# Express MongoDB COS/CDN WEB 服务器

**田世祚 无12**

## 开始之前

开始之前建议电脑上已安装：

1. Node.js
2. npm & yarn
3. Postman
4. MongoDB & MongoDBCompass
5. Docker

## Express

### 1. Express 简介

#### 背景介绍

经过上一课的学习，我们已经了解到Node.js作为浏览器的替代品（实际上是Chrome V8 引擎），为JavaScript/TypeScript提供了运行环境。这也就意味着JavaScript可以在后端服务器上运行，为网站后端开发提供了一种选项。

> ChatGPT老师：
>
> 网站的前端和后端是构成一个完整网站的两个主要部分。
>
> 前端是指用户在浏览器中看到和与之交互的部分。它主要由HTML、CSS和JavaScript组成。HTML用于定义网页的结构和内容，CSS用于样式化网页的外观和布局，JavaScript用于实现网页的交互和动态效果。前端开发人员负责将设计师提供的视觉设计转化为可交互的网页，并确保网站在不同设备和浏览器上的兼容性。
>
> 后端是指网站的服务器端，负责处理前端发送的请求并返回相应的数据。后端开发主要使用服务器端编程语言（如Python、Java、PHP等）和数据库（如MySQL、MongoDB等）来实现网站的业务逻辑和数据处理。后端开发人员负责处理用户的请求、与数据库进行交互、进行数据处理和逻辑运算，并将结果返回给前端。
>
> 前端和后端之间通过网络进行通信，前端发送请求给后端，后端处理请求并返回相应的数据给前端。这种分工使得前端和后端可以独立开发和维护，同时也提高了网站的性能和可扩展性。
>
> 综上所述，前端和后端是网站开发中不可或缺的两个部分，它们共同协作，使得网站能够呈现给用户一个完整、交互性强的体验。
>
> （笔者：我们在文章的最后会介绍Web服务器的基本知识，在了解前端和后端的搭建方式之后，组建一个完整的网站。

不像前端有统一的浏览器标准，后端的开发更加自由，我们可以选择Java， PHP，python，C，Go等语言进行开发。[电子系科协网站]采用TypeScript作为后端开发语言，采用Express快速构建后端应用。

## Why Express?

如想自己动手建立一个简单后端，理论上只需调用http-server之类的库就能接收和发送http请求了。尽管这些库已经帮我们处理了很多底层问题，但是要想实现复杂一些的功能还是相当麻烦的。 Express是一个基于Node.js的开源框架，把接收和发送http请求进行了更高级别、更用户友好的封装。官网上称其“快速、开放、极简”，表明Express在保证性能的同时，代码书写非常容易且是开源的。它能够使用开发者所选择的各种http实用工具和中间件，快速方便地创建强大的API。

## 安装Express

首先需要安装Node.js，官网链接：https://nodejs.org/en/download/。

在Windows系统上，可选择下载安装包或二进制文件并按提示安装。在Linux系统上，可选择下载二进制文件然后自行设置环境变量（nodejs/bin/node、nodejs/bin/npm都应添加到环境变量，或建立软链接至/usr/local/bin），或者在Ubuntu上可以用apt命令安装：

```
sudo apt install nodejs npm
```

> 注：apt下载源更换技巧参见ubuntu | 镜像站使用帮助 | 清华大学开源软件镜像站 | Tsinghua Open Source Mirror。使用如下命令检验是否换源成功：
>
> ```
> sudo apt update
> sudo apt upgrade
> ```

在安装好Node.js后，建议用其npm包管理工具全局安装yarn包管理工具（弥补npm的一些缺陷，不过拉取的包依然来自npm仓库，因此要想搜索库文档，可至https://www.npmjs.com/查询）：

```
sudo npm install -g yarn # 全局安装，在本机所有项目中可用，需要超级用户权限
```

在此基础上，你可以新建一个工作目录，并在其中执行：

```
yarn init
```

此命令将此目录初始化为yarn管理的目录，会自动创建一个package.json文件。在初始化过程中需要输入程序的名称、版本、入口文件等信息，这些都可以直接回车设成默认值（这里配置的入口文件我们不需要用）。下面我们在此目录下安装Express：

```
yarn add express
```

## 准备工作

### 创建后端代码

在刚才创建的安装有Express的工作目录当中，我们新建一个文件夹src，用于保存源码，并在其中新建入口文件app.ts。然后在其中导入Express包并创建Express类的实例：

```
import express from "express"; // 导入Express包
const app = express(); // 创建Express类的实例
```

这样导入的Express包是JavaScript编写的，缺乏TypeScript语法所要求的类型声明文件，因此VS Code会在import那一行出现提醒。我们可以使用如下命令安装类型声明：

```
yarn add @types/express --dev
```

安装完成后，我们看到提醒已经消除。

之后我们让程序监听3000端口，这样发送至http://localhost:3000的请求就能够被我们捕获了。

```
const port = 3000;
app.listen(port, () => { // 开始监听
    console.log(`listening on port ${port}`);
})
```

问题来了，我们如何才能运行后端呢？大家之前可能尝试过用tsc将TypeScript代码转为Javascript代码然后再用node作为解释器运行，这样对于我们日常开发就十分繁琐。Babel工具链能够将较新的JavaScript以及TypeScript语法转化为向后兼容的JavaScript语法，以便运行在当前的浏览器环境当中。

**安装Babel和nodemon**

我们使用Babel来进行语法转换，同时用nodemon工具检测代码改动，在代码改动时为我们重启后端，这就方便了开发调试。

首先，我们安装开发时所需的包：

```
yarn add @babel/core @babel/cli @babel/preset-env @babel/preset-typescript
@babel/node nodemon --dev
```

> @babel/core是Babel的核心库，@babel/cli是Babel的命令行工具，@babel/preset-env是一组将最新JavaScript语法转化的预设工具集，@babel/preset-typescript是一组将TypeScript语法转化的预设工具集，@babel/node可以应用所选的Babel工具并像node一样运行JavaScript代码，nodemon可以检测代码修改并自动重启程序。
>
> --dev 选项使得安装的依赖包只有在开发环境中才用，在生产环境下不会包含这些包。

其次，我们对Babel进行配置。在工作目录下新建`babel.config.json`文件，并向其中写入：

```
{
    "presets": [
        "@babel/preset-env",
        "@babel/preset-typescript"
    ]
}
```

> presets用于指定要使用的工具集。更多配置详见官方文档：https://www.babeljs.cn/docs/

然后，我们对nodemon进行配置。打开工作目录下的`package.json`文件，向其中新建一个如下字段：

```
"nodemonConfig": {
    "watch": [
        "src"
    ],
    "ext": "ts, json",
    "exec": "babel-node --extensions \".ts\" src/app.ts"
}
```

> 更多配置详见官方文档：https://www.npmjs.com/package/nodemon
>
> watch字段：监听的文件目录
>
> ext字段：监听的文件后缀

> exec字段：当运行nodemon命令时执行此字段中的命令。 babel-node与node的区别在于应用了babel.config.json中配置的工具，src/app.ts是你的入口文件

最后，我们设定yarn的执行脚本以简单运行node_modules中的程序。在`package.json`文件中再新建一个如下字段：

```
"scripts": {
    "start": "nodemon"
}
```

> "start": "nodemon" 等价于./node_modules/nodemon/bin/nodemon.js }

**后端，启动**

只要我们在终端中输入`yarn start`，就会看到nodemon启动并用Babel生成和运行了 JavaScript代码。此时依次输入`r` `s` `Enter`，会使后端重启；摁下`Ctrl`+`C`，会使后端停止。

恭喜您成功启动了后端！

### 安装Postman

Postman是一个API调试工具，可以模拟前端向后端发送请求，这在我们开发后端时非常有用。大家可以进入其官网注册账号（好处是你的HTTP请求会被备份并同步在不同设备）https://www.postman.com/，然后点击download desktop app相关选项或者直接链接到https://www.postman.com/downloads/，下载安装程序并安装。

打开Postman并登录后，可以在Collections中点击加号，新建一个Collection，然后在上方栏中点击加号，新建一个Request并保存到这个Collection。这样就可以编辑、保存和发送这个Request了。针对同一类API的Request可以保存在同一个Collection下以方便管理。建议大家阅读Postman官方文档（https://learning.postman.com/docs/getting-started/introduction/）以学习更多Postman的实用知识。

（演示一下操作）

## 2. Express 路由

### 何为路由?

路由决定了后端API响应前端请求的方式，通过解析URI和HTTP请求方法来实现。在Express中，可以非常方便地在路由匹配时执行一个或多个处理函数并给出响应。简单地说，当前端向后端发送HTTP请求时，Express根据URI和HTTP请求方法来匹配相应的API，之后执行由开发者设计的处理程序，最后将响应发送回前端。

### HTTP请求基础

相信大家在几天前的“通信基础”讲座上已经掌握了关于HTTP通信的一些基础知识。下面列出一些资料供大家参考：

**客户端请求格式**


**服务器响应格式**


**HTTP请求方法**

Express根据的是HTTP请求方法和URI/URL来进行路由，由开发者编写的API处理请求后将状态码和其他响应内容发回客户端。下面先来了解一下HTTP请求方法：

| 方法 | 描述 |
| --- | --- |
| GET | 向指定资源发出请求，用于获取资源，数据被包含在URL中。 |
| HEAD | 等同于向服务器发送GET请求，但不获取响应包体，只获取响应头。 |
| POST | 向指定资源提交数据进行处理请求（例如提交表单或者上传文件），数据被包含在请求体中。 |
| PUT | 向指定资源处上传其最新内容。 |
| DELETE | 请求服务器删除Request-URL所标识的资源。 |
| CONNECT | HTTP/1.1 协议中预留给能够将连接改为管道方式的代理服务器。 |
| OPTIONS | 返回服务器针对特定资源的支持，允许客户端查看服务器的性能。 |
| TRACE | 回显服务器收到的请求，主要用于测试或诊断。 |
| PATCH | 是对PUT方法的补充，用来对已知资源进行局部更新。 |

#### 状态码

服务端会对客户端请求做出响应，通过状态码告知客户端请求是否成功/错误原因是什么。下面是一些常用的状态码：

| 状态码 | 英文描述 | 中文描述 |
| --- | --- | --- |
| 200 | OK | 请求成功 |
| 400 | Bad Request | 客户端请求的语法错误，服务器无法理解 |
| 401 | Unauthorized | 客户端请求的身份认证缺失或有误，认证失败 |
| 403 | Forbidden | 服务器理解请求客户端的请求，但拒绝执行此请求 |
| 404 | Not Found | 服务器无法根据客户端的请求找到资源（网页） |
| 500 | Internal Server Error | 服务器内部错误，无法完成请求 |
| 501 | Not Implemented | 服务器不支持请求的功能，无法完成请求 |
| 502 | Bad Gateway | 作为网关或代理工作的服务器尝试执行请求时，从远程服务器接收到了一个无效的响应 |

