# 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。

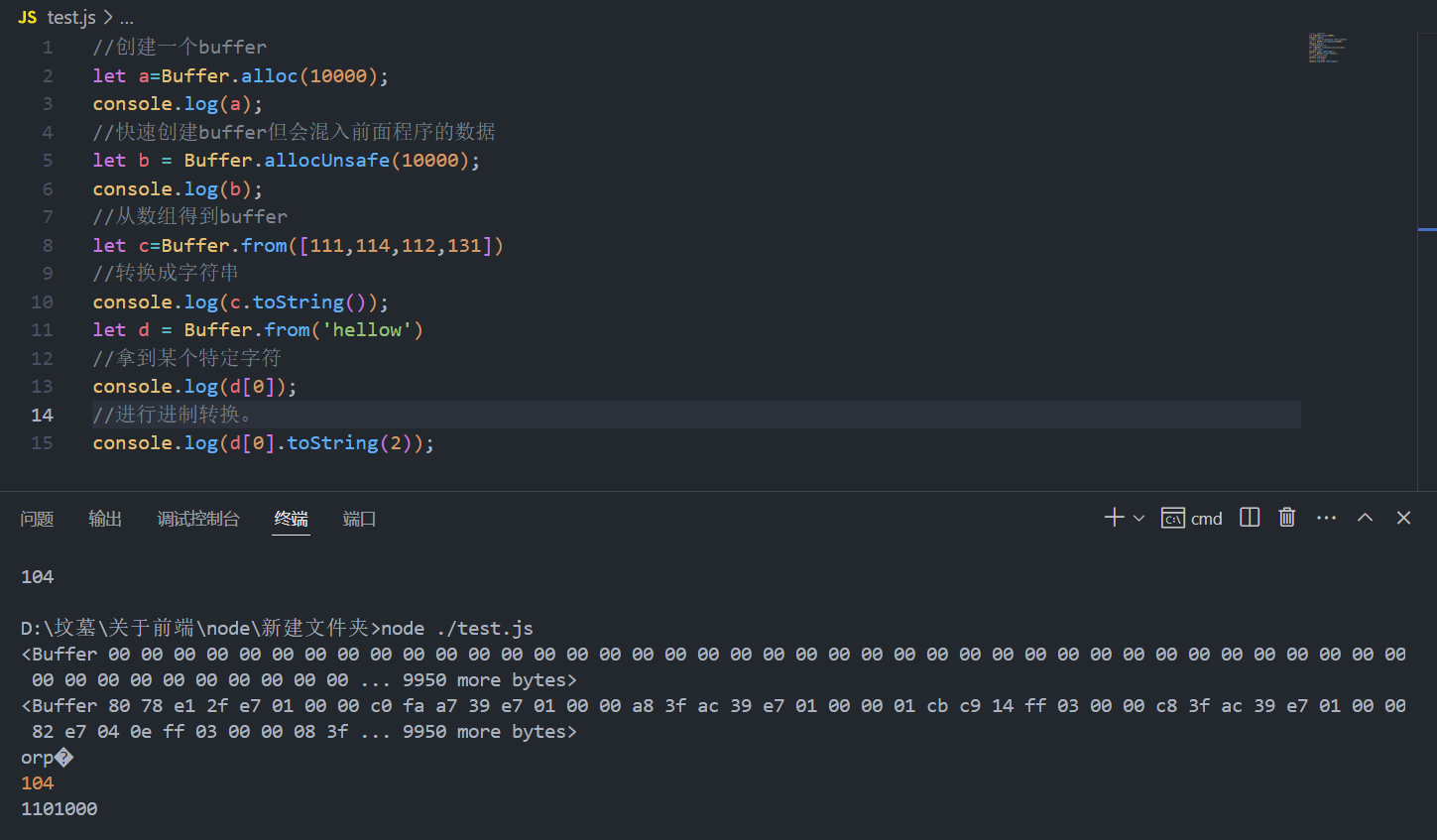
1. **Buffer**

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。

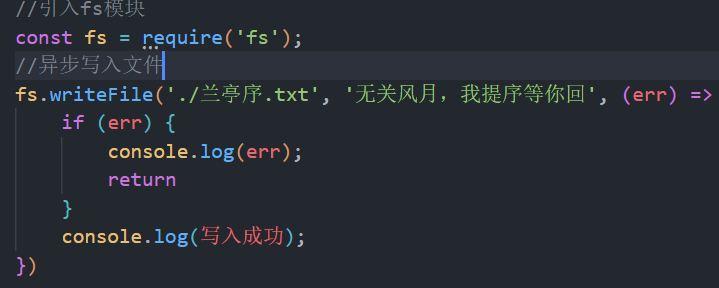
举个实际的例子：把你学编程就当做一个进程，那么node.js就是一个线程，线程进程就是这种关系。

