Express MongoDB OSS/CDN WEB服务器

刘度 无02

Express

1.Express简介与环境配置

为什么要用Express?

Node.js使JavaScript\TypeScript脚本能够脱离浏览器环境在服务端（后端）运行（实际上是对Chrome V8引擎进行了封装），为我们开发后端提供了一种选项。不像前端有统一的浏览器标准，如果不遵循的话浏览器就没法正常显示；后端的开发相对就自由许多，开发语言有很多选项，如Java，PHP，python，C，Go等，我们科协网站采用的是TypeScript语言，有关TypeScript的知识在之前课程当中已有介绍。

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：上次没有讲apt换源，如果下载过慢或失败可以采用清华源，换源方法见https://mirrors.tuna.tsinghua.edu.cn/help/ubuntu/。换源后，使用如下命令检验是否换源成功并更新软件：

```
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

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。

```
app.listen(3000, () => {        // 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。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
```

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（删去注释）：

```
{
    "presets": [                    // 指定要使用的工具集。更多配置详见官方文档：
https://www.babeljs.cn/docs/
        "@babel/preset-env",
        "@babel/preset-typescript"
    ]
}
```

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（删去注释）：

```
"nodemonConfig": {      // 更多配置详见官方文档：
https://www.npmjs.com/package/nodemon
    "watch": [          // 监听的文件目录
      "src"
    ],
    "ext": "ts, json",  // 监听的文件后缀
    "exec": "babel-node --extensions \".ts\" src/app.ts"    // 当运行nodemon命令时
执行
    // 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文件中新建一个如下字段（删去注释）：

```
"scripts": {
    "start": "nodemon"  // 等价于./node_modules/nodemon/bin/nodemon.js
  }
```

这样，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。此时输入rs回车，会使后端重启；摁下Ctrl+C，会使后端停止。

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。

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请求基本结构

要想弄清楚Express是如何路由的，有必要先了解HTTP请求的格式与结构。HTTP（超文本传输协议）基于TCP/IP协议来传递数据，是为客户端与服务器之间的通信而设计的。客户端请求的格式如下：

服务器响应的格式如下：

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：

序号	方法	描述
1	GET	向指定资源发出请求，用于获取资源，数据被包含在URL中
2	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
3	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
4	PUT	向指定资源处上传其最新内容
5	DELETE	请求服务器删除Request-URL所标识的资源
6	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
7	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
8	TRACE	回显服务器收到的请求，主要用于测试或诊断
9	PATCH	是对PUT方法的补充，用来对已知资源进行局部更新

一般我们常用的是GET、POST和PUT请求。接着我们来了解一下HTTP响应状态码，下表列举了一些可能遇到的状态码及描述：

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，但是拒绝执行此请求
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	作为网关或者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，默认为80（对于HTTPS协议默认为443），是一个HTTP选择器，是查询字符串。例如：

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

协议是`https`，主机地址是`baike.baidu.com`（会被DNS解析成IP地址），采用默认的`443`端口；路径为`/item/%E7%BB%9F.../5937042`，其中带有路径参数`/%E7%BB%9F.../5937042`；查询字符串为`fromtitle=url&fromid=110640`。

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。

普通字符串如：

```
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');
})
```

字符串模板如：

```
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');
})
```

正则表达式如：

```
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
```

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。

仍举上述URL的例子：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。则可通过以下路由解析：

```
app.get('/item/:param1/:param2', (req, res) => {
  console.log(req.params.param1);       // 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。

```
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。

MongoDB

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。我们用Windows版本的MongoDB来做演示。

何为MongoDB?