#### HTTP URL

URL = uniform resource locator，即统一资源定位系统，是在网络上标识地址的方法。具体在HTTP协议的应用上，形式如下：

`http://[host]:[port]/[path]?[searchpart]`

其中，`host`是主机地址，端口`port`在http协议中默认为80（https协议默认为443），`path`是路径中的参数，`searchpart`是查询字符串（GET请求使用的参数）。

例如：

https://www.bing.com/search?q=%E7%94%B5%E5%AD%90%E7%B3%BB%E8%BD%AF%E4%BB%B6%E4%B8%AD%E5%BF%83&mkt=zh-CN

协议为https，主机地址为`www.bing.com`，端口为默认的443，请求路径为`search`，查询字符串为`q=%E7%94%B5%E5%AD%90%E7%B3%BB%E8%BD%AF%E4%BB%B6%E4%B8%AD%E5%BF%83&mkt=zh-CN`，包含`q=%E7%94%B5%E5%AD%90%E7%B3%BB%E8%BD%AF%E4%BB%B6%E4%B8%AD%E5%BF%83`以及`mkt=zh-CN`两个参数。

> 把这个URL丢到Postman里，选择GET请求，可以在Postman界面里直观看到查询字符串等信息，按下Send会收到一个HTML页面。

## Express 路由教程

#### 路由的基本用法

在准备工作完成的基础上，我们开始运用Express编写后端。路由的基本格式是：

```
app.METHOD(PATH, HANDLER)
```

其中：

- app 是 express 类的实例。
- METHOD 是HTTP请求方法。
- PATH 是服务器上的路径。
- HANDLER 是在路由匹配时执行的函数。

例如：

```
// “/”路径接收到的GET方法匹配至该路由
app.get('/', (req, res) => {
    res.status(200).send('GET request');
})
// “/”路径接收到的POST方法匹配至该路由
app.post('/', (req, res) => {
    res.status(200).send('POST request');
})
// “/”路径接收到的所有方法匹配至该路由
app.all('/', (req, res) => {
    res.status(200).send('Any request');
})
```

对于Handler函数，Express会在路由匹配时调用，调用时传进的参数有三个（一般依次命名为req、res、next）：req对象代表“请求”，常用的属性是body（请求的包体）、query（请求的查询字符串）和params（请求的路径参数）；res对象代表“响应”，常用的方法是status（设定响应状态码）和send（设定响应内容）；next函数在调用时表示移交给下一个中间件进行处理。

如果请求匹配了前面的路由，则不会再去匹配后面的路由。在写好代码并启动后端后，大家就可以打开Postman，向http://localhost:3000/发送不同类型的请求并查看响应。

**路由的匹配路径**

匹配路径可以是普通字符串、字符串模板和正则表达式。

普通字符串：

```
app.get('/', (req, res) => { // 可匹配“/”
    res.send('root');
})
app.get('/about', (req, res) => { // 可匹配“/about”
    res.send('about');
})
app.get('/random.text', (req, res) => { // 可匹配“/random.text”
    res.send('random.text');
})
```

字符串模板：

```
app.get('/ab?cd', (req, res) => { // 可匹配“acd”或“abcd”
    res.send('ab?cd');
})
app.get('/ab+cd', (req, res) => { // 可匹配“abcd”,“abbcd”,“abbbcd”等
    res.send('ab+cd');
})
app.get('/ab*cd', (req, res) => { // 可匹配“abcd”,“abxcd”,“abRANDOMcd”,“ab123cd”等
    res.send('ab*cd');
})
app.get('/ab(cd)?e', (req, res) => { // 可匹配“abe”或“abcde”
    res.send('ab(cd)?e');
})
```

正则表达式：

```
app.get(/a/, (req, res) => { // 可匹配任何带有‘a’的路径
    res.send('/a/');
})
app.get(/.*fly$/, (req, res) => { // 可匹配“butterfly”,“dragonfly”
    res.send('/.*fly$/'); // 但不可匹配“butterflyman”,“dragonflyman”
}) // 相当于匹配以“fly”结尾的所有路径
```

> 正则表达式十分有用。可参考：正则表达式 - 教程 | 菜鸟教程 (runoob.com)

**路径中的参数与查询字符串**

路径中的参数（route parameter）是指将参数直接作为URL路径一部分的传参方式，而查询字符串（query string）是URL在路径后面用问号分割的部分，二者虽然都是在URL中传递参数，但用法不同。

以这个网址为例（假如我们是百度百科的后端开发人员）：

https://baike.baidu.com/item/%E7%BB%9F%E4%B8%80%E8%B5%84%E6%BA%90%E5%AE%9A%E4%BD%8D%E7%B3%BB%E7%BB%9F/5937042?fromtitle=url&fromid=110640

可以通过以下路由解析URL中的路径参数和查询字符串：

```
app.get('/item/:param1/:param2', (req, res) => {
    console.log(req.params.param1); // 那一长串%E7...，实际上是UTF-8编码的中文字符，转换过来是“统一资源定位系统”
    console.log(req.params.param2); // 5937042
    console.log(req.query.fromtitle); // url
    console.log(req.query.fromid); // 110640
    res.status(200).send('ok');
})
```

此外还有一些高级用法，如在路径中插入短线(`-`)和点(`.`)，甚至在路径中引入正则表达式：

```
// Request URL: http://localhost:3000/flights/LAX-SFO
app.get('/flights/:from-:to', (req, res) => {
    console.log(req.params.from); // LAX
    console.log(req.params.to); // SFO
    res.status(200).send('ok');
})
// Request URL: http://localhost:3000/date/2020.2.1
app.get('/date/:year.:month.:day', (req, res) => {
    console.log(req.params.year); // 2020
    console.log(req.params.month); // 2
    console.log(req.params.day); // 1
    res.status(200).send('ok');
})
```

**路由处理函数**

Express的路由处理函数十分灵活，可以传递多个函数、函数数组或二者的混合。当需要从上一个处理函数过渡到下一个处理函数时，需调用传入的next函数。

```
var cb0 = function (req, res, next) {
    console.log('CB0')
    next()
}
var cb1 = function (req, res, next) {
    console.log('CB1')
    next()
}
app.get('/example/d', [cb0, cb1], function (req, res, next) {
    console.log('the response will be sent by the next function ...')
    next()
}, (req, res) => {
    res.send('Hello from D!')
})
```

**Router**

express.Router类可以用于构建模块化的路由处理函数（中间件）。我们在src文件夹下面新建一个route1.ts文件，然后写入：

```
import express from "express";
const router = express.Router(); // 实例化express.Router类
router.get("/", (req, res) => {
    res.status(200).send("ok!");
})
export default router; // 导出后，这个Router就成为了一个中间件
```

然后修改app.ts类：

```
import express from "express";
import route1 from "./route1"; // 导入route1中间件
const app = express();
const port = 3000;
app.use("/route1", route1); // 在route1路径下使用route1中间件
app.listen(port, () => {
    console.log(`Example app listening on port ${port}`)
})
```

## 3. Express中间件

中间件是模块化的路由处理函数，我们刚才已经看到了路由级中间件的用法（通过express.Router创建中间件，然后通过app.use导入中间件），此外广义上刚才讲过的所有路由处理函数也都可以视为是最普通的中间件。在创建和使用中间件时，还有以下注意事项：

用app.use加载的中间件无论什么HTTP请求方法都会调用，如想限定一种HTTP请求方法，需使用app.METHOD。同时，如果没有给出路径参数，则默认任何路径的请求都会调用中间件。

中间件是按顺序加载和执行的，如果前面已经出现了匹配的路由，那么后面的路由即使匹配，也不会调用相应的中间件。next函数默认将控制权移交给当前路由的下一个处理函数，如想将控制权移交给下一个路由，需调用next('route')。如果将非"route"的参数传递给了next函数，则Express会认为该中间件出错，从而越过其他的正常处理函数直接执行异常处理函数。

```
app.get('/user/:id', (req, res, next) => {
    // if the user ID is 0, skip to the next route
    if (req.params.id === '0') next('route');
    // otherwise pass the control to the next middleware function in this stack
    else next()
}, (req, res, next) => {
    // send a regular response
    res.send('regular');
})
// handler for the /user/:id path, which sends a special response
app.get('/user/:id', (req, res, next) => {
    res.send('special');
})
```

**异常处理**

对于同步编程的中间件，当抛出一个异常时且没有自建异常处理函数时，Express会自己来处理这个异常。Express会根据错误的`err.status`(或`err.statusCode`)来设定状态码、错误信息等。我们也可以自己创建一个异常处理函数，如果为Express中间件传入四个参数，通常命名为`(err, req, res, next)`，则Express会把这一中间件当作异常处理函数。

```
app.get('/', (req, res) => {
    throw new Error('error occurs!'); // 异常由Express自建异常处理函数来处理
})
```

```
app.use((err, req, res, next) => {
    res.status(500).send('error occurs!') // 或者可以自己编写异常处理函数
})
```

对于异步编程的中间件，要想进行异常处理，需要调用`next`函数并向其中传递一个值。如果中间件返回的是一个`Promise`，则会自动调用`next(value)`。下面的异步函数返回`Promise`，如果中间抛出了异常或者`reject`，则Express会默认调用`next`函数并传入异常信息。

> 此特性从Express5开始：Starting with Express 5, route handlers and middleware that return a Promise will call `next(value)` automatically when they reject or throw an error.
>
> 目前Express5仍处于beta阶段，可以通过yarn add express@5 获取。科协后端使用的Express版本为4.18.2，因此不能演示这一特性。详询Express error handling - Express 中文文档 | Express 中文网 (expressjs.com.cn)。

```
app.get('/user/:id', async (req, res, next) => {
    var user = await getUserById(req.params.id);
    res.send(user);
})
```

更多时候，我们在中间件内部来处理异常，而不用将异常传递给Express：

```
app.get("/", async (req, res) => {
    try{
    ...
    return res.status(200).send("ok");
    } catch(err) {
    ...
    return res.status(500).send("Internal Server Error");
    }
})
```

#### 常用中间件

Express提供了一些内建中间件供用户选择，比如常用的express.json会以json格式来解析请求体，express.urlencoded会以urlencoded格式来解析请求体。还有一些第三方的中间件可以通过包管理器来安装，如cookie-parser，又如处理前端跨域访问请求的cors。

# MongoDB

> 以下内容主要引用自MongoDB 教程 | 菜鸟教程 (runoob.com)。想了解更多相关内容有请菜鸟教程。

## 1. MongoDB简介

### 下载与安装

可以在官网上下载并安装MongoDB：https://www.mongodb.com/try/download/community。也可以在Docker虚拟环境中启动一个MongoDB，一键启动，没有配ip的烦恼。

