# 利用Swagger UI生成本地mock数据

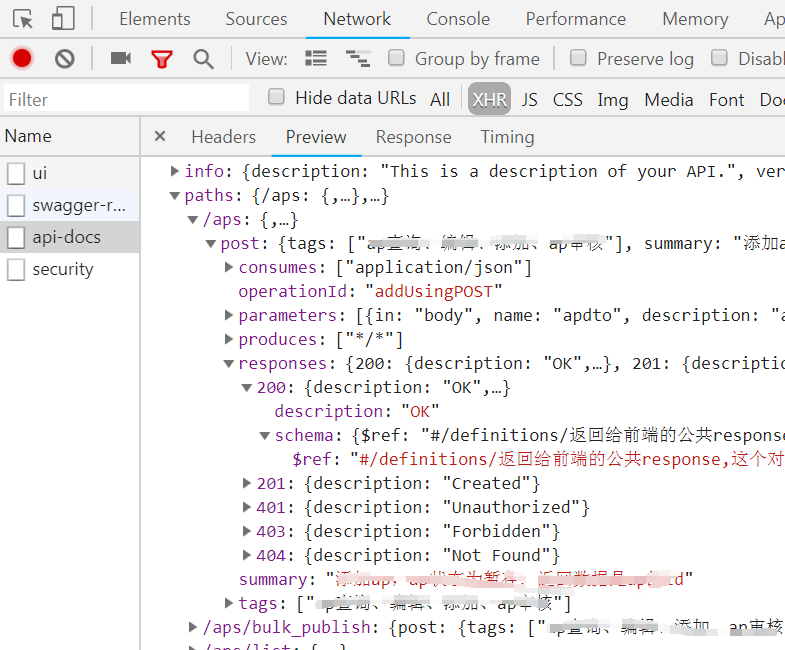
附件：[示例代码](https://github.com/zzf03680147/synchronizeSwagger)

### 背景

mock作为前端开发重要的一环，可以带来诸多好处：

* **前后端并行开发**
* **模拟各种响应，易于测试**
* **及早发现一些极端情况下的页面布局问题**
* ...

yapi、easymock等接口管理平台都提供了Swagger、Postman数据导入功能，原理大同小异，无非就是解析json文件来生成相应的api。以Swagger为例，打开network会发现有个api-docs文件:



可以想见，如果能适时同步后端部署的接口，将省去前端不少体力活。我们就以此文件为基础，来试着构建本地的mock数据。

### 目标

* **api path和mock文件目录相对应，便于查找、修改**
* **以api method为文件名,一个方法对应一个文件，减少多人编辑冲突**
* **使用mockjs包装响应值，易于模拟一些极端状况**

### Talk is cheap

#### ①解析json文件

从上图可以发现解析json文件，主要的工作在响应值类型的转换，这边我们利用easymock的一个解析模块来做这件事情。

const swaggerParserMock = require('swagger-parser-mock');  
  
 const synchronizeSwagger = {  
 init({ url, blacklist, outputPath }) {  
 this.url = url;  
 this.blacklist = blacklist;  
 this.outputPath = outputPath;  
 this.parse();  
 },  
 async parse() {  
 const { paths } = await swaggerParserMock(this.url);  
 this.generate(paths);  
 console.log(paths);  
 }  
 }  
   
 synchronizeSwagger.init({  
 // Swagger api-docs地址  
 "url":"https://raw.githubusercontent.com/zzf03680147/synchronizeSwagger/master/swagger.json",  
 // 输出目录  
 "outputPath": "./routes",  
 // 黑名单，跳过一些不需要同步的api  
 "blacklist": []  
 });

打印paths信息，格式大致如下：

"/path/foo": {  
 "get": {  
 "summary": "bar",  
 "responses": {  
 "200": {  
 "example": "'@string'" // 模块为我们做的类型转化和mock包装。  
 }  
 }  
 },  
 "post": {  
 "summary": "baz",  
 "responses": {  
 "200": {  
 "example": "'@string'"  
 }  
 }  
 }  
 }

#### ②生成文件

const fs = require('fs');  
 const { join } = require('path');  
 const { promisify } = require('util');  
 const mkdirp = require('mkdirp');  
  
 const writeFile = promisify(fs.writeFile);  
 const mkdir = promisify(mkdirp);  
  
 const synchronizeSwagger = {  
  
 // 遍历api path信息  
 traverse(paths) {  
 for (let path in paths) {  
 if (this.blacklist.includes(path)) {  
 continue;  
 }  
  
 for (let method in paths[path]) {  
 const pathInfo = paths[path][method];  
  
 if (!pathInfo['responses']['200']) {  
 continue;  
 }  
 this.generate(path, method, pathInfo);  
 }  
 }  
 },  
  
 // 生成mock文件  
 async generate(path, method, pathInfo) {  
 const outputPath = join(\_\_dirname, this.outputPath, path);  
 const {  
 summary,  
 responses: { 200: responseOK },  
 } = pathInfo;  
  
 try {  
 // 生成目录  
 await mkdir(outputPath);  
  
 const example = responseOK['example'];  
 // 生成文件内容  
 const template = this.generateTemplate({  
 summary,  
 example,  
 method,  
 path,  
 });  
  
 // 生成文件, 已存在的跳过，避免覆盖本地以及编辑的文件  
 const fPath = join(outputPath, `${method}.js`);  
 await writeFile(fPath, template, { flag: 'wx' });  
 console.log(`增加Mock文件：${fPath}`);  
 } catch (error) {  
 /\* eslint-disable no-empty \*/  
 }  
 },  
  
 generateTemplate({ summary, example, method, path }) {  
 // prettier-ignore  
 // api path中的{petId}形式改为:petId  
 return `/\*\*  
 ${summary}  
\*\*/  
const Mock = require("mockjs");  
module.exports = function (app) {  
 app.${method}('/api${path.replace(/\{([^}]\*)\}/g, ":$1")}', (req, res) => {  
 res.json(Mock.mock(${example}));  
 });  
};`;  
 },  
}

#### ③启动服务

以express为例，利用require动态特征我们来创建路由。

const fs = require('fs');  
const join = require('path').join;  
const express = require('express');  
  
const app = express();  
const port = process.env.PORT || 3000;  
  
app.listen(port, function() {  
 console.log(`server is listening ${port}`);  
});  
  
  
function scan(path, app) {  
 const files = fs.readdirSync(path);  
  
 for (let i = 0; i < files.length; i++) {  
 const fpath = join(path, files[i]);  
 const stats = fs.statSync(fpath);  
  
 if (stats.isDirectory()) {  
 scan(fpath, app);  
 }  
 if (stats.isFile()) {  
 require(fpath)(app);  
 }  
 }  
}  
  
scan(join(\_\_dirname, './routes'), app);

### 写在最后

至此我们就利用Swagger UI生成了本地mock数据，如果再加上跨域、body-parser等middleware, 一个mock服务就基本成形。方便同步，我们还可以将它加入npm scripts：

"scripts": {  
 "ss": "node ./synchronizeSwagger.js"  
 },

执行npm run ss，就能生成相应的mock数据和访问api了：

