# 引用部分

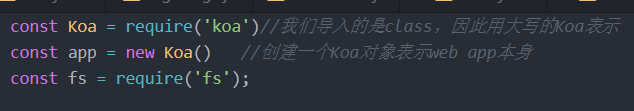
koa是Nodejs的一新款web框架，非常好用，用起来比express爽很多，用过就知道了。

初学者选择nodejs的web框架时，可以直接选择Koa这款框架。

# 原生路由实现

原生路由的实现需要引入fs模块来读取文件。然后再根据路由的路径去读取相应的文件，返回给页面，进行渲染。如果不清楚fs模块可以看[廖老师的讲解](https://www.liaoxuefeng.com/wiki/001434446689867b27157e896e74d51a89c25cc8b43bdb3000/001434501497361a4e77c055f5c4a8da2d5a1868df36ad1000)。

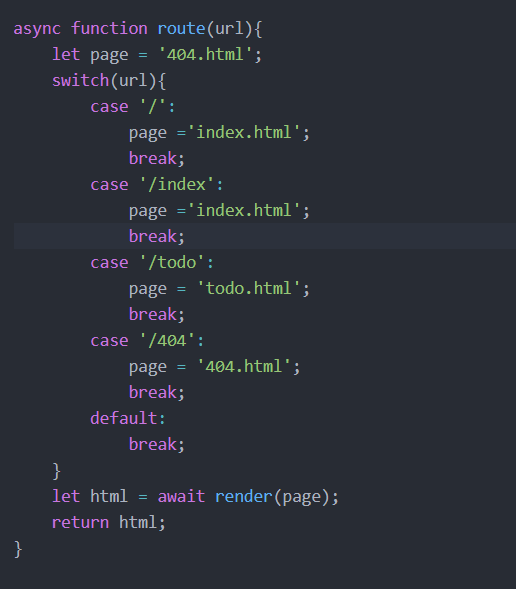
## 导入部分



## 入口



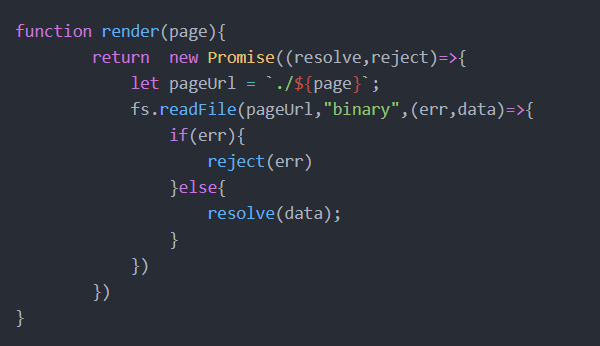
## 实现服务端路由的部分（可组件化）



上面的html页面是放在同文件夹下的，可以自己选择地方放，然后该路径

## render函数部分(重点)

Promise对象实现对路由信息的跳转，使用fs模块进行对html文件的读取(readFile)然后将结果吐出去，这也是为什么叫做服务端渲染的缘由(加入了fs模块文件读取操作)。