### 什么是MongoDB?

大家已经学习过了Hasura，知道Hasura引擎可以让用户使用非关系型的GraphQL来操作关系型数据库如PostgreSQL。今天要学习的MongoDB介于关系型和非关系型数据库之间，采用的查询语言是不同于关系型数据库的，也称为NoSQL( = Not Only SQL)。NoSQL并不是统一的标准，不同的数据库实现并不一样，MongoDB采用类似json格式的“文档”存储。这样也就有机会对某些字段建立索引，实现关系数据库的某些功能。

#### MongoDB的基础概念

在MongoDB当中，mongod程序是数据库的主守护进程。它处理数据请求，管理数据访问，并执 行后台管理操作。在启动MongoDB服务，也就是运行mongod时，必须设定数据库地址的参数“--dbpath”，否则会启动失败。

Windows环境下，在MongoDB所在路径（例如`C:\Program Files\MongoDB\Server\5.0\bin`）打开cmd，输入

```
mongod.exe --dbpath "your/custom/database/path"
```

MongoDB既有命令行的管理工具，也有图形界面的管理工具。MongoDBCompass 是一个图形界面管理工具，我们可以到官网下载安装，下载地址：https://www.mongodb.com/download-center/compass。MongoDB的命令行工具采用JavaScript语言。在Windows系统上，打开mongo.exe以进入命令行界面连接MongoDB服务；在Linux系统上，运行mongodb安装目录的下的bin目录中的mongo文件以连接MongoDB服务。

可以将MongoDB的概念同我们熟悉的SQL数据库的概念作类比：

| **SQL概念** | **MongoDB概念** |
| --- | --- |
| database（数据库） | database（数据库） |
| table（表） | collection（集合） |
| row（行） | document（文档） |
| column（列） | field（字段或域） |
| primary key（主键） | primary key（主键） |

在一个MongoDB应用上可以建立很多的数据库，MongoDB有一些保留的数据库名具有特殊作用， 如admin数据库（root权限）。集合就是MongoDB文档组，一个数据库中可以有若干集合（类比数据表），集合没有 固定的结构，里面可以存储任意文档，应当把有关联的文档放入同一集合当中。文档是由一组键值对所 组成的，可看作是一条记录，不同文档间不需要设置相同的字段，并且相同的字段不需要相同的数据类型。因此，尽管MongoDB的概念同SQL数据库的概念可以类比，但二者间有着非常明显的区别。

> 更详细信息可参考：
>
> MongoDB 概念解析 | 菜鸟教程 (runoob.com)

---

**MongoDB数据类型**

MongoDB的数据类型很多，我们只列举常见的几种：

| 数据类型 | 描述 |
| --- | --- |
| String | 字符串。存储数据常用的数据类型。在 MongoDB 中，UTF-8 编码的字符串才是合法的。 |
| Integer | 整型数值。用于存储数值。根据你所采用的服务器，可分为 32 位或 64 位。 |
| Boolean | 布尔值。用于存储布尔值（真/假）。 |
| Double | 双精度浮点值。用于存储浮点值。 |
| Min/Max keys | 将一个值与 BSON（二进制的 JSON）元素的最低值和最高值相对比。 |
| Array | 用于将数组或列表或多个值存储为一个键。 |
| Timestamp | 时间戳。记录文档修改或添加的具体时间。 |
| Object | 用于内嵌文档。 |
| Null | 用于创建空值。 |
| Symbol | 符号。该数据类型基本上等同于字符串类型，但不同的是，它一般用于采用特殊符号类型的语言。 |
| Date | 日期时间。用 UNIX 时间格式来存储当前日期或时间。你可以指定自己的日期时间：创建 Date 对象，传入年月日信息。 |
| Object ID | 对象 ID。用于创建文档的 ID。 |
| Binary Data | 二进制数据。用于存储二进制数据。 |
| Code | 代码类型。用于在文档中存储 JavaScript 代码。 |
| Regular expression | 正则表达式类型。用于存储正则表达式。 |

一个重要的类型`ObjectId`：相当于数据库的主键。它的结构如下：

- 前 4 个字节表示创建 **unix** 时间戳,格林尼治时间 **UTC** 时间，比北京时间晚了 8 个小时
- 接下来的 3 个字节是机器标识码
- 紧接的两个字节由进程 id 组成 PID
- 最后三个字节是随机数


## 2. MongoDB基础操作

> 这一部分没有什么特别值得注意的地方，相信大家看看文档自己操作一下也就会了。略去这部分内容不会对我们后续的学习造成任何影响。

**查看数据库**

查看当前MongoDB应用上的全部数据库的语法格式为：`show dbs`

查看当前所在的数据库名的语法格式为：`db`

**创建和切换数据库**

用use命令时，如果数据库不存在，则创建数据库，否则切换到指定数据库。语法格式：

```
use [database name]
```

MongoDB中默认的数据库为test，如果你没有创建新的数据库，集合将存放在test数据库中。并且，如果新创建的数据库中没有数据，则并不会被真正创建，只有在插入数据后才会被真正创建。

**删除数据库**

删除当前所在的数据库的语法格式为：

```
db.dropDatabase()
```

**查看集合**

查看当前所在的数据库当中的全部集合的语法格式为：

```
show collections
```

**创建集合**

创建集合的语法格式为：

```
db.createCollection([collection name], [options])
```

options参数以对象的形式传入，可以选用的参数如：

- **capped**（可选）：如果为true，则创建固定集合。固定集合是指有着固定大小的集合，当达到最大值时，它会自动覆盖最早的文档。
- **size**（可选）：为固定集合指定一个最大占用空间（单位：字节）。
- **max**（可选）：为固定集合指定包含文档的最大数量。

在MongoDB中，其实不需要手动创建集合。当你向不存在的集合中插入文档时，MongoDB会自动创建集合。

**删除集合**

删除指定集合的语法格式为：

```
db.[collection name].drop()
```

**查看文档**

在MongoDB中，使用find方法以非结构化的方式来查看所有文档，语法格式为：

```
db.[collection name].find([query], [projection])
```

- **query** （可选）：使用查询操作符（对象）指定筛选条件。（支持正则表达式）
- **projection** （可选）：使用投影操作符（对象）指定返回的键。查询时返回文档中所有键值， 只需省略该参数即可。

还可以使用 pretty() 方法来使获取的数据更易读，语法格式为：

```
db.[collection name].find([query], [projection]).pretty()
```

查看文档时，筛选条件有如下格式：

| 操作 | 格式 | 范例 | 对应SQL语句 |
| --- | --- | --- | --- |
| 等于 | `{<key>:<value>}` | `db.col.find({"key":"50"}).pretty()` | `where key= '50'` |
| 小于 | `{<key>:{$lt:<value>}}` | `db.col.find({"likes":{$lt:50}}).pretty()` | `where likes < 50` |
| 小于或等于 | `{<key>:{$lte:<value>}}` | `db.col.find({"likes":{$lte:50}}).pretty()` | `where likes <= 50` |
| 大于 | `{<key>:{$gt:<value>}}` | `db.col.find({"likes":{$gt:50}}).pretty()` | `where likes > 50` |
| 大于或等于 | `{<key>:{$gte:<value>}}` | `db.col.find({"likes":{$gte:50}}).pretty()` | `where likes >= 50` |
| 不等于 | `{<key>:{$ne:<value>}}` | `db.col.find({"likes":{$ne:50}}).pretty()` | `where likes != 50` |
| 与 | `{<key1>:..., <key2>:...}` | `db.col.find({"key1":100,"key2":200})` | `...AND...` |
| 或 | `{$or:[{<key1>:...}.{<key2>:...}]}` | `db.col.find({$or:[{"key1":100},{"key2":200}]})` | `...OR...` |

**创建文档**

创建或插入文档的语法格式如下：

`db.[collection name].insert([document])`

若插入的数据主键已经存在，则会抛主键重复异常，不保存当前数据。document参数可以是单个文档（json格式对象），也可以是多个文档（json格式对象数组）。

特别地，向集合插入一个新文档的语法格式如下：

`db.[collection name].insertOne([document])`

向集合一次性插入多个文档的语法格式如下：

```
db.[collection name].insertMany(
    [ [document 1] , [document 2], ... ],
    {
        ordered: [boolean]
    }
)
```

参数：

- **ordered**：指定是否按顺序写入，默认true，按顺序写入。

**更新文档**

更新文档的语法格式如下：

```
db.[collection name].update(
    [query],
    [update],
    {
         upsert: [boolean],
         multi: [boolean],
         writeConcern: [document]
    }
)
```

**参数说明：**

- **query** : update的查询条件，类似sql update查询内where后面的。
- **update** : update的对象和一些更新的操作符（如,inc...）等，也可以理解为sql update查询内set后面的
- **upsert** : 可选，这个参数的意思是，如果不存在update的记录，是否插入objNew,true为插入，默认是false，不插入。
- **multi** : 可选，mongodb 默认是false,只更新找到的第一条记录，如果这个参数为true,就把按条件查出来多条记录全部更新。
- **writeConcern** :可选，抛出异常的级别。

默认情况下update会使用新文档覆盖旧文档，如果不想覆盖而是仅仅想更新其中的某些字段，那么 我们就需要使用update的更新操作符：

以下是将操作符、格式和描述转换为表格形式：

| 操作符 | 格式 | 描述 |
| --- | --- | --- |
| `$set` | `{$set: {field: value}}` | 指定一个键并更新值，若键不存在则创建 |
| `$unset` | `{$unset: {field: 1}}` | 删除一个键 |
| `$inc` | `{$inc: {field: value}}` | 对数值类型进行增减 |
| `$rename` | `{$rename: {old_field: new_field}}` | 修改字段名称 |
| `$push` | `{$push: {field: value}}` | 将数值追加到数组中，若数组不存在则会进行初始化 |
| `$pushAll` | `{$pushAll: {field: value_array}}` | 追加多个值到一个数组字段内 |
| `$pull` | `{$pull: {field: value}}` | 从数组中删除指定的元素 |
| `$addToSet` | `{$addToSet: {field: value}}` | 添加元素到数组中，具有排重功能 |
| `$pop` | `{$pop: {field: 1}}` | 删除数组的第一个或最后一个元素 |

**删除文档**

```
db.[collection name].remove(
    [query],
    [justOne]
)
```

参数:

- **query** :（可选）删除的文档的条件。
- **justOne** :（可选）如果设为 true 或 1，则只删除一个文档，如果不设置该参数，或使用默认值 false，则删除所有匹配条件的文档。
- **writeConcern** :（可选）抛出异常的级别。

