本文主要介绍如何通过supertest发起请求，进行前端接口测试

前端接口测试

前端接口测试：是在访问页面时，模拟前端页面会发向后端起一些http的请求，并进行测试验证的过程。

优势：

1-登陆过程可以通过UI自动化模拟

2-可以配合UI自动化测试，进行接口和展示层面的双层校验

3-比单纯的http接口测试，使得测试人员更清楚前后端的真实交互过程

4-对于无后端测试经验的同学来说更加容易上手

1-技术选型

Http Ajax技术可选方案：axios、superagent、request、fetch、supertest

方案	优点	缺点
request	API 简单易用	不基于 Promise
axios	同时支持 Node.js 和浏览器 支持 Promise API 可以配置或取消请求 可以设置响应超时 支持防止跨站点请求伪造（XSRF）攻击 可以拦截未执行的请求或响应 支持显示上传进度 广泛用于 React 和 Vue 项目	用起来比较麻烦
supertest	流畅的 API 简单的 HTTP 断言 可以与 Chai.js 和 Mocha 等不同的测试套件混用	不支持浏览器
superagent	它有一个插件生态，通过构建插件可以实现更多功能 可配置 HTTP 请求发送接口友好 可以为请求链式添加方法 适用于浏览器和 Node	没什么缺点
Fetch	灵活易用 使用 Promise 避免回调地狱 支持所有现代浏览器 遵循 request-response 方案 语法简单清晰 支持 React Native	不支持服务器端使用 缺乏开发库的亮点功能，比如取消请求 没有内置默认值，如请求模式，请求头，请求凭据。

选择supertest主要原因：

Supertest 用于测试 Node.js HTTP 服务器。 该库由 SuperAgent 提供支持，它把自身的 API 和 SuperAgent 的底层 API 相结合，提供简洁的 HTTP 测试接口。

1-由完美的Superagent驱动

2-方便与 Chai.js 和 Mocha 等不同的测试套件混用

2-supertest例子

新建：test/interface/OnlineApiTest.js

在线的http协议测试，可以使用：https://jsonplaceholder.typicode.com/guide/ 提供的GET，POST，PUT，DELETE方法

get方法

测试例子：

Method：get

URL： https://jsonplaceholder.typicode.com/posts/2

response：

```
{
    "userId": 1,
    "id": 2,
    "title": "qui est esse",
    "body": "est rerum tempore vitae\nsequi sint nihil reprehenderit dolor beatae ea dolores neque\nfugiat blanditiis voluptate porro vel nihil molestiae ut reiciendis\nqui aperiam non debitis possimus qui neque nisi nulla"
}
```

测试代码：

```
const {describe,it,before}=require('mocha');
const Base=require('../Base');
const request = require("supertest");
const {expect,assert}=require('chai');

describe('OnlineAPI测试',function () {
    before(Base.before);
    this.timeout(20000);
    it('1-GET方法', async function () {
        const get_url = "https://jsonplaceholder.typicode.com";
        await request(get_url)
            .get('/posts/2')
            .set('Accept','application/json')
            .expect(200)
            .expect('Content-Type', 'application/json; charset=utf-8')
            .then((response) => {
                console.log('res'+JSON.stringify(response));
                console.log('res.body='+JSON.stringify(response.body));
                assert.equal(response.body.userId,1);
                assert.equal(response.body.id,2);
                assert.equal(response.body.title,'qui est esse');
                expect(response.body).to.have.property('title');
                })
            .catch((err) => {
                assert.fail(err);
            });
        ;
    })
})
```

post方法

测试例子：

Method：post

URL： https://jsonplaceholder.typicode.com/posts

请求体 : {"title":"foo","body":"bar","userId":1}

response：

```
{
    "title": "foo",
    "body": "bar",
    "userId": 1,
    "id": 101
}
```

测试代码：

```
it('2-POST方法', async function () {
    var body="{\"title\":\"foo\",\"body\":\"bar\",\"userId\":1}";
    const post_url = "https://jsonplaceholder.typicode.com";
    await request(post_url)
        .post('/posts')
        .send(body)
        .set('Accept','application/json')
        .expect(201)
        .expect('Content-Type', 'application/json; charset=utf-8')
        .then((response) => {
            console.log('res'+JSON.stringify(response));
            console.log('res.body='+JSON.stringify(response.body));
            assert.equal(response.body.id,101);
        })
        .catch((err) => {
            assert.fail(err);
        });
    ;
})
```

put方法

测试例子：

Method：put

URL： https://jsonplaceholder.typicode.com/posts/1

请求体 : {"title":"foo","body":"bar","userId":1}

response：

```
{
    "title": "foo",
    "body": "bar",
    "userId": 1,
    "id": 1
}
```

