```
var express=require("express");//express 是封装了一些http 的方法，让其更好用
var app=express();//app 是一个请求监听的回调函数，每当请求到来的时候，会执行这个此函数
app.listen(9091);//在当前的本机服务器上监听9090 端口，
//源码
app.use((req,res,next)=>{//use 是表示使用中间件，中间件是置于所有的路由处理之前，否则无效，因为路由匹配到了代码就不再往下执行了；；将公共的代码写在中间件中，next 是一个回调函数，有一个可选参数ERR，next 没有参数时，继续执行后面的路由和逻辑，如果传了参数，这个参数表示错误，会跳过正常的处理逻辑，交给处理错误的中间件来处理；当执行此函数时，表示此中间件执行完毕，可以执行下面的业务逻辑了


});
app.use((err,req,res,next)=>{//错误处理中间件，这个中间件是专门处理错误的，相比较普通中间件，多了一个参数err，处理错误的中间件必须放在处理正常逻辑的中间件后面；
});
/**路由和中间件的区别和联系
 * 1）他们都在一个数组中，中间件在路由之前
 * 2）中间件不匹配请求方式和路径，而路由要匹配路径名，请
```

```
求方式和方法
 * 3）中间件多了一个next 参数，他的执行与否决定了是否执行use 后面的代码;
 * 4) 在中间件里可以结束响应，调用res.end();
 * */
app.get("/",(req,res)=>{
    res.end("首页")
});//第一个参数是请求路径，第二个是回调函数，当客户端通过get 请求方式请求"/"路径的时候，执行回调函数，回调函数有连个参数（req，res）和createServer 回调的函数是一样的
app.get("/login",(req,res)=>{
    res.end("登录")
});
app.post("/login",(req,res)=>{
    res.end("登录 2")
});
app.get("/user",(req,res)=>{
    res.end("用户主页")
});
//当请求的路径没有路由能处理的时候，返回你请求页面不存在
all 匹配所有的请求方式，*是处理路径的
```

```
app.all("*",(req,res)=>{
    res.end("你请求的页面不存在")
});//匹配所有的方法,
/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/
* 参数处理
 * 1）请求行 method url(pathname+query)
 * 2）请求头 headers
 * 3）请求体 body
var express=require("express");
var http=require("http"),
    url=require("url"),
    mime=require("mime"),
    path=require("path"),
    fs=require("fs");
var app=express();
app.get("/",(req,res)=>{
   console.log(req.method);
    //var urlObj=url.parse(req.url,true);
    console.log(req.path);
    console.log(req.query);
    res.end("123")
});
```

```
//获取某个客户的信息,: id 是占位符, 匹配一个字符串
app.get("/users/:id",(req,res)=>{//app.get 方法的第一个参数还可以是一个正则;
    res.end("id="+req.params.id);//params 是 express 帮我们添加的对象的属性, 属性名就是占位符, 属性值是实际请求的时候字符串只能为负对应的部分
});
app.listen(1113,()=>{
    console.log("victory 1113");
});
/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/
var users =[{id:1,name:'张三'},{id:2,name:'李四'}];
var express = require('express');
var path = require('path');
var app = express();
//设置模板引擎 用来添加文件后缀的
app.set('view engine','html');
//console.log(app.get('view engine'));
//设置模板存放目录 指定一个模板的绝对路径
app.set('views',path.join(__dirname,'views'));
//设置对于html 类型的模板使用ejs 来进行渲染
app.engine('.html',require('ejs').__express);
```

```
// http://localhost:9090/users/1
/**
 * 1. 定义路由
 * 2. 取得ID
 * 3. 取得ID对应的user对象
 * 4. 渲染模板
 */
app.get('/users/:id',function(req,res){
    var id = req.params.id;
    var user = users.find(function(user){
        return user.id == id;
    });
    //用数据渲染模板
    // 1.模板的相对路径
    // 2 参数是渲染模板数据对象 模板里的变量要从数据对象的属性中取值
    res.render('user',user);
})
app.listen(9090);
/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/
var express = require('express');
var app = express();
```

```
app.use(function(req,res,next){
    console.log(req.path);
    next();
});
app.get('/',function(req,res){
    //重定向 告诉客户端重新向新的URL 地址发起请求
    res.redirect('/user');
});
app.get('/user',function(req,res){
    res.send('用户');
});
app.listen(9090);
/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/
var express = require('express');
var http=require("http"),
    url=require("url"),
    mime=require("mime"),
    path=require("path"),
    fs=require("fs");

var app = express();
//使用ejs 模板引擎
```

```
app.use((req,res,next)=>{
    res.locals.name="zfpx";//模板引擎渲染的时候读取的其实都是读取的这个 locals 这个对象中的属性，只不过在 render 的时候，express 会把 render 的第二个参数复制到 locals 中去;但是当很多页面模板中都需要某些数据或者变量，我们将这些数据在一个中间件中给 locals 中赋值
});
app.use((req,res,next)=>{
   res.redirect=(path)=>{
      res.statusCode=302;//重写状态码，告诉客户端要重新请求新的路径
         res.setHeader("Location",path);//重写响应头，告诉客户端我要重新定向哪里
   };
});