删除指定集合下的全部文档:

```
db.[collection name].deleteMany({})
```

删除 key 等于 value 的全部文档:

```
db.[collection name].deleteMany({ key : "value" })
```

删除 key 等于 value 的一个文档:

```
db.[collection name].deleteOne( { key: "value" } )
```

建议在删除文档以前，先执行find进行查询，以防误删文档。

# 3. MongoDB数据处理方法

> 这一部分依然没有什么特别值得注意的地方，相信大家看看文档自己操作一下也就会了。略去这部分内容不会对我们后续的学习造成任何影响。

**limit方法**

limit方法可用于在MongoDB中读取指定数量的数据记录，传入limit()方法的数字参数指定了要从MongoDB中读取的记录条数。limit方法的语法格式如下：

```
db.[collection name].find([query], [projection]).limit([number])
```

**skip方法**

skip方法可用于在MongoDB中跳过指定数量的数据记录，传入skip()方法的数字参数指定了要从MongoDB中跳过的记录条数。skip方法的语法格式如下：

```
db.[collection name].find([query], [projection]).skip([number])
```

**sort方法**

sort方法可用于在MongoDB中对查询到的数据进行排序，可以通过参数指定排序的字段，并使用 1 和 -1 来指定排序的方式，其中 1 为升序排列，而 -1 是用于降序排列。

**aggregate方法**

aggregate方法可用于“聚合”MongoDB中的数据，实际上就是对数据进行统计，语法格式如下：

```
db.[collection name].aggregate([aggregate operation])
```

常用的操作如下：

| 表达式 | 描述 | 示例 |
| --- | --- | --- |
| `$sum` | 计算总和。 | `db.mycol.aggregate([{$group: {_id: "$by_user", num_tutorial: {$sum: "$likes"}}}])` |
| `$avg` | 计算平均值。 | `db.mycol.aggregate([{$group: {_id: "$by_user", num_tutorial: {$avg: "$likes"}}}])` |
| `$min` | 获取集合中所有文档对应值的最小值。 | `db.mycol.aggregate([{$group: {_id: "$by_user", num_tutorial: {$min: "$likes"}}}])` |
| `$max` | 获取集合中所有文档对应值的最大值。 | `db.mycol.aggregate([{$group: {_id: "$by_user", num_tutorial: {$max: "$likes"}}}])` |
| `$push` | 将值加入一个数组中，不会判断是否有重复的值。 | `db.mycol.aggregate([{$group: {_id: "$by_user", url: {$push: "$url"}}}])` |
| `$addToSet` | 将值加入一个数组中，会判断是否有重复的值，若相同的值在数组中已经存在了，则不加入。 | `db.mycol.aggregate([{$group: {_id: "$by_user", url: {$addToSet: "$url"}}}])` |
| `$first` | 根据资源文档的排序获取第一个文档数据。 | `db.mycol.aggregate([{$group: {_id: "$by_user", first_url: {$first: "$url"}}}])` |
| `$last` | 根据资源文档的排序获取最后一个文档数据。 | `db.mycol.aggregate([{$group: {_id: "$by_user", last_url: {$last: "$url"}}}])` |

## 4. Mongoose入门（重点）

> 建议阅读：Mongoose v7.4.0: Getting Started (mongoosejs.com)。中文文档更新较慢（而且广告多），不过入门部分以及相关概念解释也差不多。

由于在网络应用中我们不能总是手动操作MongoDB，因此我们需要学习MongoDB在不同开发环境下的API——Mongoose.

接下来我们走一个简单的Guidance来了解Mongoose的使用方法。

**步骤1：安装Mongoose**

我们用yarn包管理器来安装MongoDB在Node.js下的API。

```
yarn add mongoose
```

**步骤2：连接到MongoDB数据库**

在您的应用程序中，首先需要连接到MongoDB数据库。在您的代码文件中，导入Mongoose并使用`connect`方法连接到数据库。例如：

```
import mongoose from 'mongoose';

try {
    await mongoose.connect('mongodb://localhost/mydatabase', {
        useNewUrlParser: true,
        useUnifiedTopology: true
    });
    console.log('Connected to MongoDB');
} catch (error) {
    console.error('Error connecting to MongoDB:', error);
}
```

在上面的示例中，我们使用`mongoose.connect`方法连接到名为`mydatabase`的本地MongoDB数据库。您可以根据自己的设置更改数据库的连接URL。

**步骤3：定义模式(Schema)和模型(model)**

Mongoose中，一切始于Schema。我们创建一个名为`userSchema`的模式，具有`name`和`email`字段：

```
const userSchema = new mongoose.Schema({
    name: String,
    email: String
});
```

我们从`userSchema`创建一个叫做`User`的模型（对应在MongoDB文档中的名字也叫User）：

```
const User = mongoose.model('User', userSchema);
```

在上面的示例中，我们使用`mongoose.Schema`方法定义了一个名为`userSchema`的模式，并使用`mongoose.model`方法将其与`User`模型关联。

> 关于模式(Schema)和模型(Model)：
>
> 官方表述："*Models are fancy constructors compiled from `Schema` definitions. An instance of a model is called a document.*"
>
> Schema是对文档结构和字段的一种描述或者声明，它经过"编译(Compile)"变成Model。Model是一个构造函数，Model构造出的实例就是一个Document。个人理解，如果按照MongoDB的概念层次划分，Model对应的是MongoDB的Collection层次，Model的实例就是document层次，Schema实际上是对field层次的描述。
>
> 通常，我们使用Model进行数据库的增删查改操作。

**步骤4：使用模型进行数据库操作**

以下是一些常见的示例：

```
try {
    // 创建文档
    const newUser = new User({
        name: 'John Doe',
        email: 'john@example.com'
    });
    await newUser.save();
    console.log('User created');

    // 读取文档
```

```
    const users = await User.find();
    console.log('Users:', users);

    // 更新文档
    await User.updateOne({ name: 'John Doe' }, { email: 'newemail@example.com'
});
    console.log('User updated');

    // 删除文档
    await User.deleteOne({ name: 'John Doe' });
    console.log('User deleted');
} catch (error) {
    console.error('Error performing database operations:', error);
}
```

可以登录MongoDBCompass验证这些文档操作有没有被完成。

**步骤5：关闭数据库连接**

在应用程序结束时，确保关闭与数据库的连接。代码文件中添加以下代码：

```
try {
    await mongoose.connection.close();
    console.log('Disconnected from MongoDB');
} catch (error) {
    console.error('Error disconnecting from MongoDB:', error);
}
```

这将关闭与MongoDB数据库的连接。

> 恭喜！我们已经完成了今天学习的前半部分，不容易啊。休息一下吧。

# COS（对象存储服务OSS）

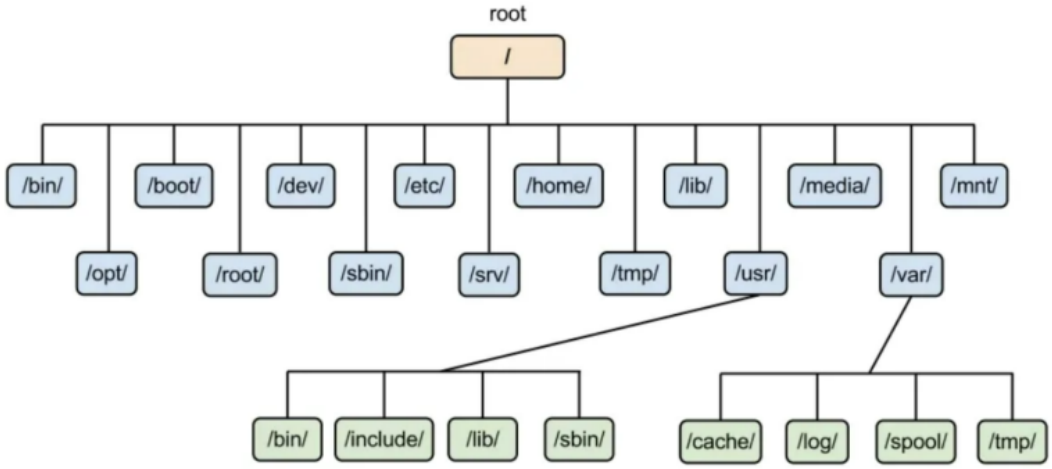
## 1. 简介

> 网站存储已经从阿里云OSS改成了腾讯云COS，本质上仍然是对象存储服务OSS，因此以下仍然介绍OSS相关内容。内容参考：对象存储数据处理_COS数据处理_数据处理方案-腾讯云(tencent.com)

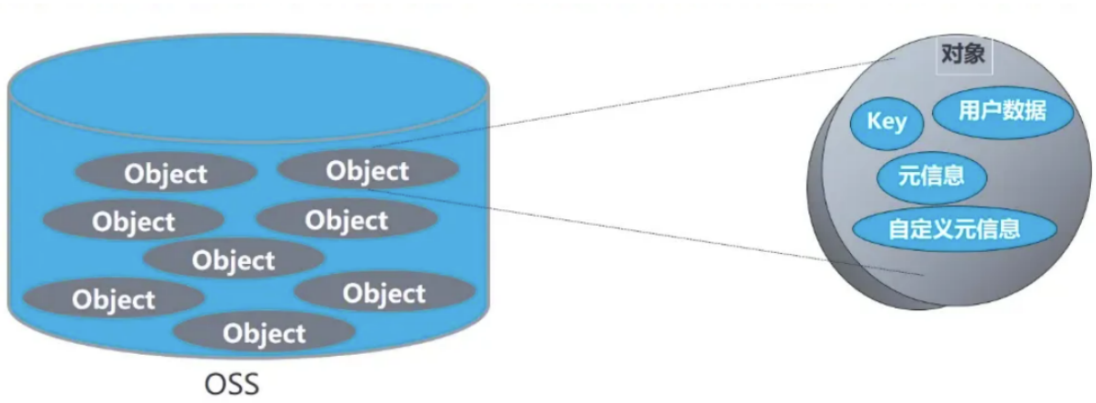
OSS = Object Storage Service，即对象存储服务。如果非要下正式定义，那么对象存储是一种使用RESTful API 存储和检索非结构化数据和元数据对象的工具。如果用正常语言来解释，可以把OSS类比为云盘，只不过我们可以通过网络请求进行下载、上传等操作。

所谓“非结构化数据和元数据”，就是说OSS将数据和元数据（描述其他数据的数据，目的是便于查 找等）封装到对象中，而这些对象存储在平面结构或地址空间中。每个对象都分配一个对象ID（唯一标 识符），使它们可以从单个存储库中检索。这就跟平常存储中结构化的数据很不一样。

