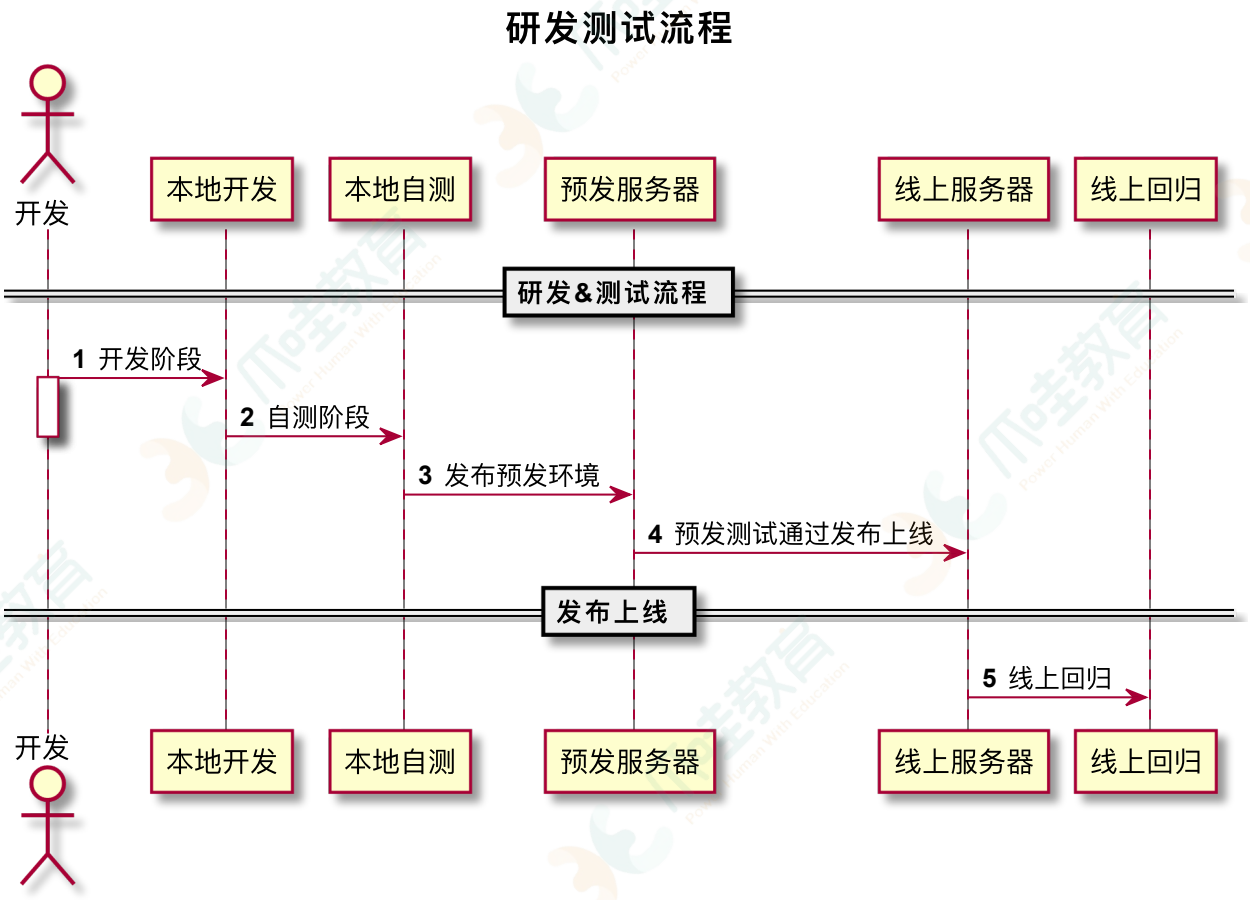
