服务端开发

服务端基础

软件结构

- C/S体系结构
  - 客户端/服务端，例如QQ、网盘
  - 优点：交互性好,对服务器压力小,安全
  - 缺点：服务器更新时需要同步更新客户端
- B/S体系结构
  - 浏览器/服务端，例如网站
  - 优点：不需要更新客户端
  - 缺点：交互性差,安全性低

web服务器

能够提供web访问服务的机器就是网站服务器，能接收浏览器端的请求，能对请求做出响应。

IP地址

在互联网中电脑的唯一标识。例如：113.45.94.17。浏览器端请求服务器端的资源首先要有这台服务器的IP地址，才能找到这台服务器。

如果将本机作为服务器，本机有一个特定的IP是 127.0.0.1

域名

由于IP地址难于记忆，所以产生了域名的概念，所谓域名就是平时上网所使用的网址。

IP地址与域名是对应的关系，在浏览器的地址栏中输入域名，会有专门的服务器 (DNS) 将域名解析为对应的IP地址，从而找到对应的服务器。

如果将本机作为服务器，本机有一个特定的域名是 localhost

端口

通过IP地址找到对应的服务器以后，还需要指定端口来进一步确定访问的是当前服务器提供的什么服务。比如80是apache服务默认占用的端口，3306是mysql服务占用的端口。

URL

统一资源定位符，是互联网上标准资源的地址。

请求与响应

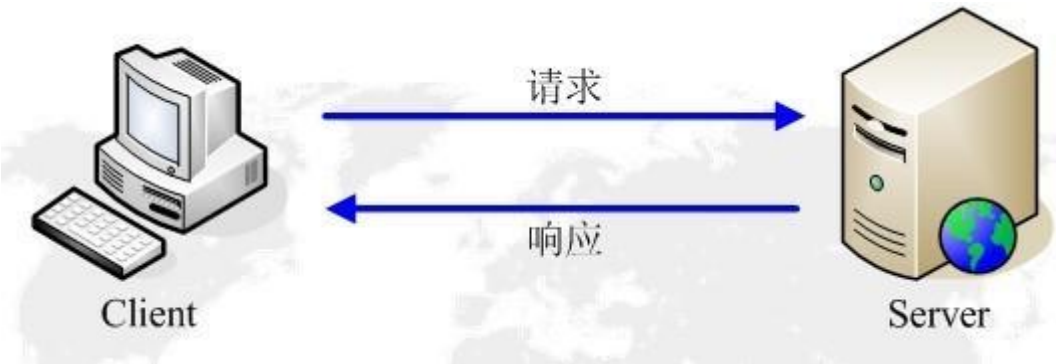
创建**web**服务器

在NodeJs中不需要安装额外的软件充当网站服务器，NodeJs中提供的HTTP模块即可创建web服务器。

```
// 引用系统模块
const http = require('http');
// 创建web服务器
const server = http.createServer();
// 当客户端发送请求的时候
server.on('request', (req, res) => {
    // 设置响应头
    res.writeHead(200, {
        'Content-Type': 'text/html;charset=utf8'
    });
    // 设置响应体
    res.write('<h1>哈哈哈</h1>');
    // 结束请求
    res.end();
});
// 监听3000端口
server.listen(3000, error => {
    if (!error) {
        console.log('服务器已启动，监听3000端口，请访问 localhost:3000')
    }
});
```

HTTP协议

超文本传输协议，用于从web服务器传输超文本到本地浏览器的传送协议，基于客户端服务器架构工作。



请求报文

- 请求方式
- 请求地址

响应报文

- HTTP状态码
    - 200 请求成功
    - 404 请求的资源没有被找到
- 内容类型
    - text/html
    - text/css

请求参数

客户端向服务器端发送请求时，有时需要携带一些客户信息，比如登录操作。客户信息需要通过参数的形式传递到服务器端。

GET传参

参数被放置在地址栏中，格式为：name=zhangsan&age=20

```
// 处理get参数
const url = require('url');
let { query } = url.parse(req.url, true);
```

POST传参

参数被放置在请求体中，格式和GET参数相同。

```
// 处理post参数
// 由于post传递的参数数据量比较大，在网络中并不是一次性传递完成的，而是分成了多次传递
// 所以在接收的时候也需要分为多次接收
// 在NodeJs中接收post参数需要使用事件完成
const querystring = require('querystring');
let formData = '';
req.on('data', chunk => formData += chunk);
req.on('end', () => console.log(querystring.parse(formData)));
```

路由

路由是指URL地址与程序的映射关系，更改URL地址可以改变程序的执行结果。简单说就是请求什么响应什么。

```
const http = require('http');
const url = require('url');
const finalhandler = require('finalhandler')
// 创建网站服务器
const server = http.createServer();
// 当客户端发来请求的时候
server.on('request', (req, res) => {
    // 处理404方法
    var done = finalhandler(req, res);
    // 获取客户端的请求路径
    let { pathname } = url.parse(req.url);
    // 对请求路径进行判断 不同的路径地址响应不同的内容
    if (pathname == '/' || pathname == '/index') {
        res.end('index');
    }else if (pathname == '/list') {
        res.end('list');
    }else if (pathname == '/about') {
        res.end('about');
    }else {
       // 如果以上路由都没有匹配上，则进行404处理
        done();
    }
});
// 程序监听端口
server.listen(3000);
```