Linux文件系统是这样的树状结构（结构化的）：




OSS文件系统是这样的扁平结构（非结构化的）：




**各大公有云厂商对对象存储的定义**

很多云存储厂商都提出了他们自己的对象存储服务：亚马逊（AWS S3），腾讯云（COS），阿里云（OSS），华为云（OBS），百度云（BOS），网易（NOS），快快云（OSS）。

**OSS与云盘的区别**

除了数据组织方式不同之外，OSS与云盘还有诸多区别：

- 一般OSS按实际存储量计费，云盘按购买的大小计费，实际上OSS是灵活的分布式存储，一般没有容量限制
- OSS需要调用API访问，一般不直接挂载到系统里面使用，云盘要挂载到系统里面直接使用
- OSS可以设置外网直接访问，云盘只能通过服务器访问
- 以腾讯云的产品为例，COS比云盘便宜（COS的计费方式决定了它在我们的应用场景下更便宜）

**OSS的优势**

- 提供 RESTful API 接口，丰富的SDK工具，还有客户端和控制台
- 可扩展性强，一般不会出现“满了”的情况
- 由于冗余存储等机制，数据可靠性非常高，腾讯云COS号称可提供99.9999999999%的数据持久性
- 支持内容分发网络CDN加速
- 支持流式写入和读出

**OSS的劣势**

- 无法在其上建立数据库，因为维护起来成本极高。这是由于对象存储不允许只更改一个数据对象的一部分，要想更改对象，哪怕只是重命名，也只能先删除再上传，而在普通文件系统当中可以轻松修改文件的一行。因此，OSS只适用于存储静态数据，如网站上的图片、视频、博客等等。

- 一般来说，操作系统无法像常规磁盘一样安装对象存储。尽管在控制台上OSS似乎跟一块云硬盘一样，但是它不能直接挂载到系统上。

考虑到科协网站目前应用COS对象存储服务，以下介绍均基于腾讯云COS相关文档。

# 2. 概念介绍

> 以下内容均源自腾讯云官方文档。

## 存储桶（Bucket）

存储桶（Bucket）是对象的载体，可理解为存放对象的“容器”。用户可以通过腾讯云控制台、API、SDK 等多种方式管理存储桶以及配置属性。例如，配置存储桶用于静态网站托管、配置存储桶的访问权限等。

## 对象（Object）

对象（Object）是对象存储的基本单元，对象被存放到存储桶中（例如一张照片存放到一个相册）。用户可以通过腾讯云控制台、API、SDK 等多种方式管理对象。在 API、SDK 示例中，对象的命名格式为 <ObjectKey>。

## 对象键（ObjectKey）

腾讯云 COS 中的对象需具有合法的对象键，对象键（ObjectKey）是对象在存储桶中的唯一标识。 例如：在对象的访问地址`examplebucket-1250000000.cos.ap-guangzhou.myqcloud.com/folder/picture.jpg` 中，对象键为 `folder/picture.jpg`。

> 这里的对象键形式上像某个文件结构目录，但是事实上它并不存在真实的文件结构，我们也无法直接获取名为folder的“文件夹”的大小（事实上不存在这个文件夹）。当然你可以自己手动把所有前缀为folder的对象加起来，但是它并不代表本质。

## 地域（Region）

地域是腾讯云托管机房的分布地区，对象存储的数据存放在这些地域的存储桶中。

## 访问域名（Endpoint）

当您将文件上传至存储桶后，对象存储（Cloud Object Storage，COS）会自动生成文件链接（文件的 URL），您可以直接通过文件 URL（即 COS 默认域名）访问该文件。若您希望通过 CDN 或自己的域名访问 COS 上的文件，需要将 CDN 域名或自己的域名绑定至文件所在的存储桶。

您可以根据实际需求，使用对应的域名访问文件。例如您想通过 CDN 加速访问文件，则需要访问由 CDN 加速域名生成的文件链接。

## SecretKey

SecretId 和 SecretKey 合称为云 API 密钥，是用户访问腾讯云 API 进行身份验证时需要用到的安全凭证，可在 API 密钥管理 中获取。SecretKey 是用于加密签名字符串和服务器端验证签名字符串的密钥。一个 APPID 可以创建多个云 API 密钥。

## SecretId

SecretId 和 SecretKey 合称为云 API 密钥，是用户访问腾讯云 API 进行身份验证时需要用到的安全凭证，可在 API 密钥管理 中获取。SecretId 用于标识 API 调用者身份。一个 APPID 可以创建多个云 API 密钥。

### 访问控制列表（ACL）

访问控制列表（ACL）是基于资源的访问管理选项之一，用于描述一个访问权限行为。 在对象存储中，可用于管理存储桶和对象的访问。使用 ACL 可向其他主账号、子账号和用户组，授予基本的读、写权限。

## 3. SDK

我们采用腾讯云COS的Node

### 安装SDK

通过`yarn`安装环境SDK：

```
yarn add cos-nodejs-sdk-v5
```

### 开始使用

参考对象存储 快速入门-SDK 文档-文档中心-腾讯云 (tencent.com)。

COS文档建议使用临时密钥上传和下载文件。后端服务器掌握固定密钥，可以给需要操作文件的用户发放一个临时密钥，让用户使用COS服务。

服务端先写一个getSTS函数（sts.ts），用于向腾讯云请求一个临时密钥（服务端需掌握固定密钥）：

```
import STS from "qcloud-cos-sts";

// 获取临时密钥
const getSTS: any = async (action: string[], prefix: string) => {
    // 配置参数
    const config = {
        secretId: 'xxxxxx',    // 固定密钥
        secretKey: 'xxxxxx',   // 固定密钥
        proxy: '',
        host: 'sts.tencentcloudapi.com',
        durationSeconds: 1800,  // 密钥有效期
        bucket: 'ababa-12345678', // 换成你的 bucket
        region: 'ap-beijing', // 换成 bucket 所在地区
    };
    const scope = [{
        action: action,
        bucket: config.bucket,
        region: config.region,
        prefix: prefix,
    }];
    const policy = STS.getPolicy(scope);
    return new Promise((resolve, reject) => STS.getCredential({
        secretId: config.secretId,
        secretKey: config.secretKey,
        proxy: config.proxy,
        policy: policy,
        durationSeconds: config.durationSeconds,
    }, (err, credential) => {
        if (err) reject(err);
        else resolve(credential);
    }))
};
```

```
export default getSTS;
```

把临时密钥发给客户端：

```
import getSTS from 'sts';
const sts = await getSTS(["name/cos:GetObject"], `THUAI6/*`);
return res.status(200).send(sts);
```

客户端拿到临时密钥后，创建一个cos对象以使用COS服务（只具有`GetObject`权限，并且可操作的对象必须以`THUAI6/`开头）：

```
import COS from "cos-nodejs-sdk-v5";
const cos = new COS({
  getAuthorization: async (
    options: object,
    callback: (params: COS.GetAuthorizationCallbackParams) => void
  ) => {
    try {
      if (!sts) throw Error("Credentials invalid!");
      callback({
        TmpSecretId: sts.credentials.tmpSecretId,
        TmpSecretKey: sts.credentials.tmpSecretKey,
        SecurityToken: sts.credentials.sessionToken,
        StartTime: sts.startTime,
        ExpiredTime: sts.expiredTime,
      });
    } catch (err) {
      return console.log(err);
    }
  },
});
```

cos对象的下载（异步）：

```
const config = {
  bucket: "ababa-12345678",
  region: "ap-beijing",
};
// 定义下载对象的异步函数
const downloadObject = async function downloadObject(
  key: string,
  outputPath: string
): Promise<boolean> {
  return new Promise((resolve, reject) => {
    cos.getObject(      //调用了getObject函数下载
      {
        Bucket: config.bucket,
        Region: config.region,
        Key: key,
        Output: fStream.createWriteStream(outputPath),
      },
      (err) => {
        if (err) {
          reject(err);
```

```
        } else {
          resolve(true);
        }
      }
    );
  });
};
// 下载对象到指定目录
await downloadObject('key', 'download/path');
```

这个示例简单地介绍了COS的使用方法。官方有更详细对象管理、存储桶管理及访问控制等示例（不过是javascript的，需要一些修改才可用于typescript，当然可以直接@ts-nocheck，不过不推荐。此外，也推荐使用异步操作。），大家可以上网查看。

> 由于能力所限，笔者对于COS的了解仅限于一些基本操作，因此只能介绍到这里了。还希望各位听众批评指正！

# CDN

## 1. 但是什么是CDN呢?

刚刚我们在讲对象存储的时候提到了它有一个优点是支持CDN加速。什么是CDN呢?

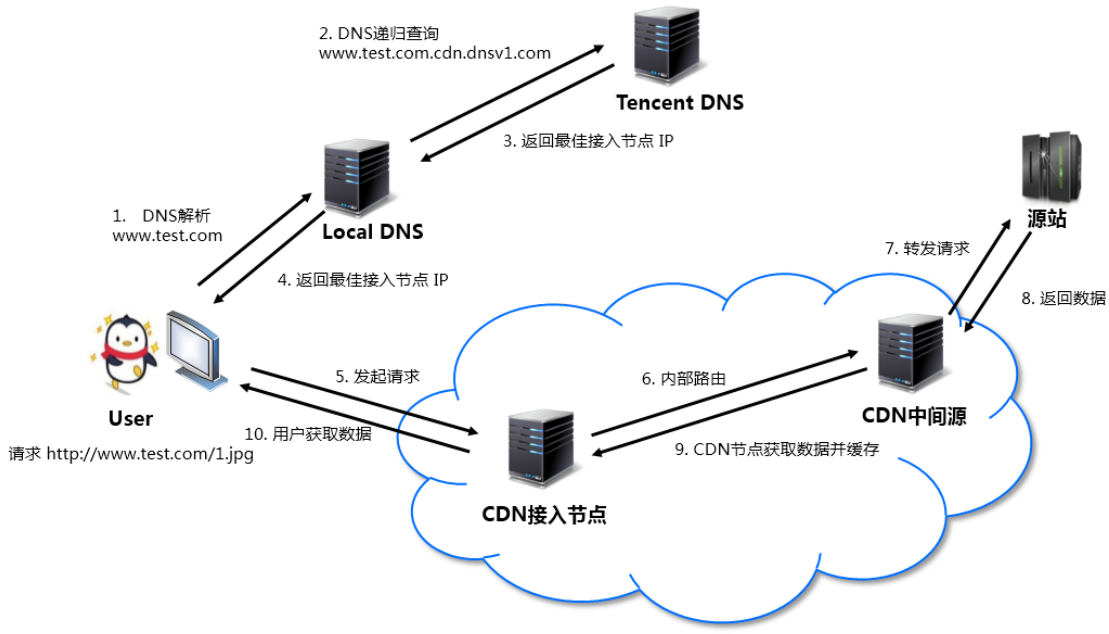
### CDN简介

CDN = Content Delivery Network，即内容分发网络，指的是一组分布在各个地区的服务器组成的网络。这些服务器存储着数据的副本，因此服务器可以根据哪些服务器与用户距离最近，来满足数据的请求。CDN有如下优势：