app.set("view engine","html");//识别 render 第一个参数的后缀名
app.set("views",path.resolve("views"));//views 是文件夹名,path.resolve("views"),从当前路径出发得到一个绝对路径,
app.engine(".html",require("ejs").__express);//解
```

```
析模板



app.get("/reg",(req,res)=>{
   res.render("reg",{title:"用户注册
"},function(err,html){
       //render 有三个参数，第一个参数是模板名字，    如果不传后调函数，render 是先渲染模板，然后返回给客户端，结束响应结束；如果传递了回调参数，只管渲染渲染模板，待渲染结束后，将渲染后字符串作为参数传递给回调函数，（与渲染后的字符串同时传递给客户端的还有err），然后就不管了，至于重写响应头和结束响应都要交给回掉函数来手动处理，这个回调函数不是很常用，除非有时需要精微控制响应内容的时候才需要这么做；
   });//模板的路径 views 的父目录+view engine 后缀；render 自带end，获取模板，渲染模板为html，发送给客户端结束响应end；
});
app.listen(1120,()=>{
    console.log("victory 1120");
});
/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/
```

```
var locals={name:"zf"};//在模板引擎渲染的时候是读取的res.locals 对象中的属性
//复制对象 es6 方法 Object.assign(target,source);方法
// for in 循环
/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/
/**
 * 静态文件中间件
 * 当客户端访问服务器一个静态文件的时候，服务器返回一个静态文件，
 *
 * */
app.use(express.static(path.resolve("public")));//这里这个public 是存放所有静态资源文件的路径

//模拟
/**
 * 1）去除请求路径
 * 2）到根目录下查找有没有该文件，如果找到后返回，没有找到next()，看看后面的路由有没有;
/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/
```

```
var http=require("http"),
    url=require("url"),
    mime=require("mime"),
    path=require("path"),
    querystring=require("querystring"),
    util=require("util"),
    fs=require("fs");
http.createServer((req,res)=>{
    var urlObj=url.parse(req.url,true);
    var pathname=urlObj.pathname;
    if(pathname=="/write"){
        res.setHeader("Set-Cookie","age=8");//如果写多个cookie时，第二个参数装一个数组，每一个数组项是cookie值["name=zf","age=9"]这样客户端可以自动解析成多个cookie
        res.end();
    }else if(pathname=="/read"){
        var cookie=req.headers.cookie;
        cookie=querystring.parse(cookie,";","=");//querystring.parse(cookie);有三个参数，第一个是转化字符串，第二个是字段分隔符，不写时默认&，第三个是key value分隔符，不写时默认= 可以指定;
```

```
        console.log(cookie);
cookie=util.inspect(cookie);//util.inspect(); 方法将对象转化成字符串，万能方法，什么都能转
        res.end(cookie);
    }
}).listen(1123,()=>{
    console.log(1123);
});
/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/
var http=require("http"),
    url=require("url"),
    mime=require("mime"),
    path=require("path"),
    fs=require("fs");
var express=require("express"),
    expressSession=require("express-session");
var app=express();
//当使用了 session 中间件之后，在请求头上就多了一个 req.session 对象，req.session 就是此客户端在服务器对应的数据对象
app.use(expressSession({
```

```
    resave:true,//每次都重新保存session对象
    secret:"zfpx",//加密值是一个密钥
    saveUninitialized:true//保存未初始化的session
}));
app.get("/visit",(req,res)=>{
   var
visit=req.session.visit?req.session.visit+1:1;
    req.session.visit=visit;
    res.send(`客官，这是你的第${visit}次访问`);
});
app.listen(1126,()=>{
    console.log(1126);
})
/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/
var http=require("http"),
    url=require("url"),
    mime=require("mime"),
    path=require("path"),
    fs=require("fs");
var express=require("express"),
    cookieParser=require("cookie-parser");
var app=express();
```

```
app.use(cookieParser());
app.get("/visit",(req,res)=>{
    var cookies=req.cookies;
    var visit=1;
    if(cookies&&cookies.visit){
        visit=parseInt(cookies.visit)+1;
    }
        //console.log(visit);
        res.cookie("visit",visit);
        res.send(`这是你的第${visit}次访问`);

});
app.listen(1125,()=>{
   console.log(1125);
});
/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/


/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/
/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/
/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/
/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/*/
```