# 一般路由的实现

## 最基本路由



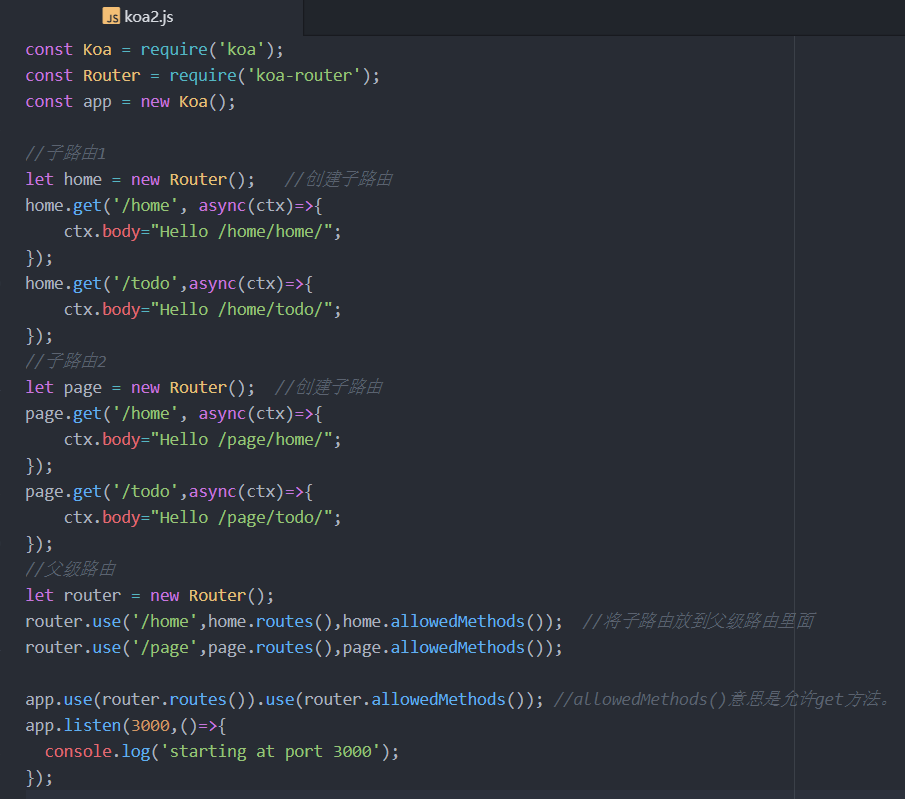
也就是express里面常用的写法。

注意那个



这是允许post、get等请求，没有用这个方法页面会出现not found

## 父子路由的实现



要将子路由装到副路由才行。

通过上面代码的方法，先设置子类路由，再设置父类路由。

执行node app.js 打开http://localhost:3000/page/todo会显示 Hello /page/todo/ 。打开[http://localhost:3000/home/todo会显示 Hello /home/todo/](http://localhost:3000/home/todo会显示%20Hello%20/home/todo/)。

这样就可以根据自己的需求，设计不同层次的路由了。

服务端渲染的基本操作，还是蛮实用的。

# GET和POST请求数据获取

## Get方法

获取GET请求数据有两个途径。

1.是从上下文中直接获取

请求对象ctx.query，返回如 { a:1, b:2 }

请求字符串 ctx.querystring，返回如 a=1&b=2

2.是从上下文的request对象中获取

请求对象ctx.request.query，返回如 { a:1, b:2 }

//返回的是对象 "req\_query":{"username":"buppt","age":"24"}

ctx\_query 一样的

请求字符串 ctx.request.querystring，返回如 a=1&b=2

//返回的是字符串"req\_querystring":"username=buppt&age=24"

ctx\_querystring一样的

其实：ctx\_query和req\_query是相同的，ctx\_querystring和req\_querystring也是相同的。



只不过就是 request对象是在ctx里面封装的，需要点出来调用

还是蛮方便的

## 原生Post方法

对于POST请求的处理，koa2没有封装获取参数的方法。所以获取Post请求需要以下步骤：

1. 解析上下文ctx中的原生node.js对象req。
2. 将POST表单数据解析成query string字符串.(例如:user=buppt&age=24)
3. 将字符串转换成JSON格式。