大家之前已经学习过了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。因此，MongoDB是非关系型数据库当中最像关系型数据库的。

MongoDB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。

MongoDB既有命令行的管理工具，也有图形界面的管理工具。MongoDB Compass 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。**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。

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。在MongoDB中，必须使用UTF-8编码的字符串。
Integer	整数。根据你所采用的服务器，可分为32位或64位。
Boolean	布尔值。
Double	双精度浮点值。
Array	数组。将多个数据类型按序存储为一个字段。
Timestamp	时间戳。记录文档修改或添加的具体时间。
Object	对象。将多个数据类型按键值对存储为一个字段。
Null	空值。
Date	日期时间。用UNIX时间格式来存储当前日期或时间。
Binary Data	二进制数。

MongoDB中的文档必须有一个_id键作主键。这个键的值可以是任何类型的，默认是个ObjectId类型的对象，这一对象的结构如下：

2.MongoDB基础操作

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，语法格式如下：

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

- query （可选）：使用查询操作符（对象）指定筛选条件。
- 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，语法格式如下：

```
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.test.find({"key":100})`	`where key = 100`

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

参数：

- query：同“查看文档”中介绍的一样的查询条件，用于筛选要更新的文档。
- update：要更新的对象和更新操作符。
- upsert：可选，如果不存在update的记录，是否插入新的记录。默认false，不插入。
- multi：可选，是否按条件查询出的多条记录全部更新。 默认false，只更新找到的第一条记录。
- writeConcern：可选，决定一个写操作落到多少个节点上才算成功。

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：

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_name :new_field_name } }	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。

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

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。

```
db.[collection name].find([query], [projection]).sort({[key]:[1 or -1]})
```

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	实例
$sum	计算总和	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : "$likes"}}}])

表达式	描述	实例
$avg	计算平均值	db.mycol.aggregate([{$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"}}}])

4.Mongoose

安装和导入

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，我们用yarn包管理器来安装MongoDB在Node.js下的API。

```
yarn add mongoose
```

接下来导入mongoose包并连接MongoDB数据库：

```
import mongoose from "mongoose";

mongoose.connect(`mongodb://localhost/test`, {      // 假设连接本地的test数据库，也可以连接远程数据库
  user: [用户名],              // 本地数据库可能不需要用户名和密码
  pass: [密码]
});

const db = mongoose.connection;                     // 数据库实例
db.on("error", (error) => {                         // 连接失败
  console.log("Database connection error: " + error);
});
db.once("open", () => {                             // 连接成功
  console.log("Database connected");
});
```

Schema

mongoose的基本概念同MongoDB类似，但使用方法同MongoDB的CLI有所不同。在mongoose当中，Schema是一个重要概念，Schema定义了数据库集合当中文档的构成。我们来定义第一个Schema：

```
const blogSchema = new mongoose.Schema({
    title:  String,
    author: String,
    body:   String,
    comments: [{ body: String, date: Date }],
    date: { type: Date, default: Date.now },
    hidden: Boolean,
    meta: {
        votes: Number,
        favs:  Number
    }
});
```

Model

Model是schema生成的构造函数，model的实例就代表着可以从数据库保存和读取的document。从数据库创建和读取Document的所有操作都是通过model进行的。也可以认为，model对应着MongoDB中的一个集合（Collection），用model可以生成其中的文档。

```
var schema = new mongoose.Schema({ name: 'string', size: 'string' });
var Tank = mongoose.model('Tank', schema);
```

第一个参数是model对应的集合（collection）名字的单数形式。Mongoose 会自动找到名称是 model 名字复数形式的 collection 。 对于上例，Tank 这个 model 就对应数据库中 tanks 这个 collection。

有了model，我们就可以很方便的构建一个集合当中的文档。

