Jarvis——前后端对接解决方案

2018

对接痛点

重复性工作

依赖后端服务

文档不规范

Jarvis 运行流程

前端

接口文档

校验接口文档

Jarvis

.ts

接口 sdk

.ts

mock 数据结构

.ts

数据模型

.ts

枚举值

对接模式

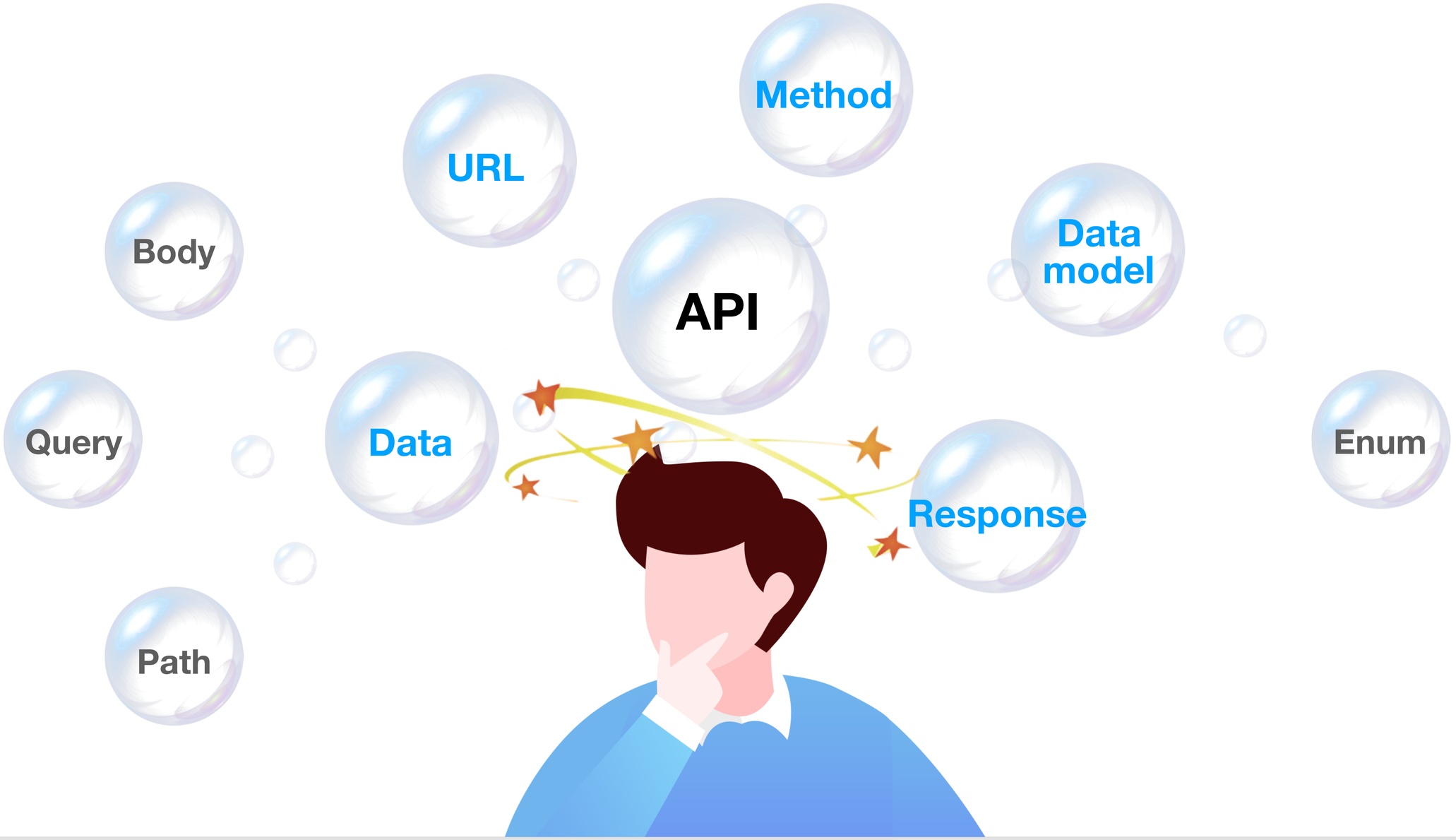
声明接口方法

```
export function deleteData(id) {
  return agent.delete(`/api/data/${id}`)
}

export function getData(options) {
  return agent.get('/api/data')
    .query(options)
    .then(res => res.body)
}
export function CreateDta({ projectId, contract }) {
  return agent.post(`/api/data/${projectId}/user`)
    .send(contract)
}
```