测试代码：

```
it('3-PUT方法', async function () {
    const put_url = "https://jsonplaceholder.typicode.com";
    const body = "{\"title\":\"foo\",\"body\":\"bar\",\"userId\":1}";
    await request(put_url)
        .put('/posts/1')
        .send(body)
        .set('Accept','application/json')
        .expect(200)
        .expect('Content-Type', 'application/json; charset=utf-8')
        .then((response) => {
            console.log('res'+JSON.stringify(response));
            console.log('res.body='+JSON.stringify(response.body));
            assert.equal(response.body.id,1);
        })
        .catch((err) => {
            assert.fail(err);
        });
    ;
})
```

delete方法

测试例子：

Method：delete

URL： https://jsonplaceholder.typicode.com/posts/1

response：

```
{}
```

测试代码：

```
it('4-DELETE方法', async function () {
    const delete_url = "https://jsonplaceholder.typicode.com";
    await request(delete_url)
        .del('/posts/1')
```

```
        .set('Accept', 'application/json')
        .expect(200)
        .expect('Content-Type', 'application/json; charset=utf-8')
        .then((response) => {
            console.log('res' + JSON.stringify(response));
            console.log('res.body=' + JSON.stringify(response.body));
            assert.equal(JSON.stringify(response.body), '{}');
        })
        .catch((err) => {
            assert.fail(err);
        });
    ;
})
```

完整代码

OnlineApiTest.js完整代码

```
const {describe, it, before} = require('mocha');
const Base = require('../Base');
const request = require("supertest");
const {expect, assert} = require('chai');

describe('OnlineAPI测试', function () {
    before(Base.before);
    this.timeout(20000);
    it('1-GET方法', async function () {
        const get_url = "https://jsonplaceholder.typicode.com";
        await request(get_url)
            .get('/posts/2')
            .set('Accept', 'application/json')
            .expect(200)
            .expect('Content-Type', 'application/json; charset=utf-8')
            .then((response) => {
                console.log('res' + JSON.stringify(response));
                console.log('res.body=' + JSON.stringify(response.body));
                assert.equal(response.body.userId, 1);
                assert.equal(response.body.id, 2);
                assert.equal(response.body.title, 'qui est esse');
                expect(response.body).to.have.property('title');
            })
            .catch((err) => {
                assert.fail(err);
            });
        ;
    })
    it('2-POST方法', async function () {
        const body = "{\"title\":\"foo\",\"body\":\"bar\",\"userId\":1}";
        const post_url = "https://jsonplaceholder.typicode.com";
```

```
        await request(post_url)
            .post('/posts')
            .send(body)
            .set('Accept', 'application/json')
            .expect(201)
            .expect('Content-Type', 'application/json; charset=utf-8')
            .then((response) => {
                console.log('res' + JSON.stringify(response));
                console.log('res.body=' + JSON.stringify(response.body));
                assert.equal(response.body.id, 101);
            })
            .catch((err) => {
                assert.fail(err);
            });
        ;
    })
    it('3-PUT方法', async function () {
        const put_url = "https://jsonplaceholder.typicode.com";
        const body = "{\"title\":\"foo\",\"body\":\"bar\",\"userId\":1}";
        await request(put_url)
            .put('/posts/1')
            .send(body)
            .set('Accept', 'application/json')
            .expect(200)
            .expect('Content-Type', 'application/json; charset=utf-8')
            .then((response) => {
                console.log('res' + JSON.stringify(response));
                console.log('res.body=' + JSON.stringify(response.body));
                assert.equal(response.body.id, 1);
            })
            .catch((err) => {
                assert.fail(err);
            });
        ;
    })
    it('4-DELETE方法', async function () {
        const delete_url = "https://jsonplaceholder.typicode.com";
        await request(delete_url)
            .del('/posts/1')
            .set('Accept', 'application/json')
            .expect(200)
            .expect('Content-Type', 'application/json; charset=utf-8')
            .then((response) => {
                console.log('res' + JSON.stringify(response));
                console.log('res.body=' + JSON.stringify(response.body));
                assert.equal(JSON.stringify(response.body), '{}');
            })
            .catch((err) => {
                assert.fail(err);
```

```
            });
        ;
    })
})
```

3-supertest api

1-method方法

request(base_url).del('/posts/1')

常见方法：get,post,del,put,head

支持的方法

```
'get',
'post',
'put',
'head',
'delete',
'options',
'trace',
'copy',
'lock',
'mkcol',
'move',
'purge',
'propfind',
'proppatch',
'unlock',
'report',
'mkactivity',
'checkout',
'merge',
'm-search',
'notify',
'subscribe',
'unsubscribe',
'patch',
'search',
'connect'
```

2-set():设置请求头

功能：设置请求头信息，一般在post请求时需要设置

3-send():设置请求体

功能：设置请求体，post，put接口需要设置请求体

4-expect()：响应断言

```
* Expectations:
*   .expect(200)//code=200
*   .expect(200, fn)
*   .expect(200, body)
*   .expect('body')
*   .expect('body', fn)
*   .expect(['json array body', { key: 'val' }])
*   .expect('Content-Type', 'application/json')
*   .expect('Content-Type', 'application/json', fn)
*   .expect(fn)
*   .expect([200, 404]) //code in 200,404
```

