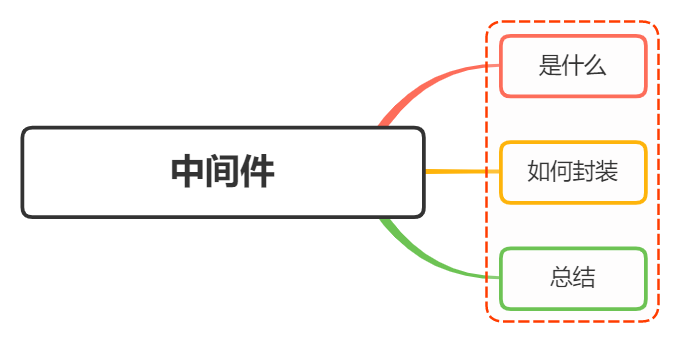
# 面试官：说说对中间件概念的理解，如何封装 node 中间件？

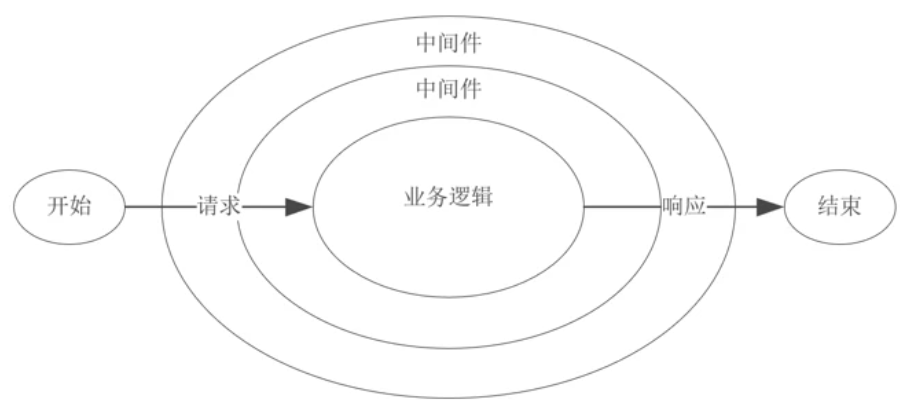


## 一、是什么

中间件（Middleware）是介于应用系统和系统软件之间的一类软件，它使用系统软件所提供的基础服务（功能），衔接网络上应用系统的各个部分或不同的应用，能够达到资源共享、功能共享的目的

在NodeJS中，中间件主要是指封装http请求细节处理的方法

例如在express、koa等web框架中，中间件的本质为一个回调函数，参数包含请求对象、响应对象和执行下一个中间件的函数



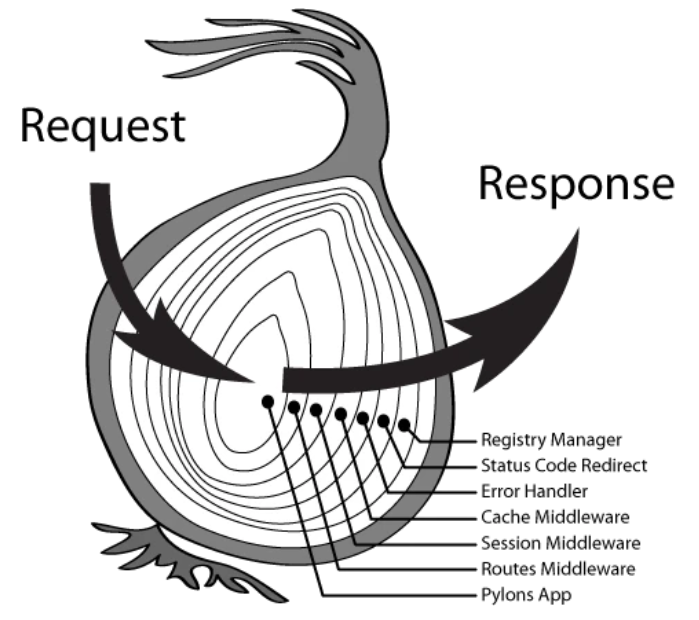
在这些中间件函数中，我们可以执行业务逻辑代码，修改请求和响应对象、返回响应数据等操作

## 二、封装

koa是基于NodeJS当前比较流行的web框架，本身支持的功能并不多，功能都可以通过中间件拓展实现。通过添加不同的中间件，实现不同的需求，从而构建一个 Koa 应用

Koa 中间件采用的是洋葱圈模型，每次执行下一个中间件传入两个参数：

* ctx ：封装了request 和 response 的变量
* next ：进入下一个要执行的中间件的函数



下面就针对koa进行中间件的封装：

Koa的中间件就是函数，可以是async 函数，或是普通函数

// async 函数  
app.use(async (ctx, next) => {  
 const start = Date.now();  
 await next();  
 const ms = Date.now() - start;  
 console.log(`${ctx.method} ${ctx.url} - ${ms}ms`);  
});  
  