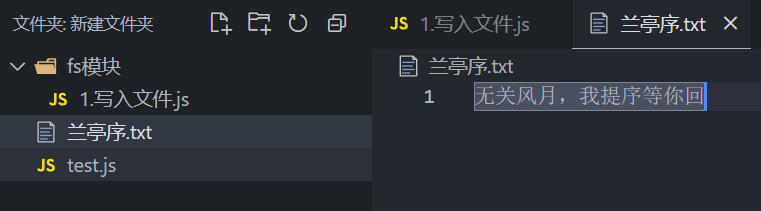
1. **关于fs模块**

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。





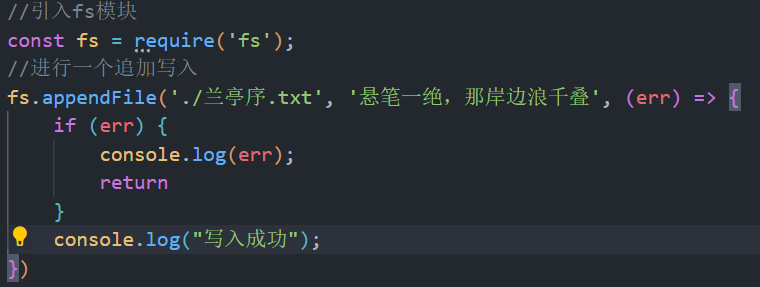
这样就写入成功了，这是异步写入，有一个回调。同步与异步问题在上面提过很多次了，无非就是一个是执行完这个才执行下面代码与这个放在任务队列，下面代码执行完再执行这个的区别。

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





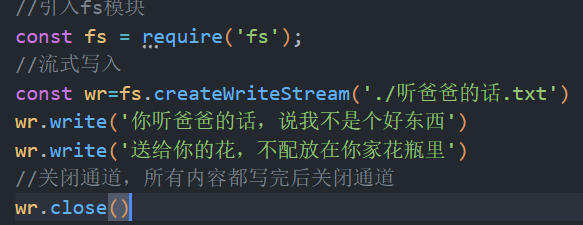
同样也有同步追加写入方法。





流式写入

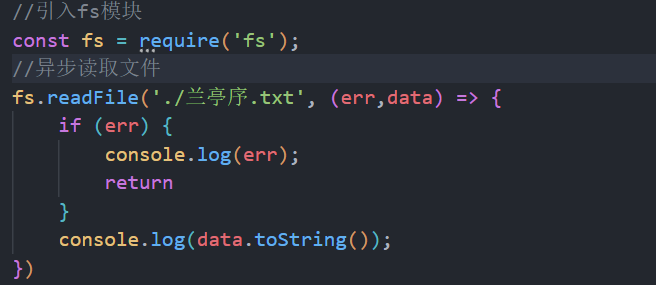
上面的写入中每次写入都要重新打开一次文件，而流式写入一次写一点，但不用重复打开，效率更高，占用内存更小，运行更快。