可以直接进行响应断言：响应code，响应headers，响应体

说明：响应体的完整断言由于JSON格式问题对比容易出错，不建议直接在expect中进行body的校验，可以放到下面的then函数进行处理；

5-then()：断言处理

可以作为回调处理，在里面进行响应信息的详细断言

6-catch()：异常捕捉

异常信息处理

4-工具化

在框架的登陆处理阶段，我们把登陆cookie持久化并且存到global.cookieStr变量中，因此可以进行在此基础上进行工具的封装（免登录）。

新建：src/utils/Api.js

```
const request = require("supertest");
const {assert}=require('chai');

/**
 * 封装常见的API
 */
const Api = {
    get: async function (url) {
        return request(url).get('')
            .set('Accept', 'application/json, text/javascript')
            .set('Cookie', global.cookieStr)
```

```
            .catch((err) => {
                assert.fail(err);
            });
    },
    post: async function (url, body) {
        return request(url).post('')
            .set('Accept','application/json, text/javascript')
            .set('Cookie',global.cookieStr)
            .send(body)
            .catch((err) => {
                assert.fail(err);
            });
    },
    put: async function (url, body) {
        return request(url).put('')
            .set('Accept','application/json, text/javascript')
            .set('Cookie',global.cookieStr)
            .send(body)
            .catch((err) => {
                assert.fail(err);
            });
    },
    delete: async function (url) {
        return request(url).del('')
            .set('Accept','application/json, text/javascript')
            .set('Cookie',global.cookieStr)
            .catch((err) => {
                assert.fail(err);
            });
    }
};
module.exports = Api;
```

5-前端接口测试

在http接口测试中，最为复杂的就是后端会进行登陆校验，因此前端的请求必须是在登陆后才可以发起，这个时候我们可选的方式有2种：1-通过UI的方式登陆后，写入cookie，设置global.cookieStr；2-通过登陆接口请求， 后续的接口带上cookie；

方法1:UI登陆并设置全局cookie参数

在cookie登陆章节已经处理过global.cookieStr参数，存放的就是浏览器访问时需要的cookie字符串，因此在每次请求时可以通过set带上cookie即可

新建：test/interface/BlogApiTest.js

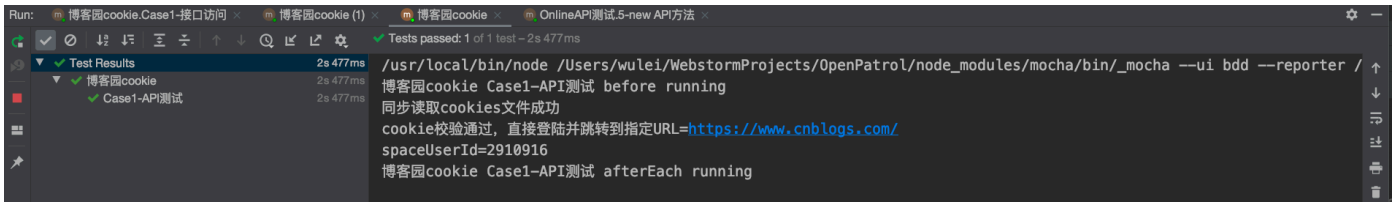
```
const {describe,it,before,after,afterEach}=require('mocha');
```

```
const Base=require('../Base');
const loginCookie = require('../../src/cookie/HandleCookie');
const {assert}=require('chai');
const Api=require('../../src/utils/Api')

describe('博客园cookie',function () {
    before(Base.before);
    // after(Base.after);
    afterEach(Base.afterEach);
    this.timeout(1000000);
    it('Case1-API测试', async function () {
        const userInfo="https://www.cnblogs.com/";
        await loginCookie.loginBlogCheck(userInfo,null,null);
        // 访问userinfo接口-get
        let spaceUserId;//参数提取
        await Api.get('https://account.cnblogs.com/user/userinfo')
            .then((response) => {
                spaceUserId=response.body.spaceUserId;
                assert.equal(response.status, 200);
            });
        console.log('spaceUserId='+spaceUserId);

        //postIds接口-post接口
        await Api.post('https://www.cnblogs.com/aggsite/MyDigged/postIds','')
            .then((response) => {
                assert.equal(response.status, 200);
            });
    })
})
```

可以看到接口访问时正常的



方法2:传统接口测试

传统的接口测试，一般是在通过登陆接口（post接口）登陆后进行cookie的处理（可以参考前面的持久化），本质是就是将UI登陆持久化cookie变换为登陆接口登陆并持久化cookie。可以参考OnlineApiTest.js 的post接口方法进行登录后，保存cookie即可。