```
Tank.create({ size: 'small' }, function (err, small) {
  if (err) return handleError(err);
  // saved!
})
```

时间有限，对于更多有关mongoose的内容不再介绍，大家可以参考官方文档http://www.mongoosejs.net/docs/index.html。

OSS

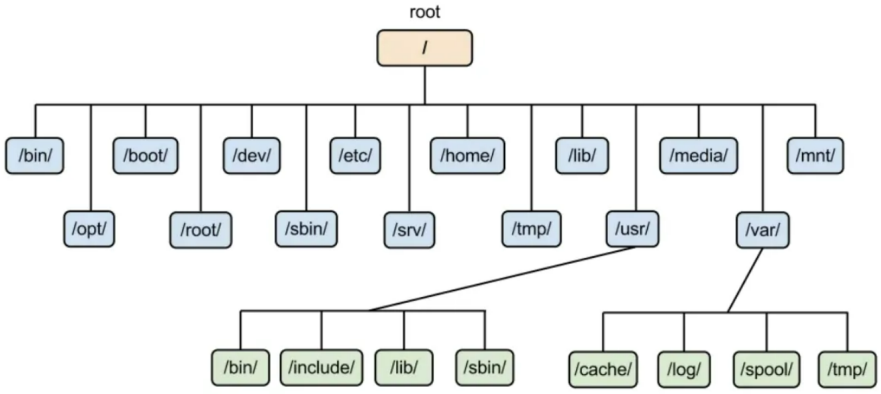
1.OSS简介

何为OSS?

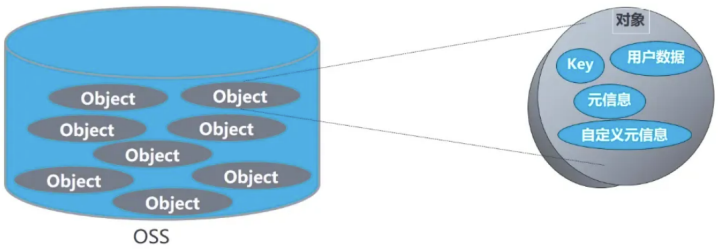
OSS = Object Storage Service，即对象存储服务。如果非要下正式定义，那么对象存储是一种使用**RESTful API** 存储和检索**非结构化**数据和元数据对象的工具。如果用正常语言来解释，可以把OSS类比为云盘，只不过我们可以通过网络请求进行下载、上传等操作。

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）：



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
- 以阿里云的产品为例，OSS比云盘便宜

OSS的优势

- 提供标准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，阿里OSS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。

阿里OSS提供了完善的技术文档，同学们可以很方便的搜索到官网上的文档，后面的介绍也大多来源于官方文档

2.OSS基本概念

存储空间（Bucket）

存储空间是用户用于存储对象（Object）的容器，所有的对象都必须隶属于某个存储空间。存储空间具有各种配置属性，包括地域、访问权限、存储类型等。用户可以根据实际需求，创建不同类型的存储空间来存储不同的数据。

- 同一个存储空间的内部是扁平的，没有文件系统的目录等概念，所有的对象都直接隶属于其对应的存储空间。
- 每个用户可以拥有多个存储空间。
- 存储空间的名称在OSS范围内必须是全局唯一的，一旦创建之后无法修改名称。
- 存储空间内部的对象数目没有限制。

存储空间的命名规范如下：

- 只能包括小写字母、数字和短划线（-）。
- 必须以小写字母或者数字开头和结尾。
- 长度必须在3~63字符之间。

对象（Object）

对象是OSS存储数据的基本单元，也被称为OSS的文件。和传统的文件系统不同，对象没有文件目录层级结构的关系。对象由元信息（Object Meta），用户数据（Data）和文件名（Key）组成，并且由存储空间内部唯一的Key来标识。对象元信息是一组键值对，表示了对象的一些属性，比如最后修改时间、大小等信息，同时用户也可以在元信息中存储一些自定义的信息。

对象的命名规范如下：

- 使用UTF-8编码。
- 长度必须在1~1023字符之间。
- 不能以正斜线（/）或者反斜线（\）开头。

ObjectKey

在各语言SDK中，ObjectKey、Key以及ObjectName是同一概念，均表示对Object执行相关操作时需要填写的Object名称。例如向某一存储空间上传Object时，ObjectKey表示上传的Object所在存储空间的完整名称，即包含文件后缀在内的完整路径，如填写为abc/efg/123.jpg。

Region（地域）

Region表示OSS的数据中心所在物理位置。用户可以根据费用、请求来源等选择合适的地域创建Bucket。一般来说，距离用户更近的Region访问速度更快。

Region是在创建Bucket的时候指定的，一旦指定之后就不允许更改。该Bucket下所有的Object都存储在对应的数据中心，目前不支持Object级别的Region设置。

Endpoint（访问域名）