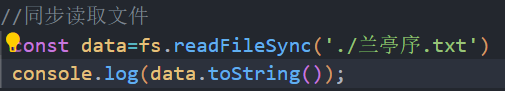




文件读取

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。





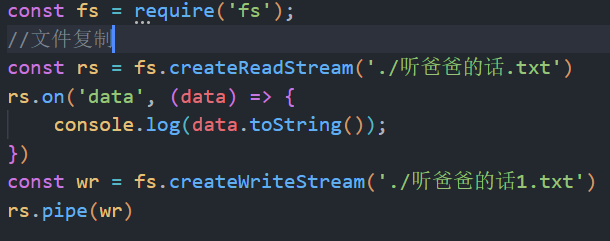
流式读取

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。

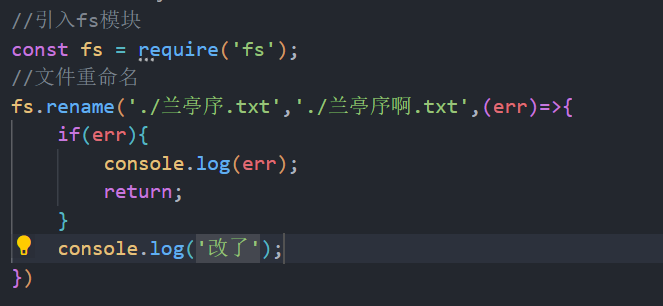


文件复制

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



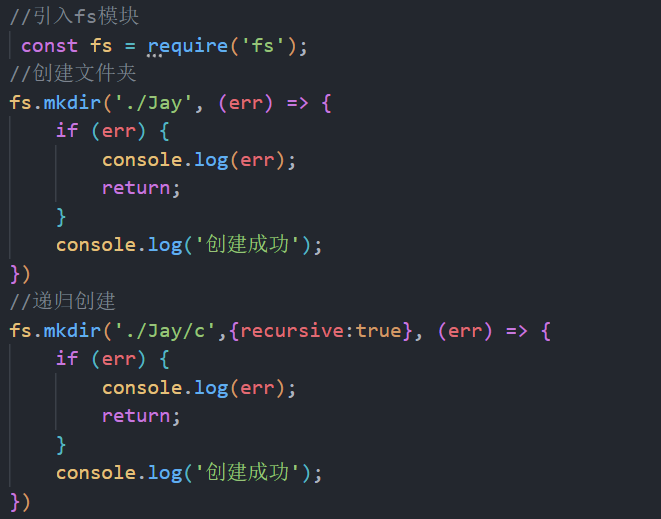
文件的移动也是用的这个原理。

文件删除

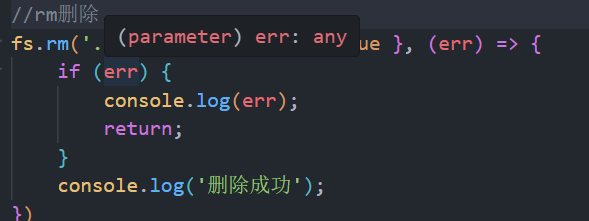


这两种删除效果是一样的。

文件夹操作



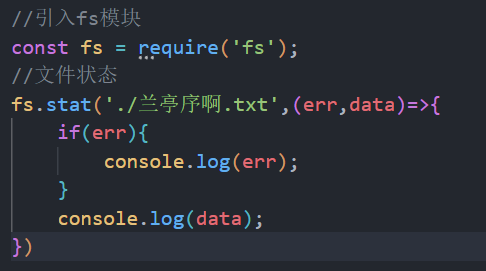




文件状态

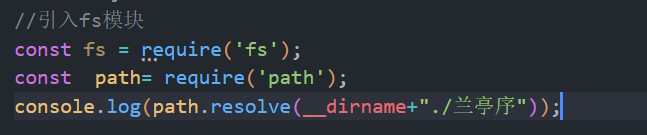
查看文件的位置，创建时间，最后打开时间等，下面这些东西就是从这个里面拿的。





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。

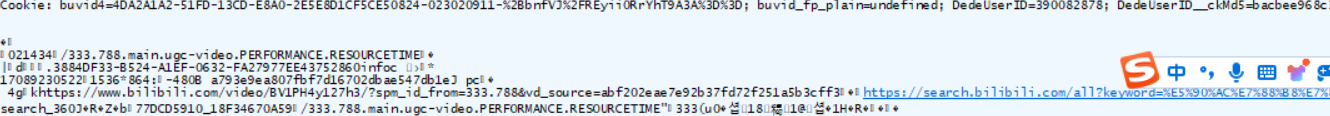
1. **HTTP协议**

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。

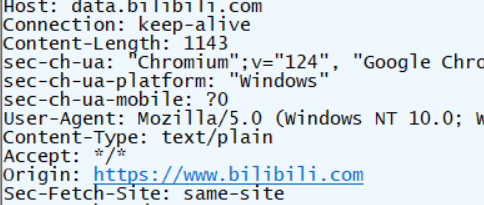
data.bilibili.com主机名

V2/log为路径

后面的一串就是查询字符串，

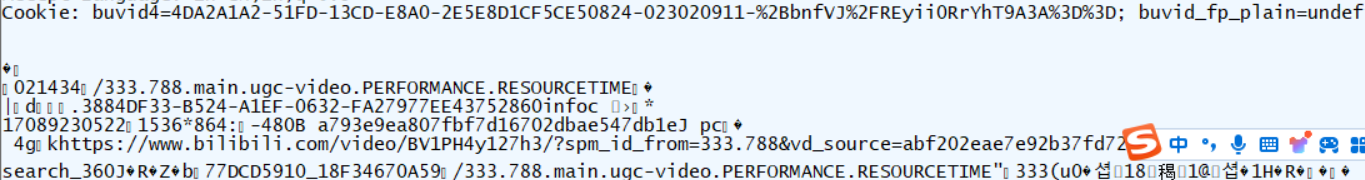
HTTP/1.1就是http协议的版本号，这里用的是1.1实际上已经更新到了3.