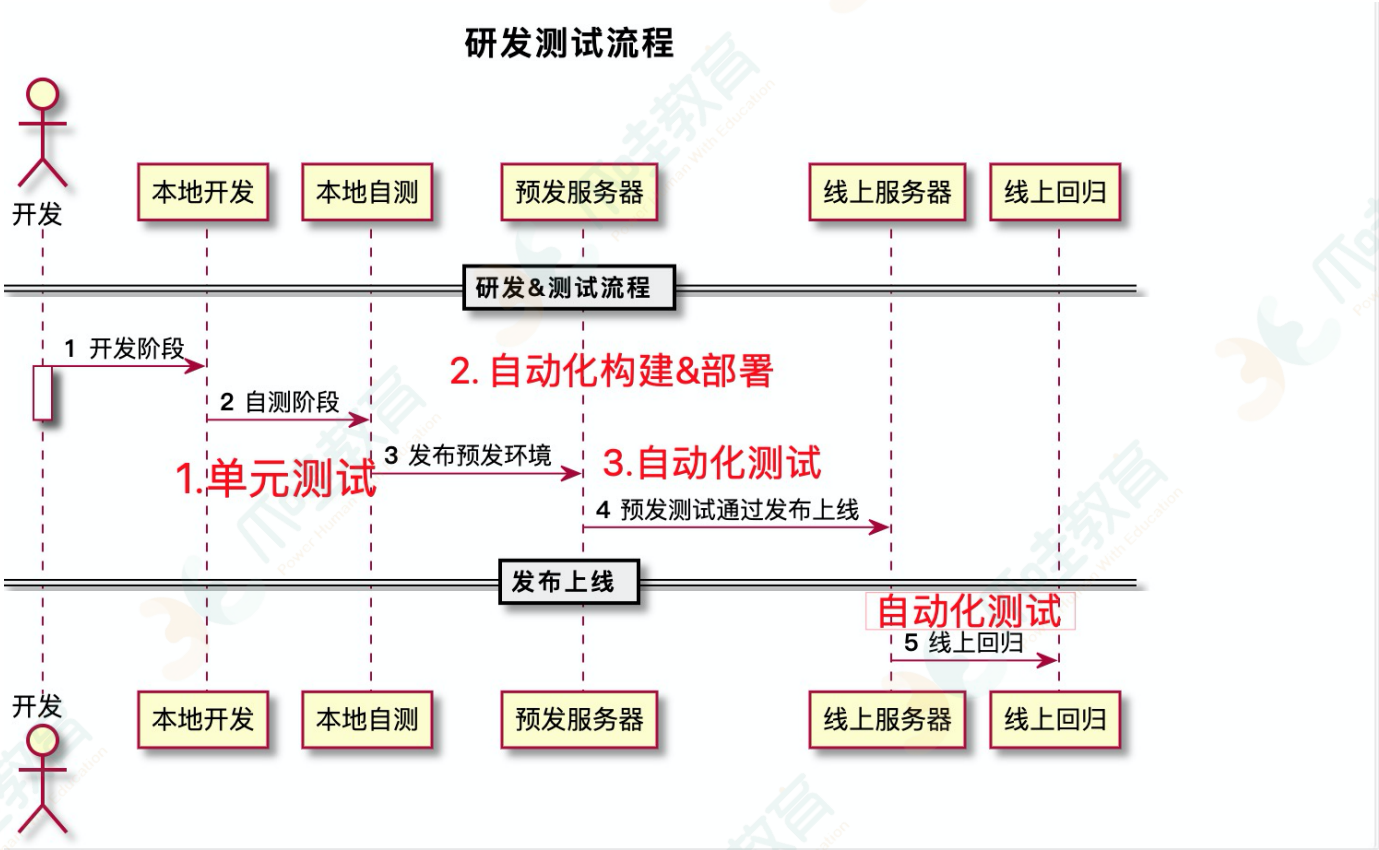
工程化-自动化测试&部署