接口文档

现有调用模式


Method
URL
Body
API
Data model
Query
Data
Enum
Response
Path

terminus

Jarvis API SDK

1. 输出格式分为 ts 和 es。

ts 格式

```ts
export const getList = (payload: {
  status: getListStatus;
  id: number;
}): Promise<ResponseModal> => {
  // @ts-ignore
  const url = '/getList';
  if (openMock) {...}
  return request
    .get(url).query(payload || {})
    .use((sp) => {...})
    .then((response: any) => {...})
    .catch((e: any) => {...});
};
```

1. 方法名称依据 http method + service name + url
2. 在调用时只需要按要求传参即可
3. 包含后端数据结构以及枚举值
4. 使用 SDK 代替 查看接口文档

terminus

Jarvis API SDK

Interface

```ts
export interface ResponseModal {
    name: string;
    status: getListStatus;
    description: string;
}
```

Enum

```ts
export enum getListStatus {
    NONE = 'NONE',
    PLATFORM = 'PLATFORM',
    SHOP = 'SHOP',
    BUSINESS = 'BUSINESS',
}
```

Jarvis API SDK

es 格式

sdk.d.ts

```ts
/**
 * @description create
 * API: /api/org/dimension/create
 */
export declare function createPmpOrgDimensionCreate(payload: {

    /**
     * name
     */
    name?: string,

    /**
     * key
     */
    key?: string,

    /**
     * projectId
     */
    projectId?: string,

    /**
     * type
     */
    type?: string,

}): Responseboolean;
```

sdk.js

```js
/**
 * @description create
 * API: `/api/org/dimension/create`
 */
export const createPmpOrgDimensionCreate = (payload) => {
  // @ts-ignore
  const url = `/api/org/dimension/create`
  // @ts-ignore
  return new Promise((resolve, reject) => {
    request
      .post(url)
      .send(payload)
      .use(setPluginsAndHeaders)
      .then((response) => {...})
      .catch((e) => {...})
  })
}
```