Endpoint表示OSS对外服务的访问域名。OSS以HTTP RESTful API的形式对外提供服务，当访问不同的Region的时候，需要不同的域名。通过内网和外网访问同一个Region所需要的Endpoint也是不同的。例如杭州Region的外网Endpoint是oss-cn-hangzhou.aliyuncs.com，内网Endpoint是oss-cn-hangzhou-internal.aliyuncs.com。具体的内容请参见各个Region对应的Endpoint。

AccessKey（访问密钥）

AccessKey简称AK，指的是访问身份验证中用到的AccessKeyId和AccessKeySecret。OSS通过使用AccessKeyId和AccessKeySecret对称加密的方法来验证某个请求的发送者身份。AccessKeyId用于标识用户；AccessKeySecret是用户用于加密签名字符串和OSS用来验证签名字符串的密钥，必须保密。对于OSS来说，AccessKey的来源有：

- Bucket的拥有者申请的AccessKey。
- 被Bucket的拥有者通过RAM授权给第三方请求者的AccessKey。
- 被Bucket的拥有者通过STS授权给第三方请求者的AccessKey。

强一致性

Object操作在OSS上具有原子性，操作要么成功要么失败，不会存在有中间状态的Object。OSS保证用户一旦上传完成之后读到的Object是完整的，OSS不会返回给用户一个部分上传成功的Object。

Object操作在OSS同样具有强一致性，用户一旦收到了一个上传（PUT）成功的响应，该上传的Object就已经立即可读，并且Object的冗余数据已经写成功。不存在一种上传的中间状态，即read-after-write却无法读取到数据。对于删除操作也是一样的，用户删除指定的Object成功之后，该Object立即变为不存在。

附：OSS与文件系统的对比

对比项	OSS	文件系统
数据模型	OSS是一个分布式的对象存储服务，提供的是一个Key-Value对形式的对象存储服务。	文件系统是一种典型的树状索引结构。
数据获取	根据Object的名称（Key）唯一的获取该Object的内容。虽然用户可以使用类似test1/test.jpg的名字，但是这并不表示用户的Object是保存在test1目录下面的。对于OSS来说，test1/test.jpg仅仅只是一个字符串，与example.jpg并没有本质的区别。因此不同名称的Object之间的访问消耗的资源是类似的。	一个名为test1/test.jpg的文件，访问过程需要先访问到test1这个目录，然后再在该目录下查找名为test.jpg的文件。
优势	支持海量的用户并发访问。	支持文件的修改，比如修改指定偏移位置的内容、截断文件尾部等。也支持文件夹的操作，比如重命名目录、删除目录、移动目录等非常容易。

对比项	OSS	文件系统
劣势	OSS保存的Object不支持修改（追加写Object需要调用特定的接口，生成的Object也和正常上传的Object类型上有差别）。用户哪怕是仅仅需要修改一个字节也需要重新上传整个Object。OSS可以通过一些操作来模拟类似文件夹的功能，但是代价非常昂贵。比如重命名目录，希望将test1目录重命名成test2，那么OSS的实际操作是将所有以test1/开头的Object都重新复制成以test2/开头的Object，这是一个非常消耗资源的操作。因此在使用OSS的时候要尽量避免类似的操作。	受限于单个设备的性能。访问越深的目录消耗的资源也越大，操作拥有很多文件的目录也会非常慢。

3.访问权限

文件的访问权限（ACL）有以下四种：

访问权限	描述	访问权限值
继承Bucket	文件遵循存储空间的访问权限。	default
私有	文件的拥有者和授权用户有该文件的读写权限，其他用户没有权限操作该文件。	private
公共读	文件的拥有者和授权用户有该文件的读写权限，其他用户只有文件的读权限。	public-read
公共读写	所有用户都有该文件的读写权限。	public-read-write

4.OSS SDK

由于科协网站采用的是阿里云的OSS，在这里我们只介绍阿里云OSS的SDK。又由于我们的前端和后端都是以JavaScript为基础的，因此我们只关心基于Node.js的SDK。可以用yarn包管理器安装ali-oss包：