**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。

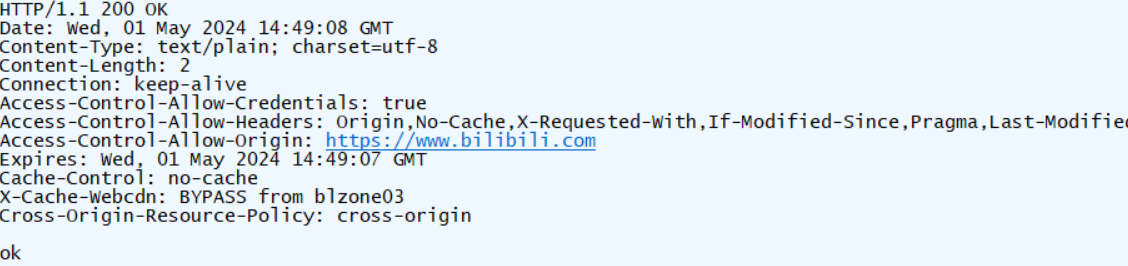
**请求体**



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**



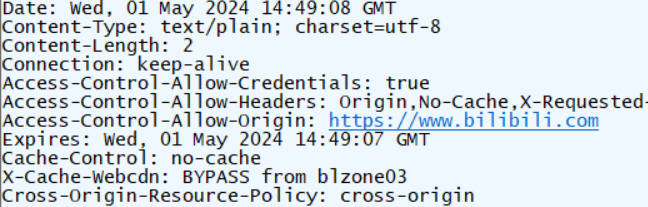
HTTP/1.1是版本号，

200是响应状态码

Ok是状态码的描述。



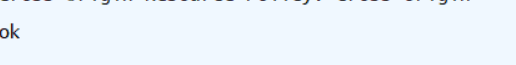
**响应头**



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。

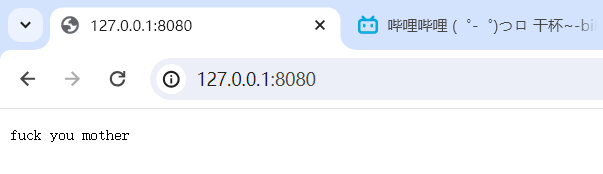
1. **关于http模块**

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。



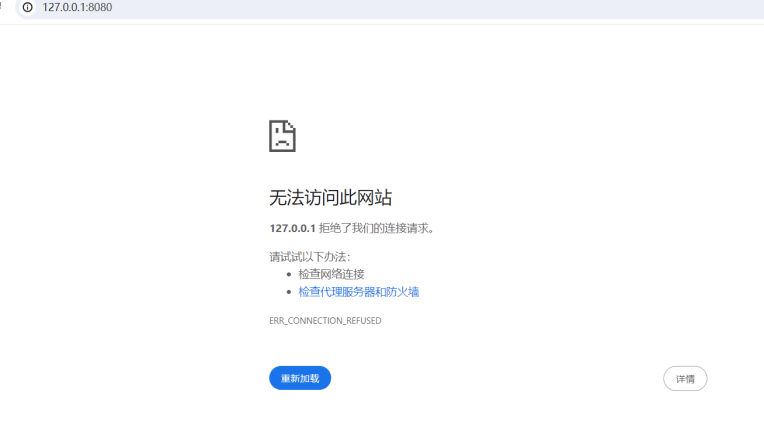
我们node ./路径名就启动成功了，然后就可以去浏览器打开服务。



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。**

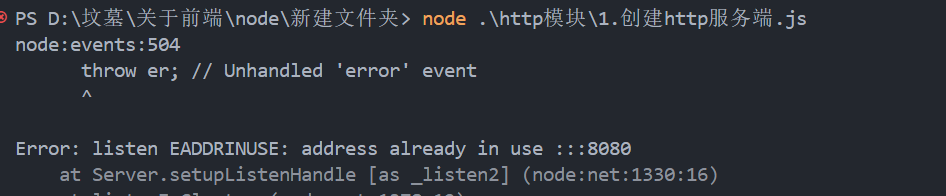


1. **服务启动后代码更新了，这时候要重启一下才能实现客户端的更新。**
2. **上面用response.end写的响应报文如果是中文就会乱码，这时候就要添加一个响应头。**

