DAY33 express

2023年6月1日 18:00

http前后端交互

后端

一.创建一个http的服务器

```
const http = require("http");
const server = http.createServer((req, res) => {
  // res.end("hello,server");
});
server.listen(9000);
console.log("服务器启动成功,端口是9000");
```

二.制作一个接口，提供给前端访问，约定请求方式是GET

```
const http = require("http");
const server = http.createServer((req, res) => {
  res.setHeader("Access-Control-Allow-Origin", "*"); //解决跨域
  res.end(
    JSON.stringify({
      name: "zhangsan",
      age: 18,
    })
  );
});
server.listen(9000);
console.log("服务器启动成功,端口是9000");
```

三.前端的代码最终放入服务器，利用服务器进行访问前端的代码。

```
const http = require("http");
const url = require("url");
const fs = require("fs");
const server = http.createServer((req, res) => {
  const { pathname } = url.parse(req.url, true);
  if (pathname === "/index") {
    //利用fs模块，读取index.html里面的内容
    fs.readFile("../client/index.html", (err, docs) => {
      //docs是index.html里面的内容
      if (err) throw err;
      res.end(docs);
    });
  }

  if (pathname === "/list") {
    //利用fs模块，读取index.html里面的内容
```

```
    fs.readFile("../client/list.html", (err, docs) => {
      //docs是index.html里面的内容
      if (err) throw err;
      res.end(docs);
    });
  }

  if (pathname === "/goods/list") {
    res.end(
      JSON.stringify({
        name: "zhangsan",
        age: 18,
      })
    );
  }
});
server.listen(9000);
console.log("服务器启动成功,端口是9000");
```

四.获取前端get方式提交的数据，返回给前端

```
const http = require("http");
const url = require("url");
const fs = require("fs");
const qs = require("querystring");
const path = require("path");
const server = http.createServer((req, res) => {
  const { pathname, query } = url.parse(req.url, true);
  // 加载首页
  if (pathname === "/index") {
    //利用fs模块，读取index.html里面的内容
    fs.readFile("../client/index.html", (err, docs) => {
      //docs是index.html里面的内容
      if (err) throw err;
      res.end(docs);
    });
  }

  // 获取首页传入的数据，并且将数据返回
  if (pathname === "/user/registry" && req.method === "GET") {
    res.end(
      JSON.stringify({
        status: 1,
        msg: "接收前端数据成功",
        data: query,
      })
    );
  }
```

获取首页传入的数据，并且将数据返回 POST

```
    if (pathname === "/user/login" && req.method === "POST") {
      // 获取POST提交的数据
      let rawData = "";
      req.on("data", (chunk) => {
        rawData += chunk; // 获取数据的事件，一段一段获取，这里拼接收集
      });
      req.on("end", () => {
        res.end(
          JSON.stringify({
            status: 2,
            msg: "接收数据成功",
            data: qs.parse(rawData),
          })
        );
      });
    }
  });
  server.listen(9000);
  console.log("服务器启动成功,端口是9000");
```

前端

index.html

```
  // 调用后端接口
  // const xhr = new XMLHttpRequest();
  // xhr.open('GET', 'http://localhost:9000/goods/list');
  // xhr.onload = function () {
  //   console.log(xhr.responseText);
  // }
  // xhr.send()
  // 将前端的数据传递给后端，采用GET
  // const xhr = new XMLHttpRequest();
  // xhr.open('GET', 'http://localhost:9000/user/registry?username=尼古拉斯赵四&age=250');
  // xhr.onload = function () {
  //   console.log(xhr.responseText);
  // }
  // xhr.send()
  // 将前端数据传递给后端，采用POST
  const xhr = new XMLHttpRequest();
  xhr.open('POST', 'http://localhost:9000/user/login');
  xhr.onload = function () {
    console.log(xhr.responseText);
  }
  xhr.setRequestHeader('content-type', 'application/x-www-form-urlencoded');
  xhr.send('xingming=张三&age=18')
```

Express

一.Express基于 Node.js 平台，快速、开放、极简的 Web 开发框架

二.Node.js的http模块可以创建服务器，但操作不够方便灵活，于是就有了express,koa等模块

三.express框架介绍

1.express 是一个简洁而灵活的 node.js Web应用框架,

2.提供了一系列强大特性帮助你创建各种 Web 应用，和丰富的 HTTP 工具。

3.使用 express 可以快速地搭建一个完整功能的网站

四.res.send()

1. send方法内部会检测响应内容的类型
2. send方法会自动设置http状态码
3. send方法会帮我们自动设置响应的内容类型及编码

五.中间件的概念

中间件就是一个具有特定功能的封装函数(新的叫法，其实就是原生js里面的函数或者方法)

- 中间件三个参数
- request 表示请求，记录了前端给后端的所有信息
- response 表示响应，记录了后端给前端的信息
- next 表示连接，将控制权交给下一个中间件，直到遇到结束请求的中间件