```
yarn add ali-oss
```

初始化

```
import OSS from "ali-oss";

let client = new OSS({
  // yourRegion填写Bucket所在地域。以华东1（杭州）为例，Region填写为oss-cn-hangzhou。
  region: 'yourRegion',
  accessKeyId: 'yourAccessKeyId',
  accessKeySecret: 'yourAccessKeySecret'
});
```

存储空间（Bucket）操作

创建存储空间：

```
// 创建存储空间。
async function putBucket() {
  try {
    const options = {
      storageClass: 'Standard', // 存储空间的默认存储类型为标准存储，即Standard。如果需要设置存储空间的存储类型为归档存储，请替换为Archive。
      acl: 'private', // 存储空间的默认读写权限为私有，即private。如果需要设置存储空间的读写权限为公共读，请替换为public-read。
      dataRedundancyType: 'LRS' // 存储空间的默认数据容灾类型为本地冗余存储，即LRS。如果需要设置数据容灾类型为同城冗余存储，请替换为ZRS。
    }
    // 填写Bucket名称。
    const result = await client.putBucket('examplebucket', options);
    console.log(result);
  } catch (err) {
    console.log(err);
  }
}

putBucket();
```

处于简便，后续实例不再写出完整函数，而是给出sdk方法和说明。

列举存储空间：

```
// 列举当前账号所有地域下的存储空间。
const result = await client.listBuckets();
// 列举当前账号所有地域下指定前缀的存储空间。
const result = await client.listBuckets({
    // 指定前缀。
    prefix: 'example'
});
// 列举当前账号所有地域下名称的字母序排在examplebucket之后的存储空间。
const result = await client.listBuckets({
    // 指定Marker。
    marker: 'examplebucket'
});
// 列举500个Bucket。
const result = await client.listBuckets({
    // max-keys用于限定此次返回Bucket的最大个数，max-keys取值不能大于1000。如果不指定max-keys，则默认返回100个Bucket。
    'max-keys': 500
});
```

查询存储空间信息：

```
async function getBucketInfo() {
  try {
      // 填写存储空间名称，例如examplebucket。
      const bucket = 'examplebucket'
      const result = await client.getBucketInfo(bucket)
      // 获取存储空间所在的地域。
      console.log(result.bucket.Location)
      // 获取存储空间的名称。
```

```
      console.log(result.bucket.Name)
      // 获取存储空间的拥有者ID。
      console.log(result.bucket.Owner.ID)
      // 获取存储空间的拥有者名称，目前和拥有者ID一致。
      console.log(result.bucket.Owner.DisplayName)
      // 获取存储空间的创建时间。
      console.log(result.bucket.CreationDate)
      // 获取存储空间的存储类型。
      console.log(result.bucket.StorageClass)
      // 获取存储空间的版本控制状态。
      console.log(result.bucket.Versioning)
  }  catch (error) {
    // 判断指定的存储空间是否存在。
      if (error.name === 'NoSuchBucketError') {
          console.log("No such bucket!")
      } else {
          console.log(error)
      }
}
```

删除存储空间

```
const result = await client.deleteBucket('yourbucketname');
```

文件上传

```
const headers = {
  // 指定该Object被下载时网页的缓存行为。
  // 'Cache-Control': 'no-cache',
  // 指定该Object被下载时的名称。
  // 'Content-Disposition': 'oss_download.txt',
  // 指定该Object被下载时的内容编码格式。
  // 'Content-Encoding': 'UTF-8',
  // 指定过期时间。
  // 'Expires': 'Wed, 08 Jul 2022 16:57:01 GMT',
  // 指定Object的存储类型。
  // 'x-oss-storage-class': 'Standard',
  // 指定Object的访问权限。
  // 'x-oss-object-acl': 'private',
  // 设置Object的标签，可同时设置多个标签。
  // 'x-oss-tagging': 'Tag1=1&Tag2=2',
  // 指定CopyObject操作时是否覆盖同名目标Object。此处设置为true，表示禁止覆盖同名Object。
  // 'x-oss-forbid-overwrite': 'true',
};