意思就是post方法没有自己封装，需要自己动手写

### 第一部分:

要先判断是不是post请求，是的话进入解析数据字符串操作



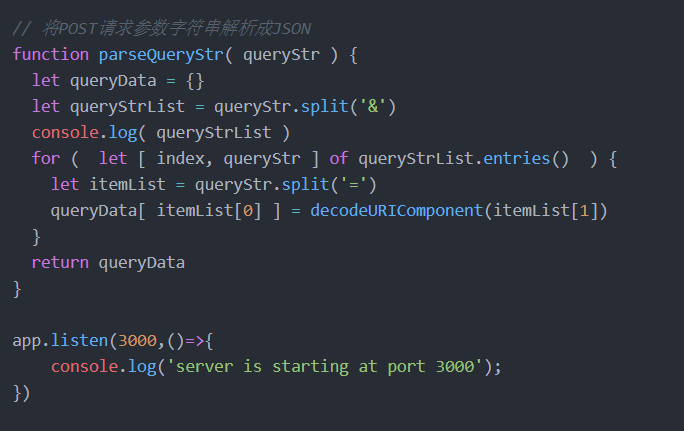
### 第二部分：

将得到的数据字符串传入json解析函数：



### 第三部分：

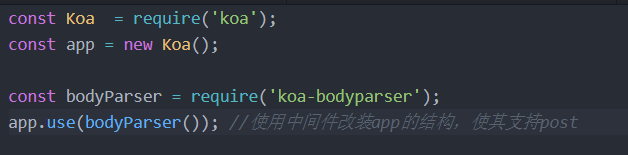
将数据字符串转化成json对象返回去



## 一般Post方法

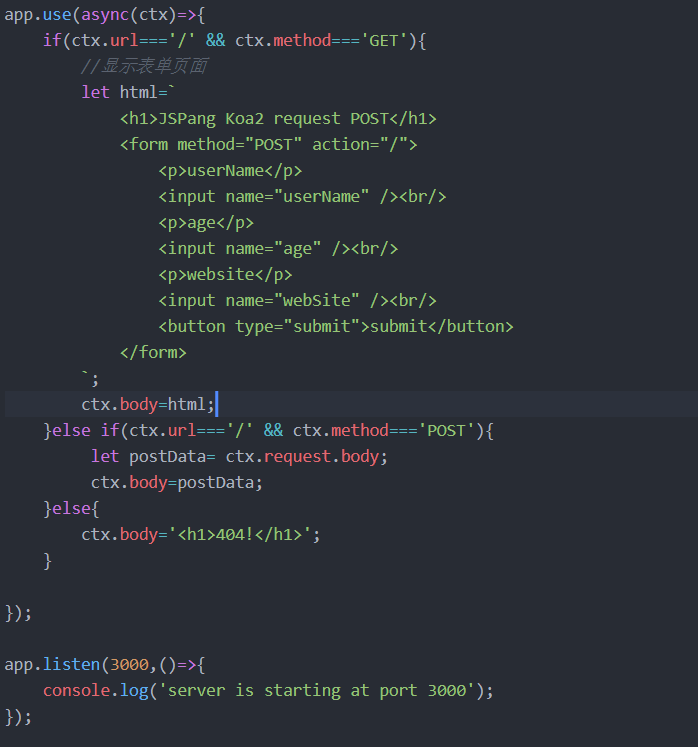
原生post这种比较麻烦的事情一定有中间件让我们使用，koa-bodyparser就是一个。对于POST请求的处理，koa-bodyparser中间件可以把koa2上下文的formData数据解析到ctx.request.body中。

### 引入中间件



中间件改装app，使其支持post方法

### 实现



就和get一样了 很简单的一行 ctx.request.body就行了

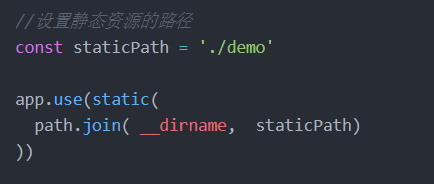
# koa-static实现静态资源服务器

在网络请求中，请求往往分成两种类型，一种是静态资源，直接从服务器的文件存储中读取，一种是动态资源，一般需要先从数据库获取数据，然后经过一定的处理，最后返回给客户端。

koa-static：用来处理静态资源的访问，因为它不涉及其他的处理过程，只是单纯的读取文件，所以单独抽离出来。



这里面是新建了一个文件夹，里面放html，js，css等文件，然后



在这里使用中间件，进行静态资源加载。

打开http://localhost:3000/demo.js 或http://localhost:3000/koa.jpg (后面是静态资源的文件名)，会在网页中显示静态资源的文件内容。