****

**这样就告诉了浏览器端响应报文的特性，解决乱码问题。**

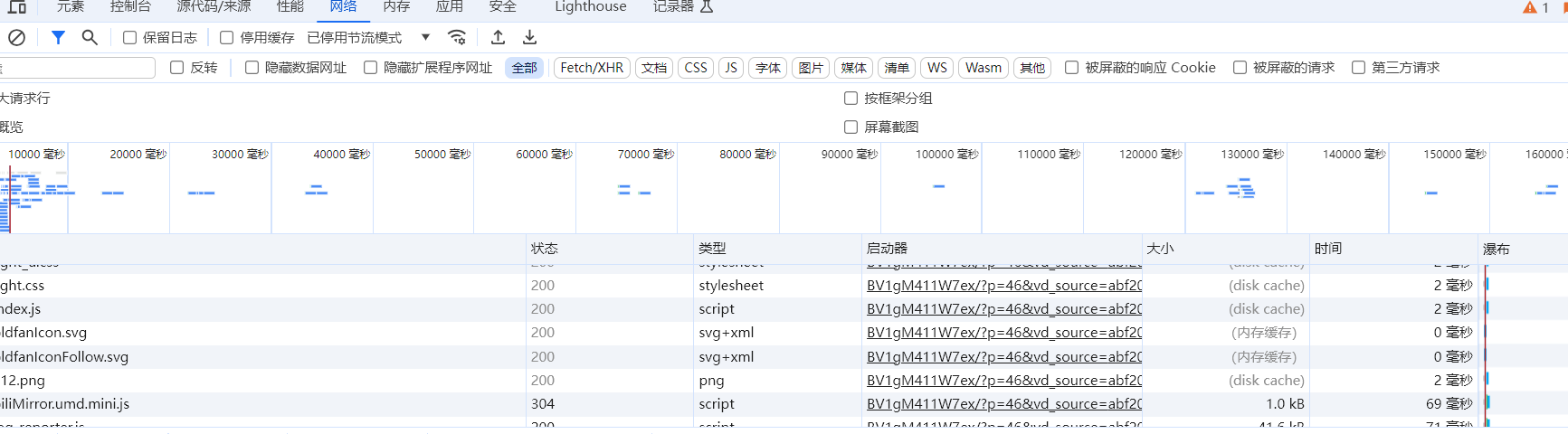
1. **如果我们已经在一个端口启用了一次服务，再次启动是无法在这个端口启动的。这时候有两种解决办法，第一种是终止之前打开的端口，另一种就是换一个端口。**



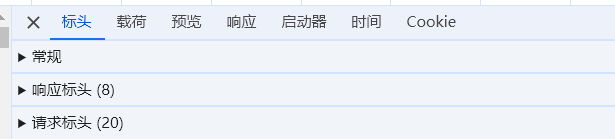
**上面是端口被占用的报错，另外补充一点。http协议有默认端口。80,8080,9000等等。用这些端口默认就不加端口号了。百度就是用这样的方法。**