async function put () {
  try {
    // 填写OSS文件完整路径和本地文件的完整路径。OSS文件完整路径中不能包含Bucket名称。
    // 如果本地文件的完整路径中未指定本地路径，则默认从示例程序所属项目对应本地路径中上传文件。
    const result = await client.put('exampleobject.txt',
path.normalize('D:\\localpath\\examplefile.txt'));
    // const result = await client.put('exampleobject.txt',
path.normalize('D:\\localpath\\examplefile.txt'), { headers });
    console.log(result);
```

```
  } catch (e) {
    console.log(e);
  }
}
```

文件下载

```
const result = await client.get('exampleobject.txt',
'D:\\localpath\\examplefile.txt');
```

文件权限设置

```
// yourObjectName填写不包含Bucket名称在内的Object的完整路径。
const result = await client.getACL('yourObjectName');   // 获取权限
// yourObjectName填写不包含Bucket名称在内的Object的完整路径。
await client.putACL('yourObjectName', 'public-read');   // 修改权限
```

列举文件

```
const result = await client.list({
    // 设置按字母排序最多返回前10个文件。
    "max-keys": 10,
    // 列举文件名中包含前缀foo/的文件。
    prefix: 'foo/',
    // 设置正斜线（/）为文件夹的分隔符。
    delimiter: '/',
    // 列举文件名在xyz之后（字典序）的文件。
    marker: "xyz"
});
```

通过delimiter和prefix两个参数可以模拟文件夹功能：

- 如果设置prefix为某个文件夹名称，则会列举以此prefix开头的文件，即该文件夹下所有的文件和子文件夹（目录）均显示为objects。
- 如果在设置了prefix的情况下，将delimiter设置为正斜线（/），则只列举该文件夹下的文件和子文件夹（目录），该文件夹下的子文件夹（目录）显示为CommonPrefixes，子文件夹下的文件和文件夹不显示。

删除文件