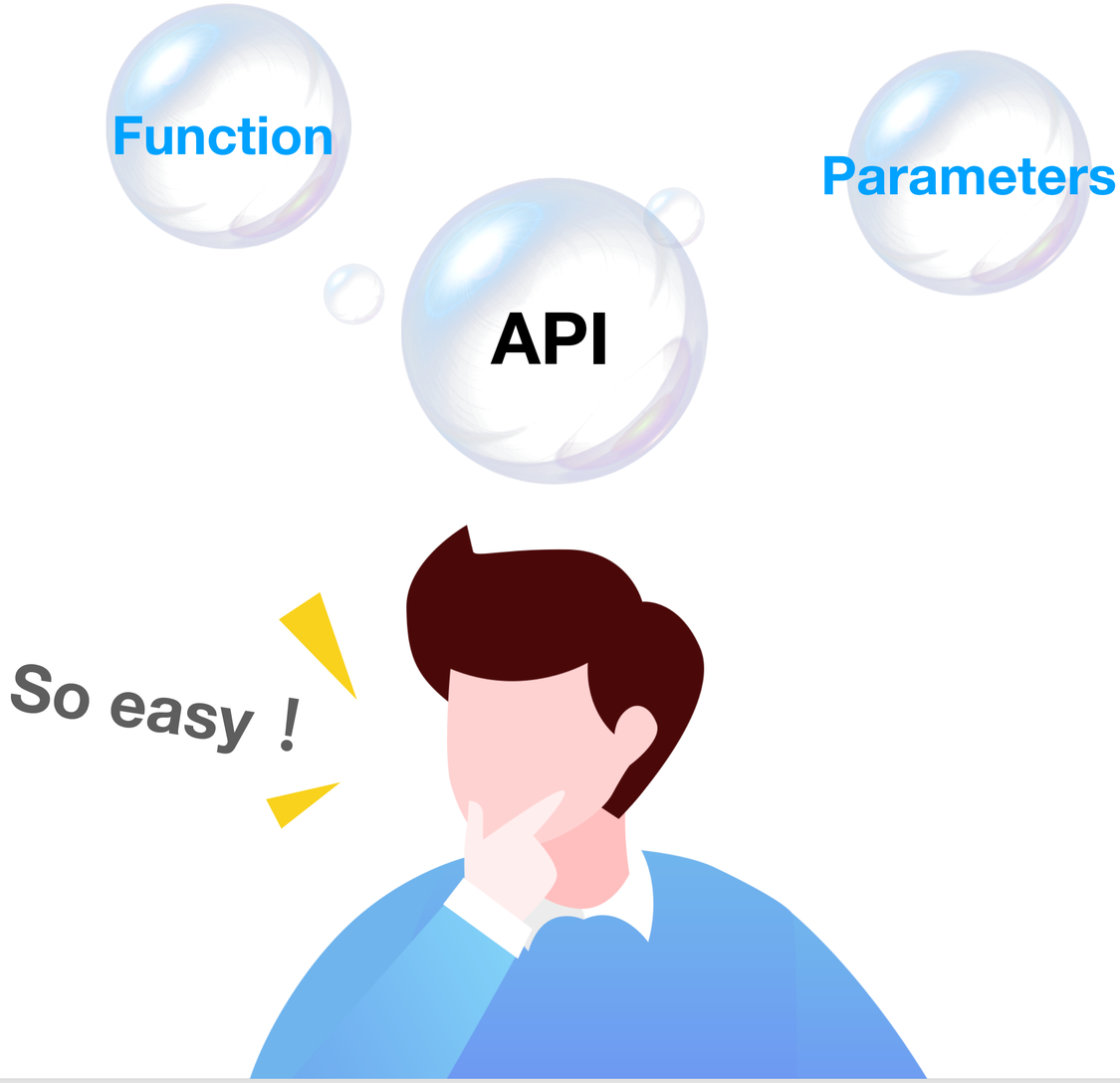
Jarvis API SDK

es 模版中枚举是这样的

```
export const CompanyBaseInfoVOType = {
  SUPPLIER: 'SUPPLIER',
  PURCHASER: 'PURCHASER',
  SUPPLIER_AND_PURCHASER: 'SUPPLIER_AND_PURCHASER',
  VIRTUAL_SUPPLIER: 'VIRTUAL_SUPPLIER',
  STOCK_ORG: 'STOCK_ORG',
  DEALER: 'DEALER',
};
```

terminus

Jarvis API SDK



Jarvis——依赖后端服务

Jarvis 也提供了 Mock 的功能！

terminus

Mock 的现状

1. Mock 和 后端服务只能二选一

2. 需要配置 Mock 数据

Jarvis Mock

1. 零配置

Jarvis Mock 是本地 Mock，不依赖其他服务，比较稳定。

Jarvis Mock 不需要配置接口返回结构。

2. 按需 Mock

Jarvis Mock 支持按需 Mock 请求。当后端服务不可用时，才使用 Mock 功能

3. 可定制 Mock 数据类型

默认情况下， 数据是随机的。如果对数据类型有一定要求，Jarvis Mock 提供mock 策略功能。

Jarvis Mock 使用方法

1. **sdk.ts 提供一个 init 方法**

   openMock: true. 全流量 mock

   openMock: [‘errorMock’, ‘404’, ‘502’, ‘503’] 当后端返回的状态码是这些的时候开启mock

terminus

Jarvis Mock 策略使用方法

Mock 策略配置

```
import { init,  getSomeDataAPI } from 'sdk.ts'

await init({
    // 开启mock
    openMock: true,
    mockStrategies: {
        // 全局配置
        '*': {
            createdAt: {
                type: 'date',
            }
        },
        getSomeDataAPI: {
            // 字段
            status: {
                // string, address, number, boolean, name, cname, image
                type: 'enum',
                enum: [ 'type1', 'type2' ]
            }
        }
    }
})

await getSomeDataAPI();
```

terminus

Jarvis 应用

1. npm i -g jarvis
2. jarvis init，配置 .jarvis.yml

```yaml
# 所需要替换的 clients 脚本路径列表
swaggers:
# 后端swagger地址, url请带上 /v2/api-docs
- swagger_url: http://www.test.com/v2/api-docs
  # 显示在通知上的。
  alias: service_name
  # 生成的 client 存放路径, 相对路径
  target_path: ./test-app/src/notice-sdk.js
# api client 生成的类型, 现在仅支持 js ts
target_language: js
# 所依赖的请求代理模块, default:  null | superagent | custom
template: default
# 自定义template模版文件相对路径, 执行 jarvis template 获取模版.
template_config: null
```

Jarvis 应用

2. 使用 jarvis 命令，在指定目录生成 sdk 文件

3. 调用接口，如下

```jsx
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';
import { init, getPmpProjectProjectIdOrgList } from './notice-sdk';

class App extends Component {
  async componentWillMount() {
    await init({
      openMock: true,
      headers: [
        {
          key: 'test-key',
          value: 'test',
        },
      ],
      onError() {
        console.error('error');
      },
      onResponse(response) {
        return response.result;
      },
    });

    const response = await getPmpProjectProjectIdOrgList({
      projectId: 1,
    });

    response.data.list.forEach((item) => {
      console.log(item.index);
    });
  }
  render() {
    return (
      <div className="App">
        <header className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <p>
            Edit <code>src/App.js</code> and save to reload.
          </p>
          <a
            className="App-link"
```

Q&A