- 通过 CDN 向用户分发传输静态资源文件，可以降低我们自身服务器的请求压力。
- 大多数 CDN 在全球都有服务器，所以 CDN 上的服务器在地理位置上可能比你自己的服务器更接近你的用户。地理距离会按比例影响延迟。

### CND的工作流程

假设您的业务源站域名为 `www.test.com`，域名接入 CDN 开始使用加速服务后，当您的用户发起 HTTP 请求时，实际的处理流程如下图所示：




**详细说明如下：**

1. 用户向 `www.test.com` 下的某图片资源（如：1.jpg）发起请求，会先向 Local DNS 发起域名解析请求。
2. 当 Local DNS 解析 `www.test.com` 时，会发现已经配置了 CNAME `www.test.com.cdn.dnsv1.com`，解析请求会发送至 Tencent DNS（GSLB），GSLB 为腾讯云自主研发的调度体系，会为请求分配最佳节点 IP。
3. Local DNS 获取 Tencent DNS 返回的解析 IP。
4. 用户获取解析 IP。
5. 用户向获取的 IP 发起对资源 1.jpg 的访问请求。
6. 若该 IP 对应的节点缓存有 1.jpg，则会将数据直接返回给用户（10），此时请求结束。若该节点未缓存 1.jpg，则节点会向业务源站发起对 1.jpg 的请求（6、7、8），获取资源后，结合用户自定义配置的缓存策略，将资源缓存至节点（9），并返回给用户（10），此时请求结束。

> 个人理解：CDN的作用类似于CPU中的高速缓存，会在离用户比较近的一个服务器里保存用户曾经请求过的资源，下次再有用户请求时就直接把保存的数据给用户。
>
> 用户意识不到他们正在访问cdn服务器，而是以为自己在访问源站本体，因为他们访问的网址是源站的网址而不是cdn的网址。之所以请求的是源站得到的却是cdn服务器的响应，是因为DNS服务器将源站的域名**重定向**到了cdn服务器的ip地址。重定向的方法便是设置CNAME：
>
> “**CNAME**记录将一个域名（别名）指向另一个域名（目标）。当用户访问别名域名时，DNS服务器会将请求重定向到目标域名，并返回目标域名的IP地址。这样，用户的请求最终会被发送到目标域名所在的服务器上。（ChatGPT老师如是说）”
>
> 另外补充一下DNS服务器的知识（待会在Web服务器部分也会讲）：
>
> “**DNS服务器**（Domain Name System Server，域名系统服务器）是一种网络服务器，它负责将域名转换为对应的IP地址。本地DNS服务器收到查询请求后，会首先检查自己的缓存中是否有该域名对应的IP地址。如果有，它会立即返回IP地址给用户的浏览器。如果没有，本地DNS服务器会向根DNS服务器发送查询请求。（Chat老师语）”
>
> 总之，DNS服务器实现域名到ip的转换。DNS服务器是递归式的查询，一个服务器处理不了就会交给另一个服务器，在CDN中会最终交给DNS调度服务器解决。
>
> 综上所述，CDN的工作流程大概是这样：
>
> 用户向DNS请求把源站的网址解析为ip地址-->本地DNS服务器发现源站网址在CNAME记录里，应该把请求重定向给CNAME记录指向的网址，并把请求抛给另一个DNS服务器-->最终网址由DNS调度服务器（文中的GSLB）解析，DNS调度服务器挑选一个最合适的ip地址给用户，这个ip通常是离用户最近的CDN服务器-->用户和CDN服务器建立连接，完成偷梁换柱。

## 2. CDN配置流程

以阿里云的CDN服务为例：

**开通CDN服务**

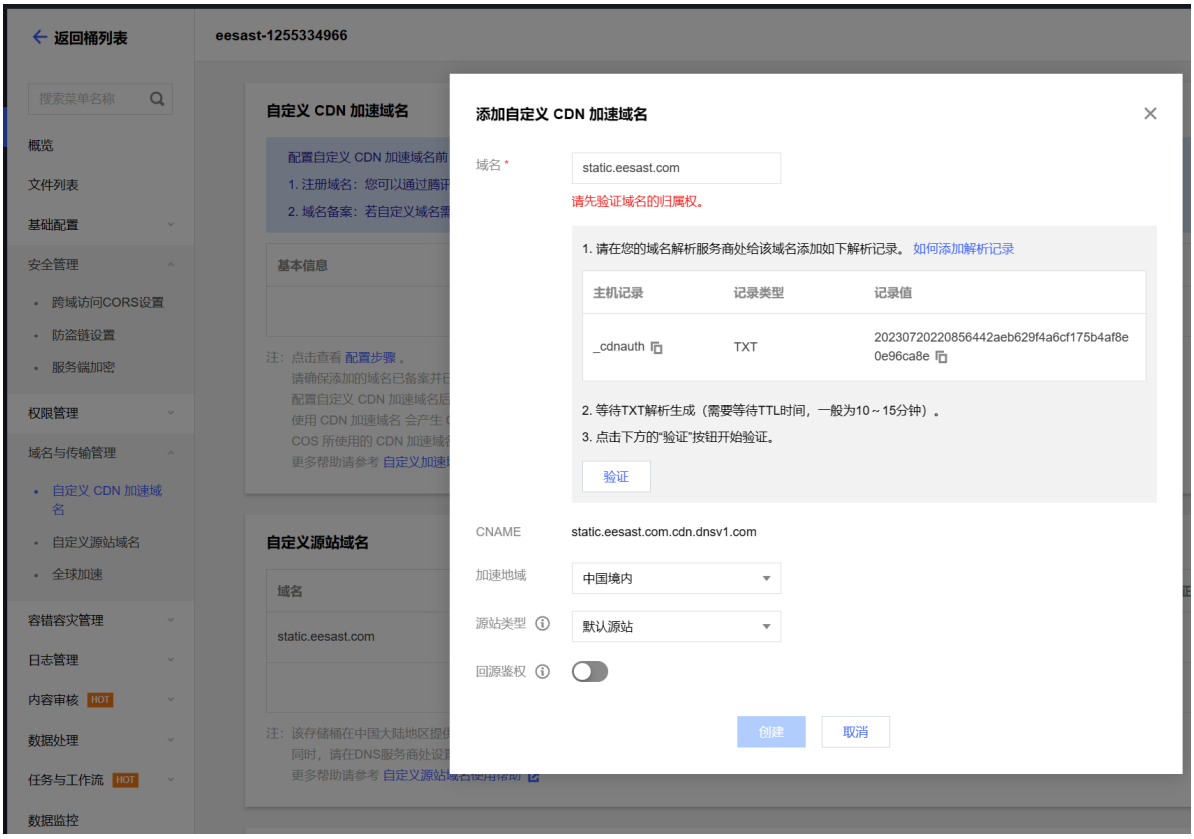
1. 登录阿里云CDN平台。
2. 单击**立即开通**。
3. **计费类型**默认按使用流量计费，并选中服务协议。
4. 单击**立即开通**。

**添加加速域名**

1. 登录CDN控制台。
2. 在左侧导航栏，单击**域名管理**。
3. 单击**添加域名**，完成基础信息和业务信息配置。
4. 完成基础信息和业务信息配置后，单击**新增源站信息**。
5. 在**新增源站信息**页面，完成配置。

6. 完成源站配置后，单击**下一步**。
7. 等待人工审核。

对于COS存储桶，我们可以在下面的界面启用并设置CDN加速域名（注意验证域名归属权）。




### 验证域名归属权

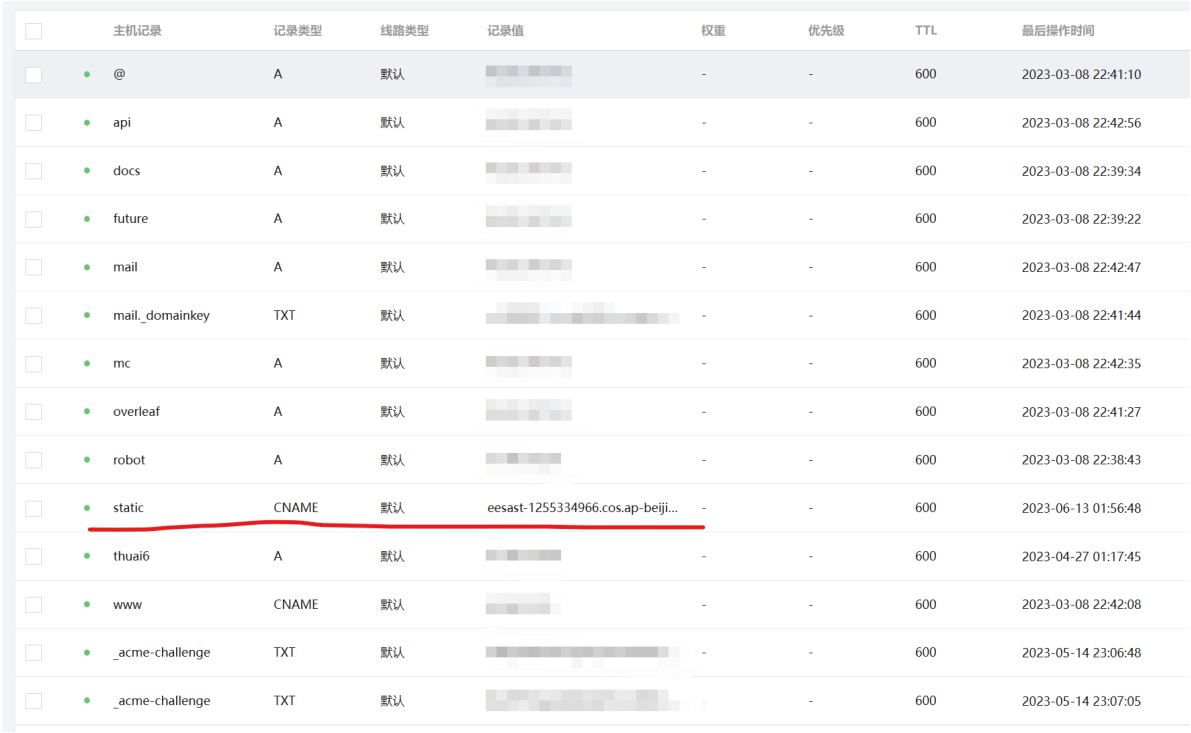
- 方法一：DNS解析验证（推荐）：需要添加TXT记录
- 方法二：文件验证
- 方式三：通过API验证