**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。

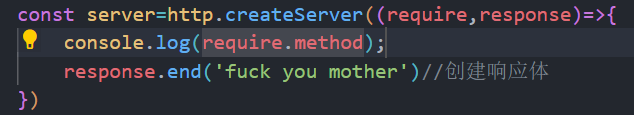
响应体点击响应就可以查看，请求体看载荷就行。



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：**

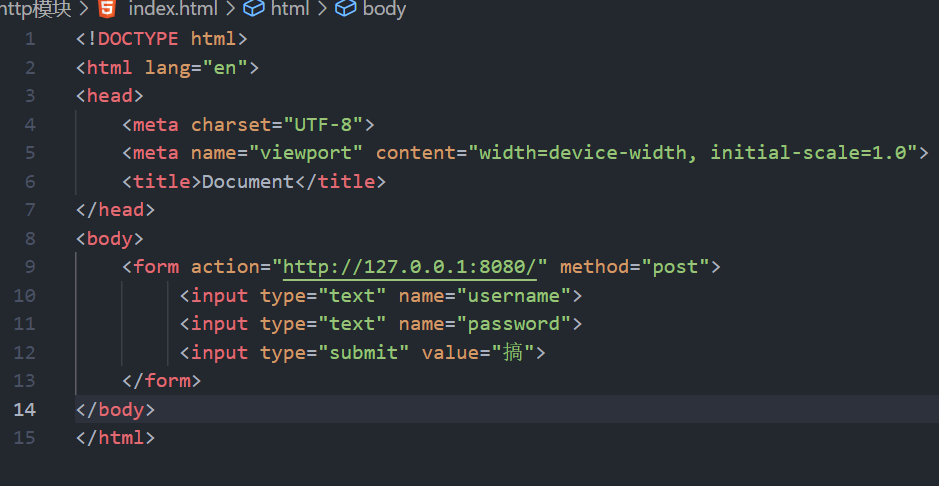
这些就是方法不细细一个个写了，注意写在我们创建的服务里面就行了。如下面这样。



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。

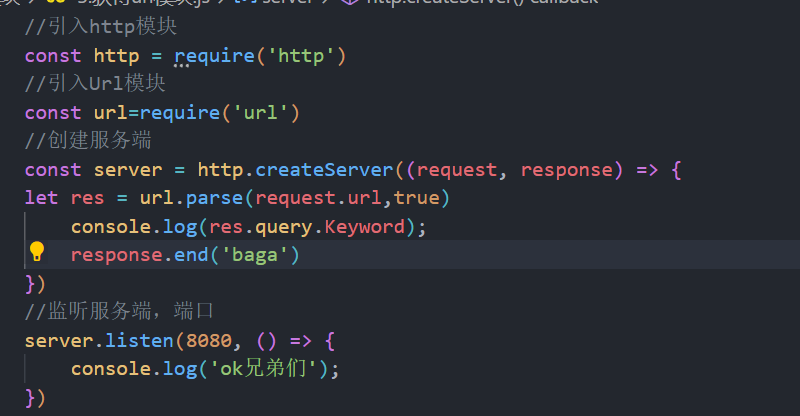


通过这个表单就可以实现请求体的传输。

**获得路径和查询参数的方法：**

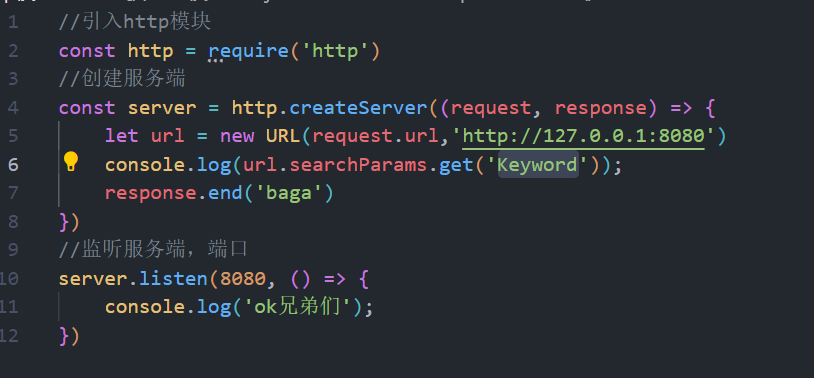
上面的获得请求地址上面是有获得URL地址的方式的，但是获得的都是混合起来的地址，不方便操作。但不是不能操作，用split这种方法分开了也是能操作的。但是开发讲究效率所以提供两种方法操作。