```
// 填写Object完整路径。Object完整路径中不能包含Bucket名称。
let result = await client.delete('exampleobject.txt');
// 填写需要删除的多个Object完整路径并设置返回模式为详细模式。Object完整路径中不能包含Bucket名称。
let result = await client.deleteMulti(['exampleobject-1', 'exampleobject-
2','testfolder/sampleobject.txt']);
console.log(result);
// 填写需要删除的多个Object完整路径并设置返回模式为简单模式。Object完整路径中不能包含Bucket名称。
let result = await client.deleteMulti(['exampleobject-1', 'exampleobject-2',
'testfolder/sampleobject.txt'], {quiet: true});
console.log(result);
```

有关OSS SDK的相关内容还有很多，大家可以在需要使用时进行查询，官网文档上有详细介绍：https://help.aliyun.com/document_detail/32067.html。

CDN

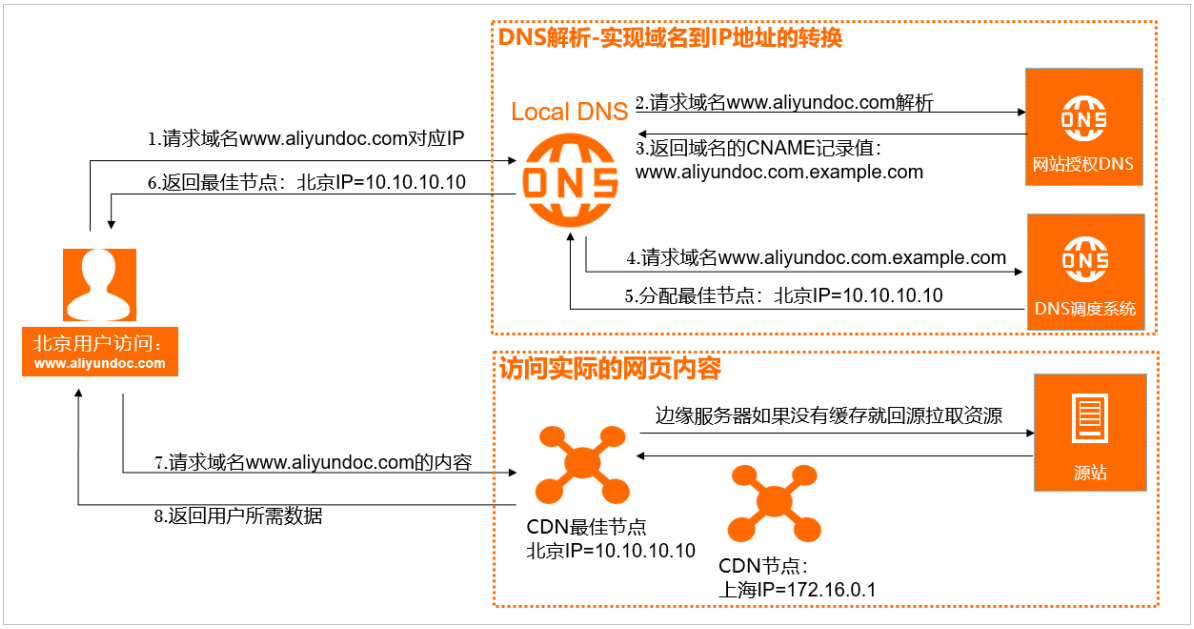
1.CDN简介

何为CDN?

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。

CDN工作流程



1. 当终端用户向`www.aliyundoc.com`下的指定资源发起请求时，首先向Local DNS（本地DNS）发起请求域名`www.aliyundoc.com`对应的IP。
2. Local DNS检查缓存中是否有`www.aliyundoc.com`的IP地址记录。如果有，则直接返回给终端用户；如果没有，则向网站授权DNS请求域名`www.aliyundoc.com`的解析记录。
3. 当网站授权DNS解析`www.aliyundoc.com`后，返回域名的CNAME `www.aliyundoc.com.example.com`。
4. Local DNS向阿里云CDN的DNS调度系统请求域名`www.aliyundoc.com.example.com`的解析记录，阿里云CDN的DNS调度系统将为其分配最佳节点IP地址。
5. Local DNS获取阿里云CDN的DNS调度系统返回的最佳节点IP地址。
6. Local DNS将最佳节点IP地址返回给用户，用户获取到最佳节点IP地址。
7. 用户向最佳节点IP地址发起对该资源的访问请求。

2.CDN配置

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。

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：https://help.aliyun.com/document_detail/27144.html。

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。

Web服务器

1.请求-响应流程与代理简介

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。

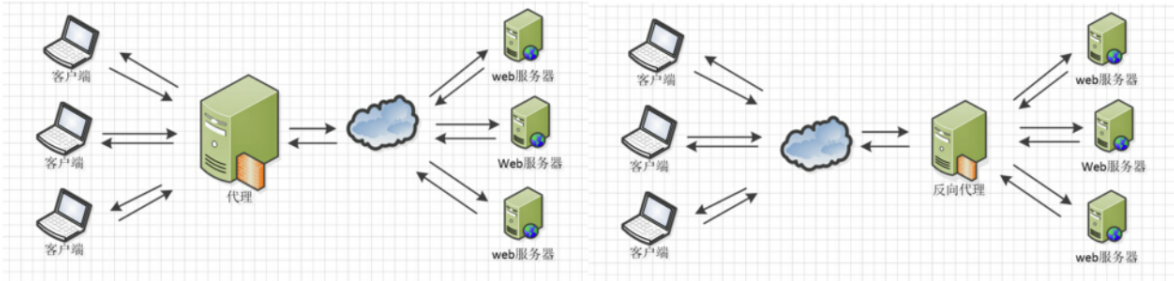
请求和响应

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

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。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。下面这张图比较形象：



在服务端使用反向代理可以带来诸多好处，例如通过隔离外部网络用户和内部服务器增强了内部服务器上数据的安全性，减小了系统受攻击的危险。

2.Nginx

Nginx安装与使用

Nginx是一款著名的Web和反向代理服务器，此外还有Apache等。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
```