### 配置CNAME

成功添加加速域名（完成源站配置）后，CDN会自动分配一个CNAME域名（也就是源站域名将作为CNAME域名的别名）。只有在域名解析服务商处将源站域名的DNS解析记录指向CNAME域名，访问请求才能转发到CDN节点上，实现CDN加速。

配置CNAME的方法随源站域名的服务商不同而不同，在此不再赘述。需要用到的同学可以参考：配置CNAME。

如图，电子系科协网站将`static.eesast.com`的CNAME指向了腾讯云COS的存储桶，将访问资源的请求转发给腾讯云COS。这是未启用CDN的状态。如果我们要启用CDN，就需要把CNAME的记录值改成之前配置源站域名时的`static.eesast.com.cdn.dnsv1.com`。



| 主机记录 | 记录类型 | 线路类型 | 记录值 | 权重 | 优先级 | TTL | 最后操作时间 |
|---|---|---|---|---|---|---|---|
| @ | A | 默认 | [redacted] | - | - | 600 | 2023-03-08 22:41:10 |
| api | A | 默认 | [redacted] | - | - | 600 | 2023-03-08 22:42:56 |
| docs | A | 默认 | [redacted] | - | - | 600 | 2023-03-08 22:39:34 |
| future | A | 默认 | [redacted] | - | - | 600 | 2023-03-08 22:39:22 |
| mail | A | 默认 | [redacted] | - | - | 600 | 2023-03-08 22:42:47 |
| mail._domainkey | TXT | 默认 | [redacted] | - | - | 600 | 2023-03-08 22:41:44 |
| mc | A | 默认 | [redacted] | - | - | 600 | 2023-03-08 22:42:35 |
| overleaf | A | 默认 | [redacted] | - | - | 600 | 2023-03-08 22:41:27 |
| robot | A | 默认 | [redacted] | - | - | 600 | 2023-03-08 22:38:43 |
| static | CNAME | 默认 | eesast-1255334966.cos.ap-beiji... | - | - | 600 | 2023-06-13 01:56:48 |
| thuai6 | A | 默认 | [redacted] | - | - | 600 | 2023-04-27 01:17:45 |
| www | CNAME | 默认 | [redacted] | - | - | 600 | 2023-03-08 22:42:08 |
| _acme-challenge | TXT | 默认 | [redacted] | - | - | 600 | 2023-05-14 23:06:48 |
| _acme-challenge | TXT | 默认 | [redacted] | - | - | 600 | 2023-05-14 23:07:05 |

## 3. CDN常用功能

**资源刷新**

强制删除CDN所有节点上的缓存资源，当用户向CDN节点请求资源时，CDN会直接回源站获取对应的资源并返回，同时将资源重新缓存到CDN节点。刷新功能会降低缓存命中率，但是能保证用户获取到最新的内容。

一般用于资源更新和发布，以及违规资源清理等。

**资源预热**

源站主动将对应的资源缓存到CDN节点，当客户首次请求资源时，即可直接从CDN节点获取到最新的资源，无需再回源站获取。预热功能会提高缓存命中率，但是会造成源站短时高负载。

一般用于运营大型活动，以及安装包发布等。

> 能坚持看到这里真是辛苦了，不容易啊不容易。我已经写麻了，一边写材料一边发现自己还有很多不懂的地方。

# Web服务器

## 1. 请求-响应流程 & 代理简介

### 域名解析

当我们在浏览器上输入一个域名发送网络请求时，发生了什么呢?

首先，浏览器会去搜索浏览器自身的DNS（域名服务器）缓存，如果存储有域名对应的IP地址，则缓存命中；否则去搜索操作系统自身的DNS缓存和host配置文件，如果没找到则去请求本地DNS服务器（一般在同一城市）进行解析，如果本地DNS服务器上还没有，则要由本地DNS服务器去请求13台IPv4 或25台IPv6根服务器，最终将域名映射到IP地址。只要是用域名发送的网络请求，都要经历域名解析这 一过程。

### 请求和响应

有了服务器的IP地址以后，客户端就可以同服务器经历握手建立TCP/IP连接。在连接建立后，客户端就可以向服务器发起HTTP请求，请求格式在之前已经讲解。服务器处理完毕后会将HTTP响应发回客户端，响应格式在之前也已经讲解。如果是浏览器向服务器发送请求，得到的响应是HTML文件，浏览器解析HTML文件来渲染页面，当遇到js/css或者图片等资源时，则再向服务器发送HTTP请求确认本地缓存的文件是否有修改，如果有修改则再次下载；当要进行API调用时，则通过本地的前端代码向服务器后端发送请求，由后端作出响应。

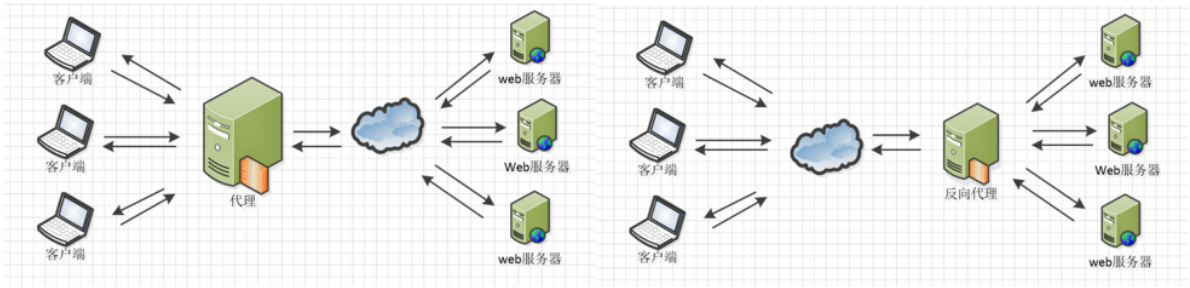
如果使用代理，那客户端就不再是直接同目标服务器建立连接，而是同代理服务器或反向代理服务器建立连接，间接地访问目标服务器。下面来详细介绍代理：

### 代理与反向代理

代理（正向代理）是指，客户端先将请求发送至代理服务器这一中转站，再由代理服务器发送至目标服务器。在此过程中，客户端知道目标服务器的地址，但目标服务器不知道客户端的地址，只知道请求是从代理服务器发来的。例如虚拟专用网络（VPN）使用了正向代理。

反向代理则相反，反向代理服务器接受外部网络用户的请求并代表外部网络用户向内部服务器发出请求，对外表现为一个服务器。在此过程中，目标服务器知道客户端的地址，但客户端并不知道真正处理请求的内部服务器地址，其请求都发送给了反向代理服务器。Nginx就是一款经典反代服务器，接下来会介绍它的应用。

下面这张图比较形象：




## 2. Nginx

Nginx是一款著名的Web和反向代理服务器，此外还有Apache等。我们将使用Nginx完成前端和后端的通信，从而搭建一个具有完整功能的网站。

#### Nginx的安装与使用

在Ubuntu上，我们通过apt工具安装Nginx：

```
sudo apt update
sudo apt install nginx
```

下面介绍Nginx的常用命令：

```
nginx -v # 查看nginx版本
nginx -c filename # 设置配置文件(默认是:/etc/nginx/nginx.conf)
nginx # 启动nginx服务
nginx -s stop # 强制停止nginx服务
nginx -s quit # 正常停止nginx服务（即处理完所有请求后再停止服务）
nginx -s reload # 重新加载nginx，适用于当nginx.conf配置文件修改后，使用下面命令可以使
得配置文件生效
```

> 有了Docker之后，可以直接在Dockerhub上搜索nginx镜像，写好配置并设置好ip和端口就可以跑了，更加方便。

**Nginx配置**

最终生效的配置文件默认地址：`/etc/nginx/nginx.conf`，这一地址可以通过`nginx -c`命令修改。但是我们一般不去直接改这个文件，大家可以进这个文件看一下，这个文件实际上引用了`/etc/nginx/sites-enabled`里的配置文件，因此凡是`/etc/nginx/sites-enabled`里的配置文件都会被加载生效（enable）。而进到`/etc/nginx/sites-enabled`目录会发现，里面的`default`是一个软链接，指向`/etc/nginx/sites-available`。因此实际的配置流程是这样，在`sites-available`目录下写好可能要用到的配置文件，将希望生效的配置文件软链接到`sites-enabled`目录下，然后用`sudo service nginx restart`命令即可重启 nginx服务。

配置文件的结构如下：

第一部分：全局块

```
# 每行配置必须有分号结束
worker_processes [number/auto]; # 允许启动的进程数，影响服务器并发处理能力
user [user] [user group]; # 配置nginx工作进程（处理请求的进程）所属的用户或者
组，默认为nobody nobody
pid [...../nginx.pid]; # 指定nginx进程运行文件存放地址，一般不用改
error_log [log addr] debug; # 指定日志路径和级别（默认为error）
```

第二部分：events块

```
events {
    accept_mutex on; # 设置网路连接序列化，防止惊群现象发生，默认为on
    multi_accept on; # 设置一个进程是否同时接受多个网络连接，默认为off
    use epoll; # 事件驱动模型，select|poll|kqueue|epoll|resig|/dev/poll|eventport
    worker_connections 1024; # 最大连接数，默认为512
}
```

第三部分：http块

```
http {
    # http全局块
    include mime.types; # 文件扩展名与文件类型映射表
    default_type application/octet-stream; # 默认文件类型，默认为text/plain
    access_log off; # 取消服务日志
    log_format myFormat '$remote_addr-$remote_user [$time_local] $request
$status $body_bytes_sent $http_referer $http_user_agent $http_x_forwarded_for';
# 自定义日志格式
    access_log log/access.log myFormat; # combined为日志格式的默认值
    sendfile on; # 允许sendfile方式传输文件，默认为off
    sendfile_max_chunk 100k; # 每个进程每次调用传输数量不能大于设定的值，默认为0，即不设上
限
    keepalive_timeout 65; # 连接超时时间，默认为75s
    upstream mysvr {
        server 127.0.0.1:7878;
        server 192.168.10.121:3333 backup; # 热备
    }
    error_page 404 https://www.baidu.com; # 错误页
    server {
        keepalive_requests 120; # 单连接请求上限次数。
        listen 4545; # 监听端口
```