输入其他url会显示helloworld。

也算是属于搭建路由的一种方式吧，具体怎么用还要看后面。

补充：ps://blog.csdn.net/u014374031/article/details/80234602

这篇文章是讲怎样把static管理的静态资源在浏览器可视化的操作。

# cookie的使用

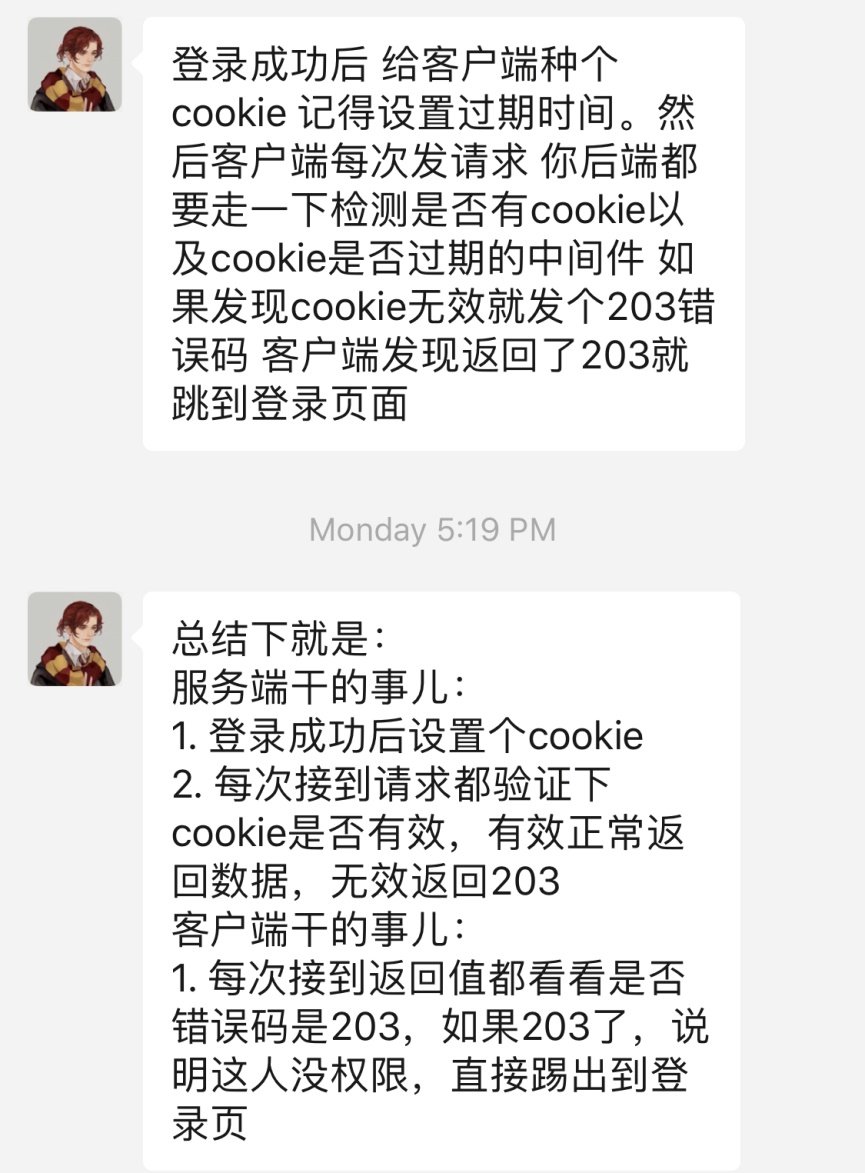