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。

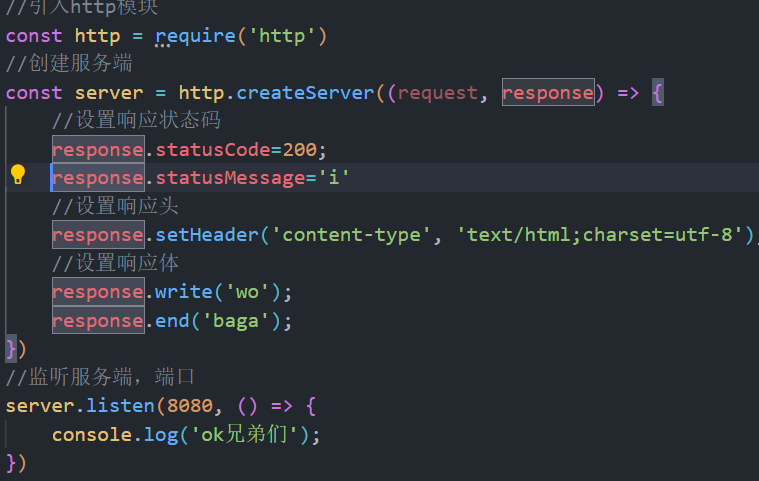


代码如下：

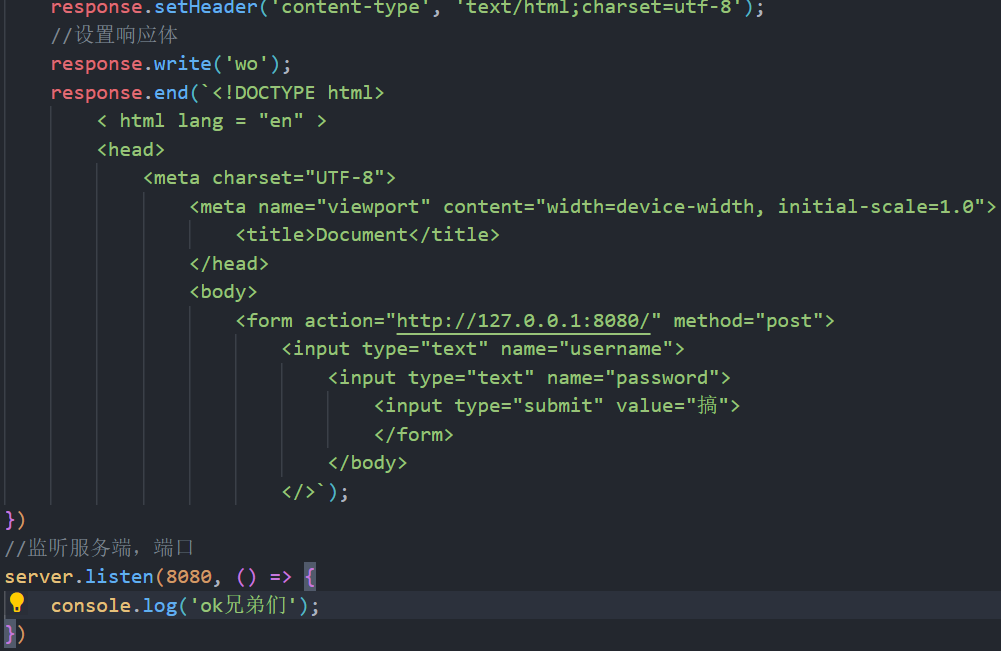


本质其实就是一个通过路径来设置响应体，并没有什么特别的地方。就是结合了一下上面的语法。像获得路径，请求方式之类的。

**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。

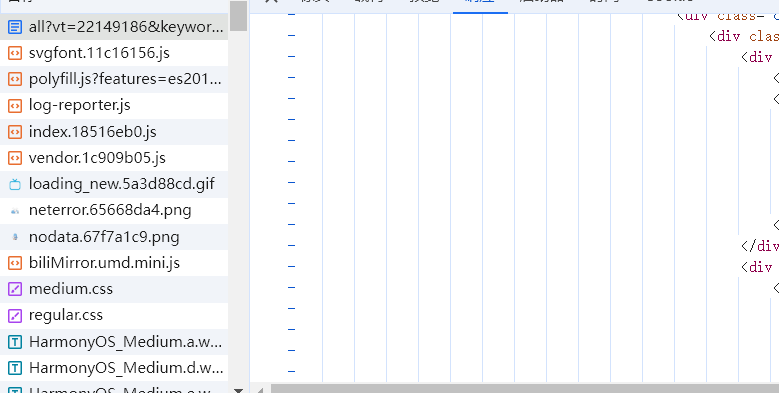


这样写响应太多且不会给你报错写错了也不知道，为了简单省事。我们选择直接用fs模块从外部引入。这就和上面的fs联系起来了。

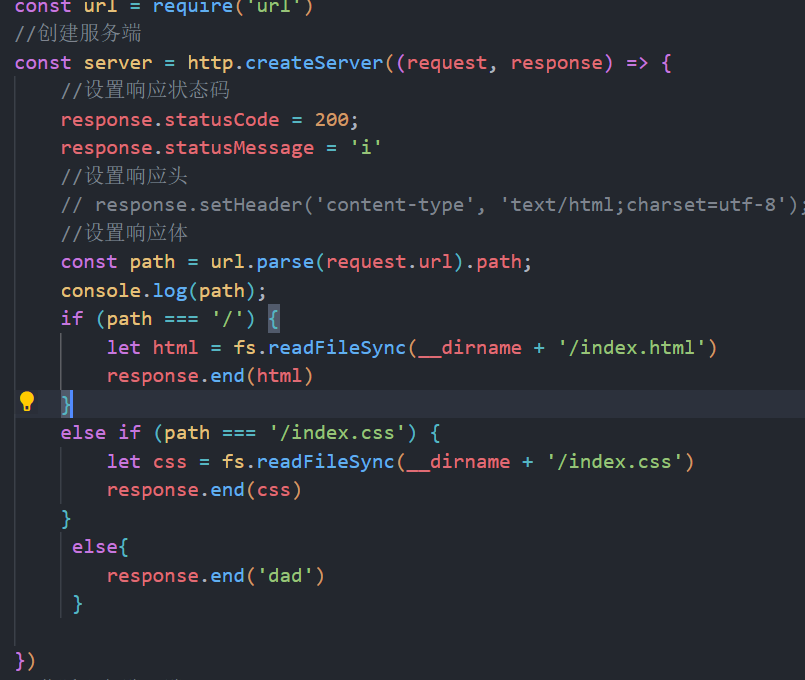


**网页资源请求与响应的过程：**

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



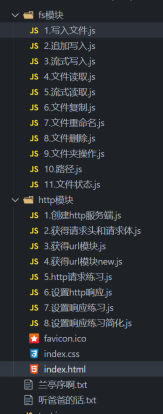
这样写一个两个可以，多了之后操作复杂程度必将爆炸，所以我们就找寻其中的规律，发现他的请求绝对路径\_\_dirname＋pathname进行一个拼接就是他的请求路径。代码如下：可以省去if，else判断。



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。

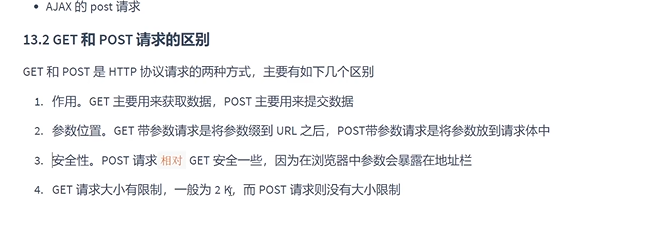


URL通常用于一下场景。