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
```

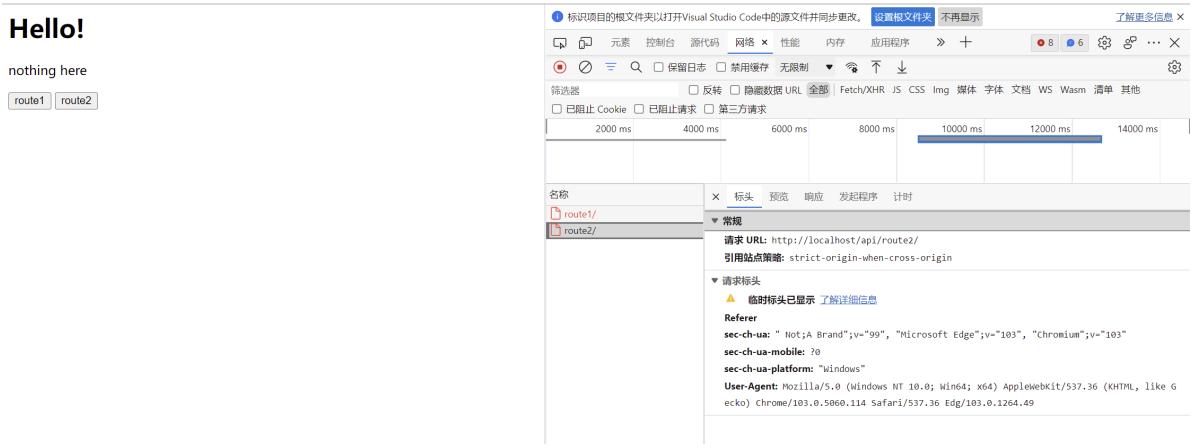
```
    keepalive_timeout 65;                   # 连接超时时间，默认为75s

    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>
```

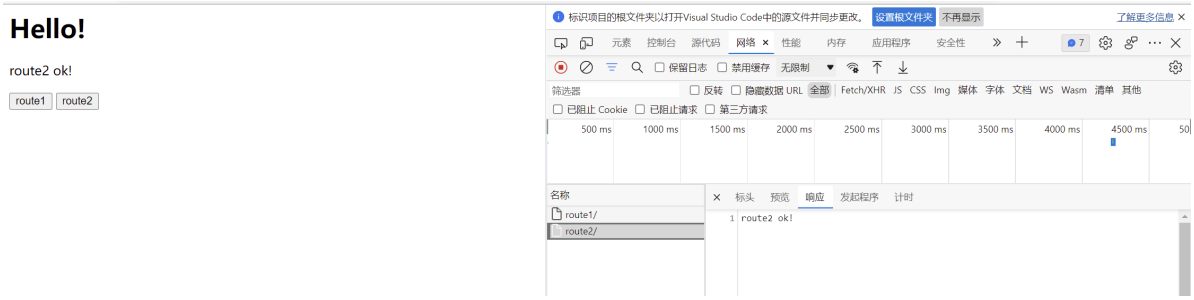
我们将上述代码写在index.html当中，作为前端的入口文件（事实上是唯一文件），然后保存在一个目录下，比如D:\server（路径中最好不要有空格、中文之类的，容易出错）。可以双击这一文件用浏览器预览一下，现在如果点击按钮则什么都不会发生。可以摁F12打开浏览器的开发人员工具（以Edge为例），在上方栏点更多找到网络标签，进去后再次点击按钮会发现请求失败。



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。



3.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。

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 / {
        # First attempt to serve request as file, then
        # as directory, then fall back to displaying a 404.
        try_files $uri $uri/ =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。

```
"scripts": {
  "start": "nodemon",
  "build": "babel src -d build -x \".ts\""
},
```

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：43.138.46.203），就能看到网站了。

以上介绍了一些基本操作，在实际部署中还要更复杂一些，但原理是类似的，感兴趣的同学建议查阅网络资料。