// 普通函数  
app.use((ctx, next) => {  
 const start = Date.now();  
 return next().then(() => {  
 const ms = Date.now() - start;  
 console.log(`${ctx.method} ${ctx.url} - ${ms}ms`);  
 });  
});

下面则通过中间件封装http请求过程中几个常用的功能：

### token校验

module.exports = (options) => async (ctx, next) {  
 try {  
 // 获取 token  
 const token = ctx.header.authorization  
 if (token) {  
 try {  
 // verify 函数验证 token，并获取用户相关信息  
 await verify(token)  
 } catch (err) {  
 console.log(err)  
 }  
 }  
 // 进入下一个中间件  
 await next()  
 } catch (err) {  
 console.log(err)  
 }  
}

### 日志模块

const fs = require('fs')  
module.exports = (options) => async (ctx, next) => {  
 const startTime = Date.now()  
 const requestTime = new Date()  
 await next()  
 const ms = Date.now() - startTime;  
 let logout = `${ctx.request.ip} -- ${requestTime} -- ${ctx.method} -- ${ctx.url} -- ${ms}ms`;  
 // 输出日志文件  
 fs.appendFileSync('./log.txt', logout + '\n')  
}

Koa存在很多第三方的中间件，如koa-bodyparser、koa-static等

下面再来看看它们的大体的简单实现：

### koa-bodyparser

koa-bodyparser 中间件是将我们的 post 请求和表单提交的查询字符串转换成对象，并挂在 ctx.request.body 上，方便我们在其他中间件或接口处取值

// 文件：my-koa-bodyparser.js  
const querystring = require("querystring");  
  
module.exports = function bodyParser() {  
 return async (ctx, next) => {  
 await new Promise((resolve, reject) => {  
 // 存储数据的数组  
 let dataArr = [];  
  
 // 接收数据  
 ctx.req.on("data", data => dataArr.push(data));  
  
 // 整合数据并使用 Promise 成功  
 ctx.req.on("end", () => {  
 // 获取请求数据的类型 json 或表单  
 let contentType = ctx.get("Content-Type");  
  
 // 获取数据 Buffer 格式  
 let data = Buffer.concat(dataArr).toString();  
  
 if (contentType === "application/x-www-form-urlencoded") {  
 // 如果是表单提交，则将查询字符串转换成对象赋值给 ctx.request.body  
 ctx.request.body = querystring.parse(data);  
 } else if (contentType === "applaction/json") {  
 // 如果是 json，则将字符串格式的对象转换成对象赋值给 ctx.request.body  
 ctx.request.body = JSON.parse(data);  
 }  
  
 // 执行成功的回调  
 resolve();  
 });  
 });  
  
 // 继续向下执行  
 await next();  
 };  
};

### koa-static

koa-static 中间件的作用是在服务器接到请求时，帮我们处理静态文件

const fs = require("fs");  
const path = require("path");  
const mime = require("mime");  
const { promisify } = require("util");  
  
// 将 stat 和 access 转换成 Promise  
const stat = promisify(fs.stat);  
const access = promisify(fs.access)  
  
module.exports = function (dir) {  
 return async (ctx, next) => {  
 // 将访问的路由处理成绝对路径，这里要使用 join 因为有可能是 /  
 let realPath = path.join(dir, ctx.path);  
  
 try {  
 // 获取 stat 对象  
 let statObj = await stat(realPath);  
  
 // 如果是文件，则设置文件类型并直接响应内容，否则当作文件夹寻找 index.html  
 if (statObj.isFile()) {  
 ctx.set("Content-Type", `${mime.getType()};charset=utf8`);  
 ctx.body = fs.createReadStream(realPath);  
 } else {  
 let filename = path.join(realPath, "index.html");  
  
 // 如果不存在该文件则执行 catch 中的 next 交给其他中间件处理  
 await access(filename);  
  
 // 存在设置文件类型并响应内容  
 ctx.set("Content-Type", "text/html;charset=utf8");  
 ctx.body = fs.createReadStream(filename);  
 }  
 } catch (e) {  
 await next();  
 }  
 }  
}

## 三、总结

在实现中间件时候，单个中间件应该足够简单，职责单一，中间件的代码编写应该高效，必要的时候通过缓存重复获取数据

koa本身比较简洁，但是通过中间件的机制能够实现各种所需要的功能，使得web应用具备良好的可拓展性和组合性

通过将公共逻辑的处理编写在中间件中，可以不用在每一个接口回调中做相同的代码编写，减少了冗杂代码，过程就如装饰者模式

## 参考文献

* https://segmentfault.com/a/1190000017897279
* https://www.jianshu.com/p/81b6ebc0dd85
* https://baike.baidu.com/item/%E4%B8%AD%E9%97%B4%E4%BB%B6