种cookie。

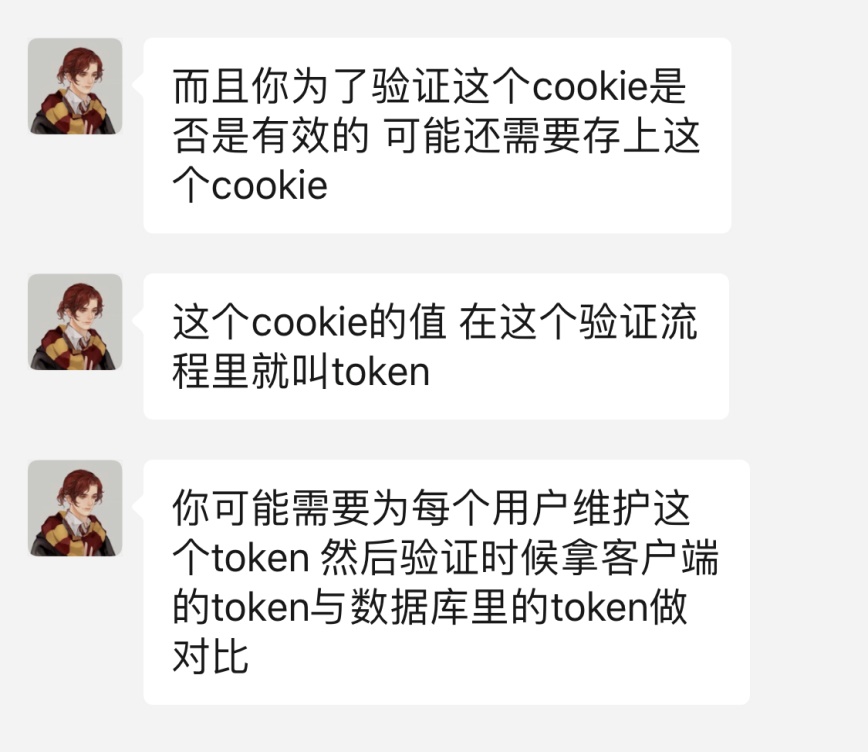
koa提供了从上下文直接读取、写入cookie的方法  
ctx.cookies.get(name, [options]) 读取上下文请求中的cookie  
ctx.cookies.set(name, value, [options]) 在上下文中写入cookie