客户端请求方式

1. 浏览器地址栏
2. Form表单提交
3. link标签的href属性
4. script标签的src属性
5. image标签的src属性

静态资源处理

服务器端不需要处理，可以直接响应给客户端的资源就是静态资源，例如CSS、JavaScript、image文件。

使用Node.js的第三方模块serve-static处理静态资源的响应。

```
const finalhandler = require('finalhandler');
const http = require('http');
const serveStatic = require('serve-static');
// 静态资源目录
var serve = serveStatic('public');
// 创建web服务器
var server = http.createServer(function onRequest (req, res) {
    // 处理静态资源
    serve(req, res, finalhandler(req, res));
});
server.listen(3000)
```

Node.js工作原理

同步异步

同步或者异步指的是服务器端处理请求的方式。

同步方式

当你来到餐馆,一个指定的服务员被分配给你服务,当你点完餐时,服务员将订单送到厨房并在厨房等待厨师制作菜肴,当厨师将菜肴烹饪完成后,将菜肴送到你的面前,服务完成.此时这个服务员才能服务另外的客人.同步模式就是一个服务员同时只能服务于一个客人.

你可以想象,餐馆就是服务器,服务员就是线程,客户端向服务器端发送的请求就是吃饭的客人.当客户端向服务器发送请求时,服务器端需要为请求分配一个线程处理该请求,在请求的过程中可能会去读取出数据或者文件,这是一个需要时间的操作,就好像厨师炒菜需要时间一样,此时线程只是等待操作的完成.如果此时有新的请求来了,服务器需要分配一个新的线程为该请求服务,必须要知道的是,分配一个线程大约消耗服务器端2MB的内存.想象一下,如果同一时间,服务器接收到大量的请求,服务器可能会消耗掉所有内存提供服务,同一时间再有请求来的时候必须等待,等待有空闲的线程才能为该请求服务,如果不想让该请求等待,服务器端需要增加更多的硬件(内存)设备.这种同步方式显然没有有效的利用现有的资源.

异步方式

当你来到餐馆,在点餐后服务员将你的订单送到厨房,此时服务员没有在厨房等待厨师烹饪菜肴,而是去服务了其他客人,当厨房你的菜肴烹饪好以后,服务员再将菜肴送到你的面前.异步模式就是一个服务员同时可以服务多个客人.

Node就是基于异步的应用程序.在Node中,用一个线程处理所有的请求.当客户端向服务器端发送请求时,服务器端的一个线程处理该请求,在请求的过程中如果去查询数据库或者读取文件,该线程不会等待查询的结果,转而去处理其他请求,Node程序会监控数据库的查询结果,如果查询完成,该线程会重新回来处理该请求.Node擅长处理高并发的网络请求或频繁的数据取读取型应用程序,不会带来硬件成本的增加.这种异步的模式显然有效的利用了现有的资源.

代码执行顺序

同步方式

代码一行一行执行,下一行的代码必须等待上一行代码执行完成以后才能执行.

异步方式

代码在执行过程中某行代码需要耗时,代码的执行不会等待耗时操作完成以后再去执行下一行代码,而是不等待直接向后执行.异步代码的执行结果需要通过回调函数的方式处理.

Promise

Promise出现的目的是解决Node.js异步编程中回调地狱的问题

Promise本身是一个构造函数，要使用promise解决回调地狱的问题 需要使用new运算符创建Promise构造函数的实例对象

在创建对象的时候需要传入一个匿名函数,匿名函数中有两个参数 resolve，reject

```
let promise = new Promise((resolve, reject) => {
    setTimeout(() => {
        if (true) {
            resolve({name: '张三'})
        }else {
            reject('失败了')
        }
    }, 2000);
});
promise.then(result => console.log(result); // {name: '张三'})
       .catch(error => console.log(error); // 失败了)
```

异步函数

异步函数是异步编程语法的终极解决方案，它可以让我们将异步代码写成同步的形式，让代码不再有回调函数嵌套，使代码变得清晰明了

```
const fn = async () => {};
async function fn () {}
```

async

1. 普通函数定义前加async关键字 普通函数变成异步函数
2. 异步函数默认返回promise对象
3. 在异步函数内部使用return关键字进行结果返回 结果会被包裹的promise对象中 return关键字代替了resolve方法
4. 在异步函数内部使用throw关键字抛出程序异常
5. 调用异步函数再链式调用then方法获取异步函数执行结果
6. 调用异步函数再链式调用catch方法获取异步函数执行的错误信息

await关键字

1. await关键字只能出现在异步函数中
2. await promise await后面只能写promise对象 写其他类型的API是不不可以的
3. await关键字可是暂停异步函数向下执行直到promise返回结果

promisify改造函数

```
// 改造现有异步函数api 让其返回promise对象 从而支持异步函数语法
const promisify = require('util').promisify;
// 调用promisify方法改造现有异步API 让其返回promise对象
const readFile = promisify(fs.readFile);
```