六.app.use()匹配所有的请求方式

可以直接传入请求处理函数，代表接收所有的请求

第一个参数也可以传入请求地址，代表不论什么请求方式，只要是这个请求地址就接收这个请求。

```
// app.use方法
// app.use((req, res, next) => {
//   res.send("111111111111111");
// });
```

localhost：8000

```
// app.use("/hehe", (req, res, next) => {
//   res.send("hehehehehehe");
// });
```

localhost：8000/hehe

```
// use配合next中间件
// app.use((req, res, next) => {
//   console.log(1111111111111111111);
//   next();
// });
```

```
// app.use((req, res, next) => {
//   console.log("22222222222222222");
//   next();
// });
// app.use((req, res, next) => {
//   console.log("333333333333333333");
//   next();
// });
// app.use((req, res, next) => {
//   res.send("44444444444");
// });

// 发起get请求
// app.get("/index", (req, res, next) => {
//   req.name = "你好，中间件";
//   next();
// });
// app.post("/index", (req, res, next) => {
//   res.send(req.name);
// });
// app.put("/index", (req, res, next) => {
//   res.send(req.name);
// });
// app.delete("/index", (req, res, next) => {
//   res.send(req.name);
// });
```

加载静态资源(前端的页面)

```
app.use(express.static(path.join(__dirname, "client")));
app.listen(8000);
console.log("服务器启动成功");



// 准备第一个接口
// app.get("/index", (req, res, next) => {
//   res.send("这是第一个get接口返回的数据");
// });

// 准备接收前端get传入的数据，并将其返回给前端
// app.get("/user/registry", (req, res, next) => {
```

req.query:直接获取前端传入的数据

```
//   console.log(req.query);
//   res.send({
//     status: 1,
//     msg: "接口请求成功，获取你传入的参数",
//     data: req.query,
//   });
```

```
// });
```

准备接收前端post传入的数据，并将其返回给前端

使用body-parser第三方模块解决post获取数据

如果extended的值是false，借助node内置querystring模块进行解析数据

如果extended的值是true，借助node第三方qs模块进行解析数据

```
// app.use(bodyParser.urlencoded({ extended: false }));
// app.post("/user/login", (req, res, next) => {
//   //req.body：获取前端post方式传入的数据
//   res.send({
//     status: 1,
//     msg: "接口请求成功，获取你传入的参数",
//     data: req.body,
//   });
// });
// put请求
// app.put("/user/test", (req, res, next) => {
//   res.send("ok1");
// });
// app.delete("/user/del", (req, res, next) => {
//   res.send("ok2");
// });
// app.listen(5000);
// console.log("服务器启动成功,监听的端口是5000");
```

路由

url和函数的对应关系

```
const express = require("express");
const fs = require("fs");
const app = express();

// 导入二级路由，然后使用二级路由
// const indexRouter = require("./router/index");
// 路由中间件
// app.use("/hehe", (req, res, next) => {
//   res.send("hehe地址的返回值");
// });
// 路由中间件
// app.get("/list", (req, res, next) => {
//   res.send("list地址的返回值");
// });
```

```
// 错误中间间
// app.use((req, res, next) => {
//   fs.readFile("./hehe.txt", (err, docs) => {
//     if (docs) {
//       res.send(docs);
//     } else {
//       next(err); //将错误信息传给下一个中间件
//     }
//   });
// });
// app.use((err, req, res, next) => {
// 自由设定http状态码，同时发送错误信息
//   res.status(500).send(err);
// });

// 二级路由
// 导入入二级路由，然后使用二级路由
const indexRouter = require("./router/index");
const listRouter = require("./router/list");
// 使用二级路由
app.use("/index", indexRouter);
app.use("/api", listRouter);
app.listen(5000);
console.log("服务器启动成功,监听的端口是5000");
```

localhost: 5000/index/hehe

router/index.js

```
// 基于模块化的方式
const express = require("express"); //引入express
const router = express.Router(); //利用Router方法创建router对象
// 二级路由
router.get("/hehe", (req, res, next) => {
  res.send("我是二级路由hehe地址的返回值");
});
module.exports = router; //导出模块
```

http和WebScoket

1.http:单向通信，前端(客户端,浏览器端)发起请求，后端(服务器端)响应请求。

2.WebScoket:双向通信，前后端都可以发起请求。

WebSocket 是 HTML5 提供的一种在单个 TCP 连接上进行全双工通讯的协议。（双向通信协议）

实现客户端与服务器之间的双向通信，允许服务端主动向客户端推送数据

在 WebSocket API 中，浏览器和服务器只需要完成一次握手，两者之间就直接可以创建持久性的连接，并进行双向数据传输。

3.WebSocket 和 HTTP 区别?

相同点：

1. WebSocket和http都是一样基于TCP的可靠性传输协议；

异同点：

1. WebSocket可以双向发送或接受信息，而 HTTP 是单向的（HTTP 通信只能由客户端发起，不具备服务器主动推送能力）；
2. WebSocket的使用，需要先进行一次客户端与服务器的握手，两者建立连接后才可以正常双向通信，而HTTP是一个主动的Request对应一个被动的Response；