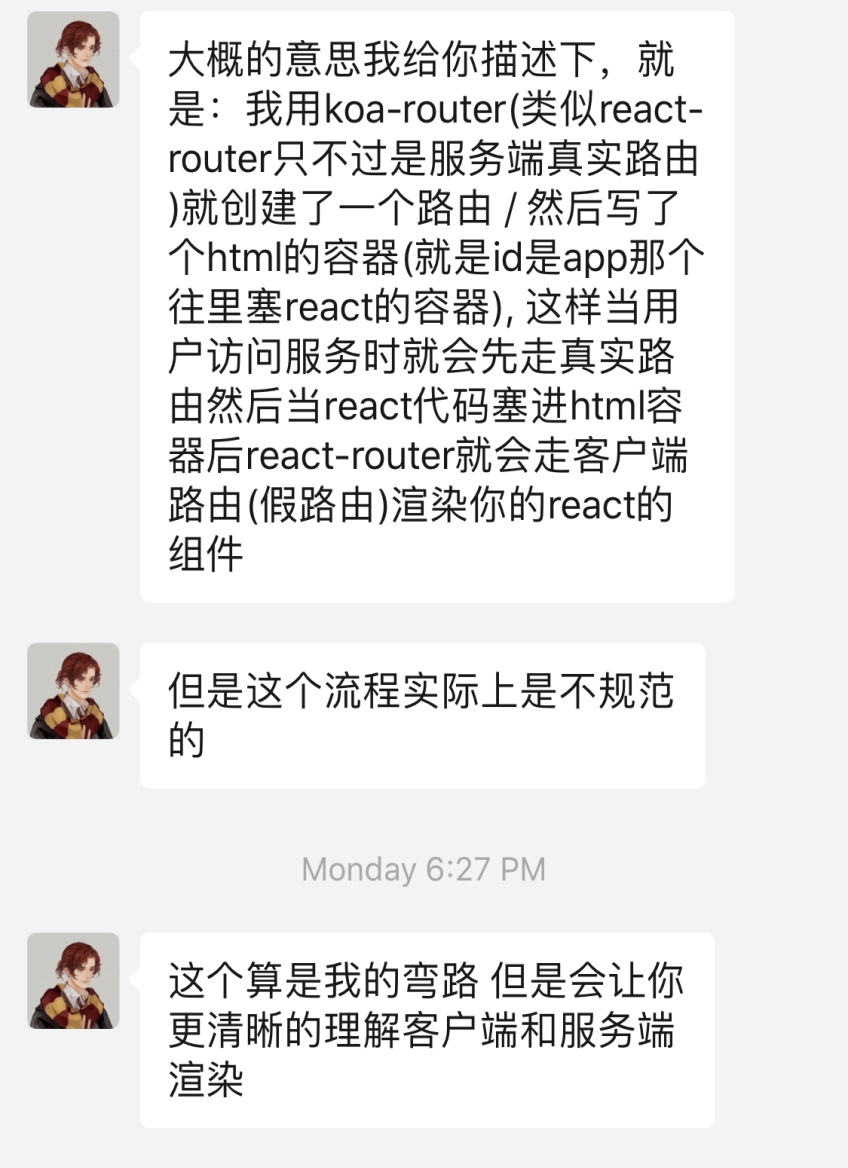


这时打开http://localhost:3000/ cookie里是没有cid的，先访问http://localhost:3000/index， f12查看cookie，cid已经存了进去，再访问http://localhost:3000/ 发现把cookie的cid读出并显示在了页面上。

具体操作都在这里了,获取的方法是差不多的,cookie的用法可以参考董哥给的建议.