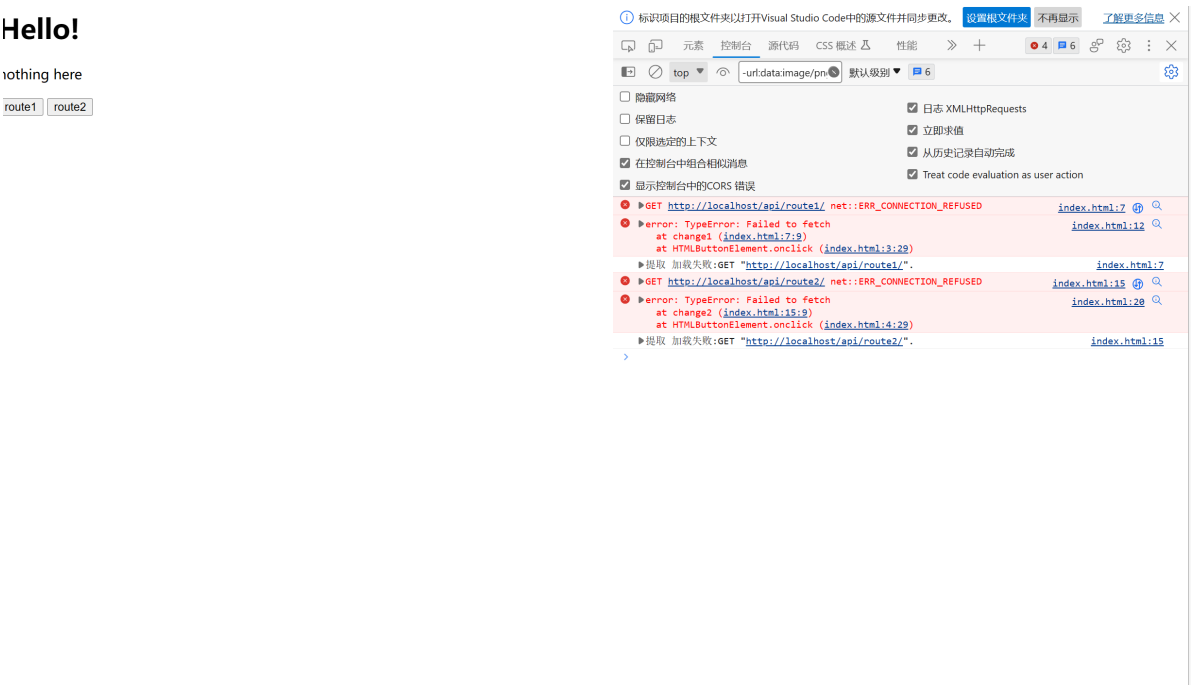
```
        server_name 127.0.0.1; # 主机名称
        location / { # 请求的url过滤，正则匹配，~为区分大小写，~*为不区分大小写。
            root path; # 定义根目录
            index index.html; # 设置默认页
            proxy_pass http://mysvr; # 请求转向mysvr定义的服务器列表
            deny 127.0.0.1; # 拒绝的ip
            allow 172.18.5.54; # 允许的ip
        }
    }
}
```

**用Nginx部署前后端**

由于本节课的重点是后端，因此我们的前端采用最简单的一个html文件，功能是在点击“route1”按钮或“route2”按钮时向后端发送一个请求，并将返回的文本显示出来，代码如下：

```
<h1> Hello! </h1>
<p id="text"> nothing here </p>
<button onclick="change1()"> route1 </button>
<button onclick="change2()"> route2 </button>
<script>
  function change1() {
    fetch('http://localhost/api/route1/', { method: 'GET' }).then(data => {
      return data.text();
    }).then((text) => {
      console.log(text);
      document.getElementById("text").innerHTML = text;
    }).catch(error => console.error('error:', error));
  }
  function change2() {
    fetch('http://localhost/api/route2/', { method: 'GET' }).then(data => {
      return data.text();
    }).then((text) => {
      console.log(text);
      document.getElementById("text").innerHTML = text;
    }).catch(error => console.error('error:', error));
  }
</script>
```

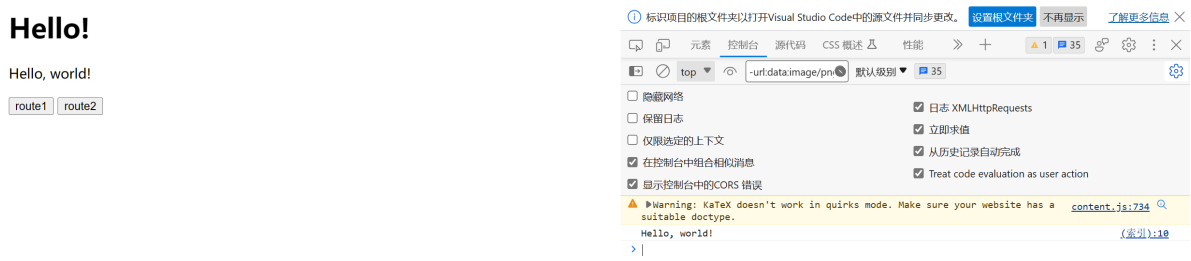
我们将上述代码写在`index.html`当中，作为前端的入口文件（事实上是唯一文件），然后保存在一个目录下，比如`D:\server`（路径中最好不要有空格、中文之类的，容易出错）。可以双击这一文件用浏览器预览一下，现在如果点击按钮则什么都不会发生。可以摁F12打开浏览器的开发人员工具（以Edge为例），在上方栏点更多找到网络标签，进去后再次点击按钮会发现请求失败。



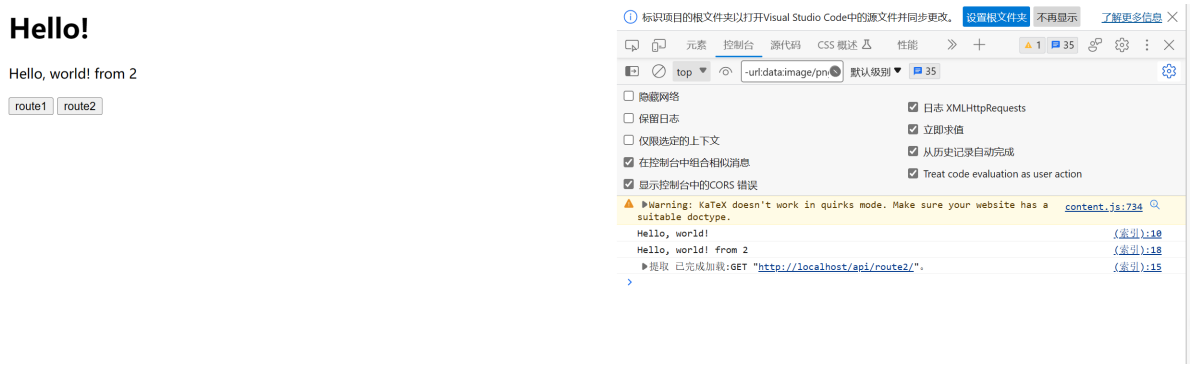

用超级用户权限打开`/etc/nginx/sites-available`目录下面的`default`文件，修改这一文件就能起到配置nginx的目的。我们把这一文件修改成下面这样：

```
# 省略一些注释
server {
    listen 80 default_server; # 监听端口，http默认80
    listen [::]:80 default_server;
    # 省略一些注释
    root /mnt/d/server; # 将服务器根目录设为自己index.html所在的目录
    # Add index.php to the list if you are using PHP
    index index.html index.htm index.nginx-debian.html;
    server_name _;
    location / { # 其他路径匹配至此
        # First attempt to serve request as file, then
        # as directory, then fall back to displaying a 404.
        try_files $uri $uri/ /index.html =404;
    }
    location /api/ { # /api路径设为后端专用路径，优先匹配
        proxy_pass http://127.0.0.1:3000/; # 转发至后端监听的端口（代理）
    }
    # 省略一些注释
}
# 省略一些注释
```

保存并退出后，用`sudo service nginx restart`命令重启nginx服务，用`yarn start`命令启动 后端（感兴趣的同学可以用babel将后端build成JavaScript代码再用node运行入口文件，但简便起见用开发启动也能达到相同效果），然后在浏览器中进入localhost（127.0.0.1），就会看到我们在`index.html`中写的简单前端，同时如果点击按钮也能看到显示文字的改变。




点击route1




点击route2

### 在云服务器上部署网站

理论上来讲，每一台安装有基本网络协议的计算机都可以做客户端与服务器，也可以像我们刚才的例子那样既做客户端又做服务器。只要做好防火墙入站规则等一系列设置，再向运营商申请一个公网IP，就能把你的电脑变成大家都能访问的服务器。但是由于性能和管理的诸多原因，一般用户大都选择购买云服务器，由云服务商负责相关运维，节省用户的精力。

> 个人觉得公网ip的申请大概是个难题（x）云服务器因此更具优势一点

下面简要介绍在云服务器上部署简易网站的方法： 首先，在任一家服务商购买相关产品，根据提示完成配置后进入系统（假设为Ubuntu），确保有超级用户权限。同时，在控制台开放80端口（用于HTTP）和22端口（用于SSH），根据需要开放443端口（用于HTTPS）。

然后，在系统内安装Nginx和Node.js，按照之前的方法配置Nginx（如根目录直接路由至前端的`index.html`以呈现主页，`/api`目录代理至本机的3000端口供后端监听处理）。注意前端发送请求不能再到`localhost`，而要到`http://[主机ip]/api`下面。

```
location / { # 其他路径匹配至此
    # First attempt to serve request as file, then
    # as directory, then fall back to displaying a 404.
    try_files $uri $uri/ /index.html =404;
}
location /api/ {
    proxy_pass http://127.0.0.1:3000/
}
```

最后，用Babel或其他工具将TypeScript后端转换成JavaScript，生成到`build`目录下，上传到服务器，在服务器后台运行`node ./build/app.js`以启动后端。

这样，我们的简易网站就完成了。在浏览器中直接输入ip地址（暑培测试用：140.143.163.190），就能看到网站了。

> 用户：test，密码：eesast111

# 作业

大家愿意写我的作业我感到非常荣幸（x

## 作业内容

在以下两个项目中二选一：

1. 为“成绩管理系统”添加一个后端。前端可以采用前几天学习过的html/js，react，或者其他任何形式的前端，或者没有前端（至少要介绍一下怎么发请求）。后端采用Express构建，要负责响应前端的请求（录入信息，查询成绩），并使用MongoDB数据库管理学生成绩数据。
2. 任何一个你想做的并且使用到本次讲座中至少一项技术的项目。不要求代码的复杂程度，但是至少是一个完整的项目。例如经典的todo List（你甚至可以给它加登录，注册，云端同步等功能）。

## 一些帮助（不断更新）

- 如果你遇到了跨域问题（`No 'Access-Control-Allow-Origin' header`），可以试试使用cors（一个npm包）
- 欢迎随时发邮件or在群里提问！

## 请告诉我们你的姓名和班级，我们会给你加创新学分(x)

你可以发送邮件到tsz21@mails.tsinghua.edu.cn，主题为2023暑培_姓名_班级，附件包含一个且仅有一个压缩文件（zip/7z），包含项目的全部源码（不过请不要放**node_modules**进来）以及说明文件（如何启动项目，以及介绍两句这个项目有什么功能，随便写写即可）。

祝大家暑假愉快！