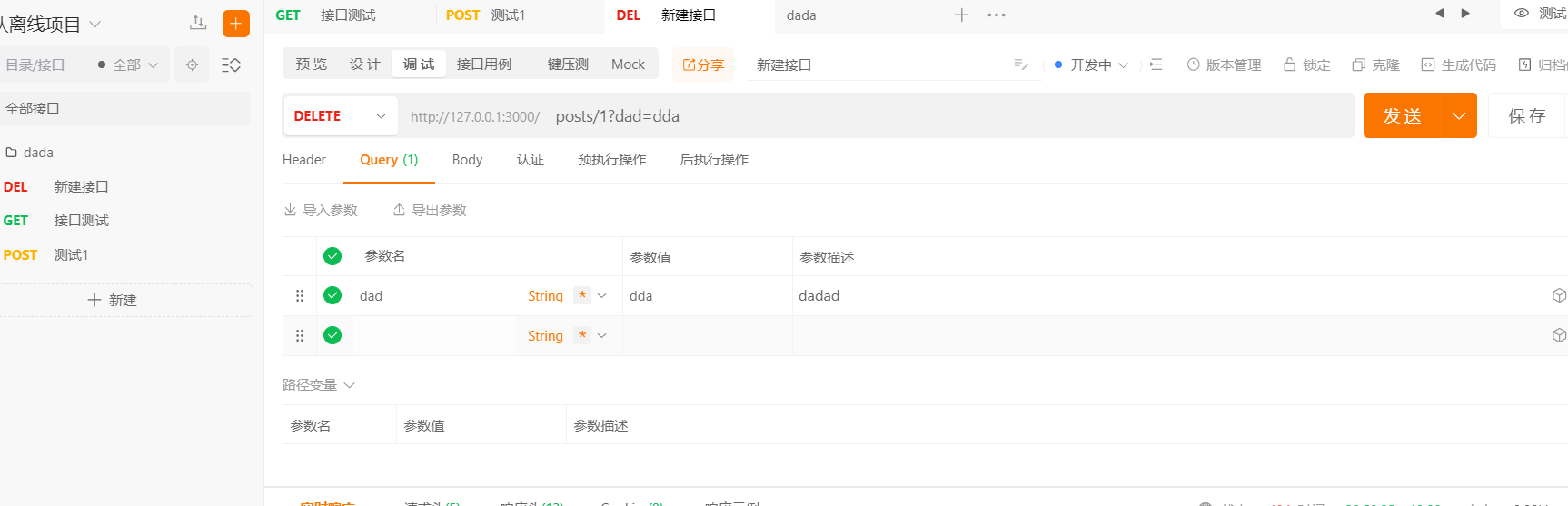
默认环境配置可以帮我们少打接口前面的协议，域名和端口一类的东西。



这里是请求方法，我们根据请求方法可以配合请求体实现增删改查。



这里可以设置请求体。然后再是一个建文件夹共用请求体等方法，最后就是分享api接口文档了。这个是后端学的，前端了解一下能用就行了。



会话控制

会话控制的来源是解决HTTP无状态的一个问题。那么什么是无状态呢。就是当你浏览器端发起请求之后客户端不知道是谁发的请求，不知道用户的信息只能一直返回值。举一个具体例子：我充了会员，另一个人没充这时候如果不用会话控制，服务器就无法分别我和另一个人谁没充会员。又或者说我设置了一个购物车，另一个也设置了一个购物车。如果都发给服务器，没有会话控制的情况下，购物车是无法区分哪个购物车是我的，哪个购物车是别人的。这样就使服务乱了套了。所以就有了会话控制技术。

常见的会话控制有三种：

Cookie

Session

Token

下面意义进行解释。

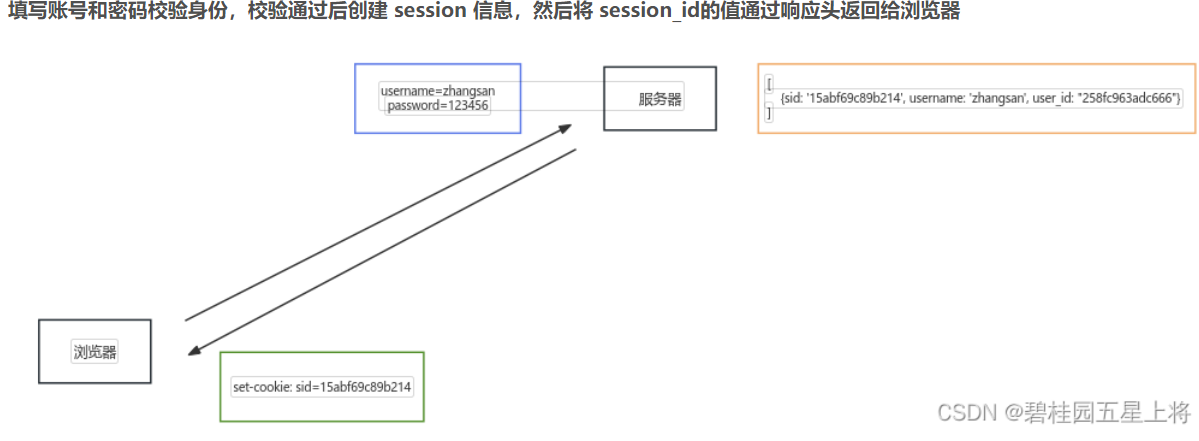
Cookie:小饼干，是存储在客户端的数据。运行原理为我们账号密码给了服务器，服务器校验没问题就会在响应头里用set-cookie属性来分发cookie到浏览器，浏览器会实现一个缓存，之后每次请求都会以cookie这个请求头的方式携带cookie进行请求。





另外操作cookie用的是cookie-parser这个npm包。

Session：主要运行在服务器端。运行原理为，我们把信息提交给服务器。服务器校验没问题会在服务器端生成一个用户对象。每个生成的对象都带有一个唯一ID。之后会以set-cookie这个响应头的方法把这个唯一ID返回给浏览器端，浏览器端储存cookie。之后再带着cookie请求服务器根据cookie就能找到对应的ID从而找的对应的用户，对用户的数据进行一个操作。





另外，操作session用的是express-session这个npm包。

cookie 和 session 的区别主要有如下几点：

1. 存在的位置

cookie：浏览器端

session：服务端

2. 安全性

cookie 是以明文的方式存放在客户端的，安全性相对较低

session 存放于服务器中，所以安全性相对较好

3. 网络传输量

cookie 设置内容过多会增大报文体积， 会影响传输效率

session 数据存储在服务器，只是通过 cookie 传递 id，所以不影响传输效率

4. 存储限制

浏览器限制单个 cookie 保存的数据不能超过4K，且单个域名下的存储数量也有限制

session 数据存储在服务器中，所以没有这些限制

Token：主要用在移动端里。Token也是存在浏览器端，但他与cookie的一个根本区别就是cookie是你有一个cookie后他就会自动添加cookie。而token是要手动添加的。运行原理依然是，我们把用户信息给服务器校验，校验成功后他会以响应体的形式返回加密过的token。之后我们要进行一些需要token的参数携带token这个请求头向服务器请求就行了。