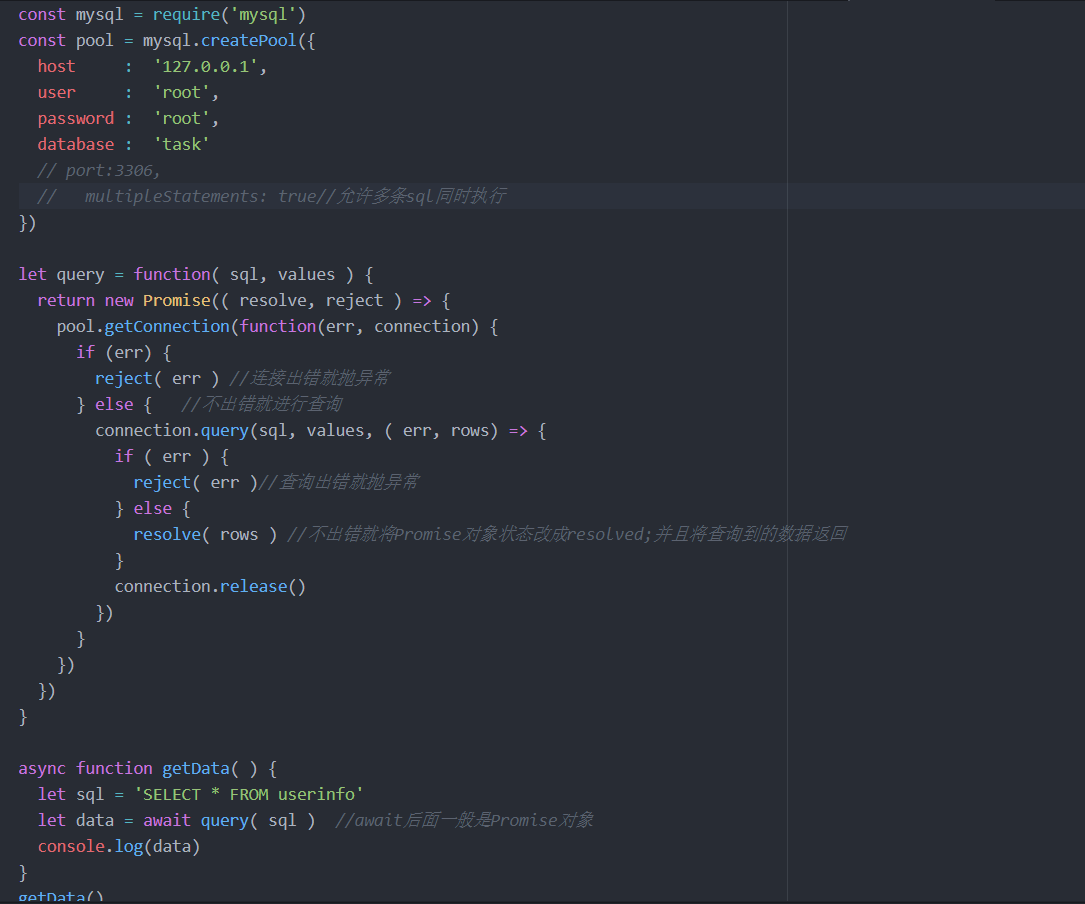






# Mysql的链接与操作

原本是这么连得：

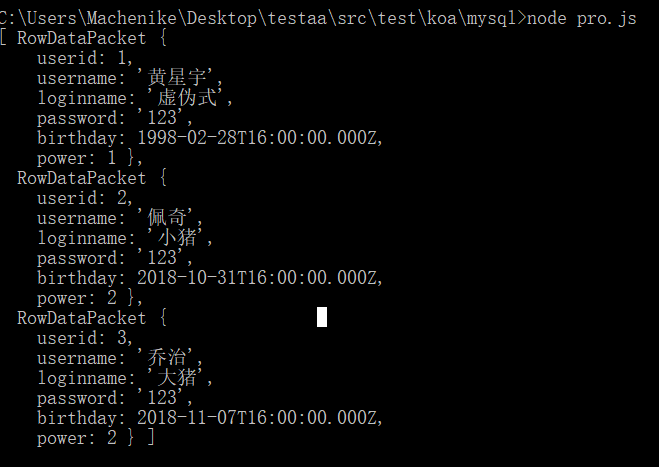


加了mysql-pro后可以省去手动连接的一步：

<https://www.npmjs.com/package/mysql-pro>（官方文档）



结果：



# 关于模块化

查阅了下关于Node的模块化和Es6模块化的区别所在。

具体在：<https://segmentfault.com/a/1190000010426778>

Node里面是用的

module.exports / exports进行模块输出，只能用require进行模块导入

ES6里面是用的export/export default进行模块输出，require和import进行模块导入



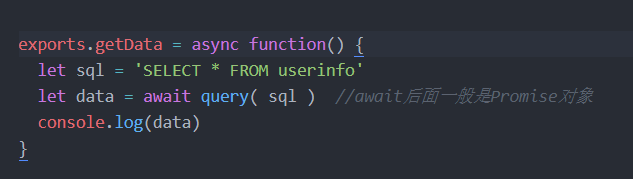
指向内存的解释如上。

exports只辅助module.exports操作内存中的数据，辛辛苦苦各种操作数据完，累得要死，结果到最后真正被require出去的内容还是module.exports的，真是好苦逼啊。就是说exports是操作的工具，改变的内存指向的其实是module的那部分。