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

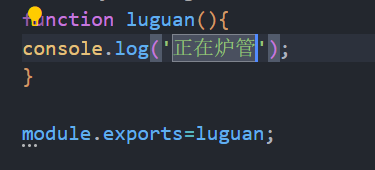
与vue的组件化是一个意思。之前

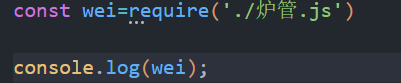
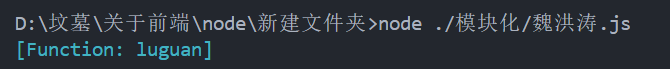
语法为：

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

导入：require(路径)

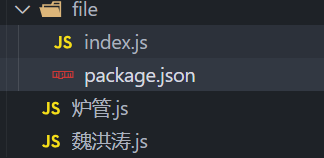
下面是一个例子：

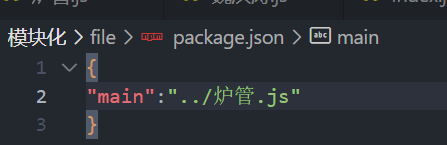
创建了一个撸管方法用module.exports导出暴露。再用require通过相对路径引入，与内置模块进行一个区分，内置模块直接引入就行了不用./，../这种相对路径。引用代码如下

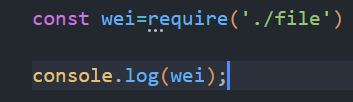
，require用的是相对路径但是有一点是要注意的./不能不写但是后面的后缀可以不写，他会给你默认了，如果名字一样那就默认先认js再认JSON，所以实际开发中还是加着比较好不然重名很难维护。

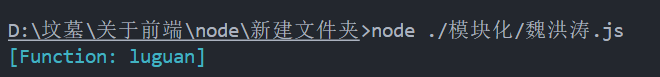
这样就引入成功了。其实还有一种暴露数据的方法就是exports.value=值，其实mode.export=exports={}是一个对象形式，这样就很清除明了了。

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。