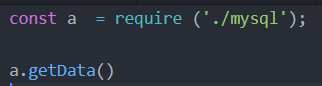
尽量都用 module.exports 导出，然后用require导入。

## Node列子

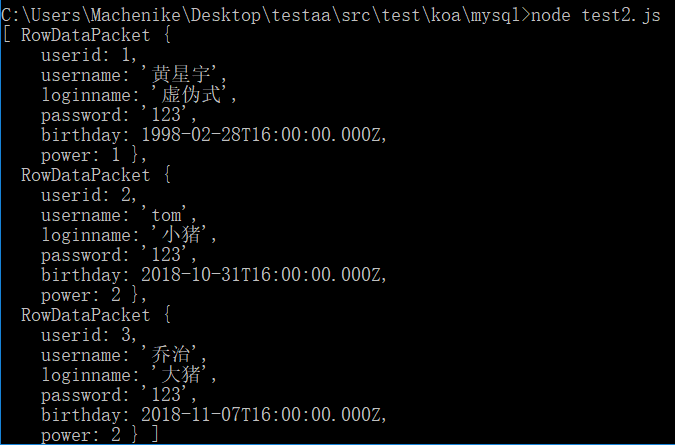
导出部分：



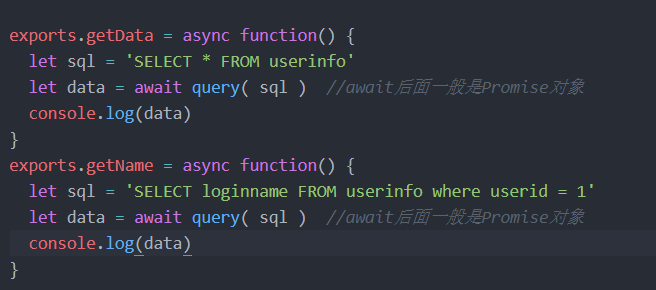
导入部分：



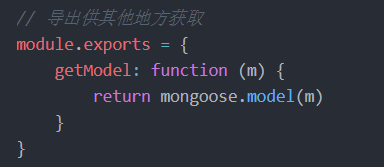
结果：



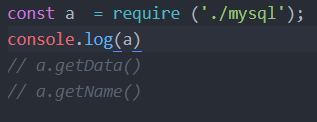
如果导出多个的话：



或者：

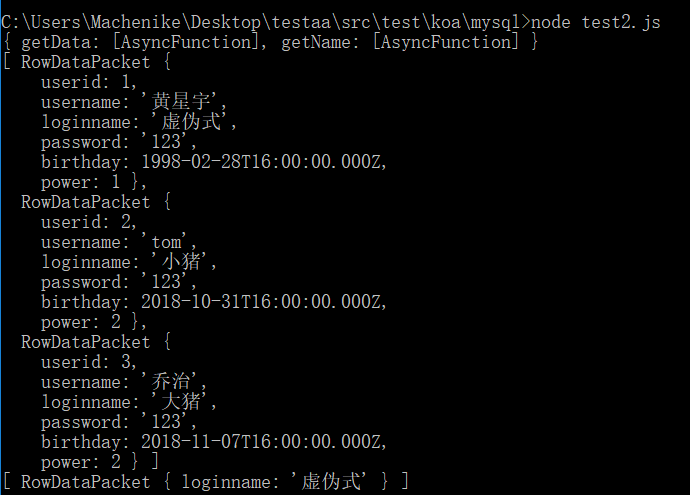


其实上面这个更方便直观



可以看到他是封装成了一个对象的（a），里面有对应的方法名，直接调用就行。





## ES6例子

主要是export和export default的区别

export类似于单个的无脑输出，要用{ }括起来，import那边也要用{ }找到对应的货。

export default规范一点，顾名思义是default 默认，就是说打包出去的是一个对象，

不要用{ }来构建而在import部分则使用抽象对象来使用即可(类比React组件的输出，是自己定义的)。

摘要：





ES6肯定更加灵活，具体还是看上面的网址，里面写的还可以。

# 注意

其实讲了这么多，这些后台的输出，也不过就是个接口，不负责页面渲染，那是React干的事，不过他本身也可以渲染（服务端渲染），恩要分清楚这个（不是单指董哥上面说的话啊哈哈）。