自动化测试工具&原理

正常流程



加上自动化流程



单元测试

单元测试是指对软件中最小可测试单元进行检查验证，也称作模块测试。在nodeJs中通常是指对某个函数或者API进行正确验证，以保证代码的可用性。

单元测试有很多种，常见的有：行为驱动开发（BDD）和 测试驱动开发（TDD）。

- 行为驱动开发（BDD）：行为驱动开发关注的是整个系统的最终实现是否和用户期望一致。
- 测试驱动开发（TDD）：测试驱动开发的目的是取得快速反馈，使所有功能都是可用的。

> https://mochajs.cn/ Mocha.js是被广泛使用的JavaScript测试框架，在浏览器和Node环境都可以使用， 测试风格又有两种 BDD和TDD https://www.zhihu.com/question/20161970

mocha

Mocha 支持 BDD/TDD 等多种测试风格，默认使用 BDD 接口。BDD（行为驱动开发）是一种以需求为导向的敏捷开发方法，相比主张”测试先行“的 TDD（测试驱动开发）而言，它强调”需求先行“，从一个更加宏观的角度去关注包括开发、QA、需求方在内的多方利益相关者的协作关系，力求让开发者“做正确的事“。

原理：

- describe：行为描述，代表一个测试块要干什么，是一组测试单元的集合； 源码
- it：描述了一个测试单元，是最小的测试单位；
- before：Hook 函数，在执行该测试块之前执行；

- after：Hook 函数，在执行该测试块之后执行；
- beforeEach：Hook 函数，在执行该测试块中每个测试单元之前执行；
- afterEach：Hook 函数，在执行该测试块中每个测试单元之后执行。

一个简单的例子

```
describe('Array', function () {
  describe('#indexOf()', function () {
    it('should return -1 when the value is not present', function
() {
      assert.equal([1, 2, 3].indexOf(4), -1);
    });
  });

});
```

异步代码测试

> 比如我们需要测试异步执行接口的返回。只需要在用例函数里边加一个done回调，异步代码执行完毕后调用一下done，就可以通知mocha，我执行完啦，去执行下一个用例函数

```
describe('Ajax', function () {
    it('should success', function (done) {
      Ajax().then((res) => {
        console.log('111', res.success);
        assert.ok(res.success);
        done();
      })
    });
  });
```

钩子

> mocha提供钩`before()`，`after()`，`beforeEach()`，和`afterEach()`。这些应该用于设置前置条件并在测试后进行清理。

```
describe('hooks', function() {
  before(function() {
    // runs before all tests in this block
  });

  after(function() {
    // runs after all tests in this block
  });

  beforeEach(function() {
    // runs before each test in this block
  });

  afterEach(function() {
    // runs after each test in this block
  });

  // test cases
});
```

断言库：chai

https://github.com/chaijs/chai

> 断言库，所谓"断言"，就是判断源码的实际执行结果与预期结果是否一致，如果不一致就抛出一个错误

Assert

https://www.chaijs.com/api/assert/

Expect/Should

https://www.chaijs.com/api/bdd/

自动化测试：无头浏览器

https://zhuanlan.zhihu.com/p/76237595

中文文档： https://zhaoqize.github.io/puppeteer-api-zh_CN/#?product=Puppeteer&version=v8.0.0&show=api-pagewaitfornavigationoptions

Puppeteer 是什么?

> Puppeteer 是 Node.js 工具引擎， 用来模拟 Chrome 浏览器的运行。 Puppeteer 提供了一系列 API，通过 Chrome DevTools Protocol 协议控制 Chromium/Chrome 浏览器的行为

能做什么?

- 爬取 SPA 或 SSR 网站、网页截图或者生成 PDF
- **UI 自动化测试，模拟表单提交，键盘输入，点击等行为**
- 创建一个最新的自动化测试环境，使用最新的 js 和最新的 Chrome 浏览器运行测试用例
- 测试 Chrome 扩展程序 等等

实际工作

https://github.com/puppeteer/puppeteer/blob/v8.0.0/docs/api.md#table-of-contents

Browser： **浏览器级别，**可在浏览器上新建页面（newpage）等

```
browser = await puppeteer.launch(opts);
page = await browser.newPage();
```

Page: **浏览器页面解绑，** 页面上点击、跳转等

```
it('1.模拟已经登录', async function () {

    await page.setCookie({
      name: 'token',
      value: '9572d486975656e55006faf480821e28',
      url: '',
      domain: '.lechebang.com',
      path: '/',
      //expires: '2031-04-10T10:02:31.928Z',
    });
  });
it('2. 打开填写爱车信息页面', async function () {
    const title = '智能保养方案';
    // 等到加载完了页面才点击
    await page.waitForXPath('//*[@id="main"]/article/div/div[2]')
;
    const [response] = await Promise.all([
```

```
      page.waitForNavigation({}), // The promise resolves after navigation has finished
      page.click('.wi-main-flex2'), // Clicking the link will indirectly cause a navigation
    ]);
    const url = page.url();
    await page.goto(url, { waitUntil: 'networkidle2' });
    const pageTitle = await page.title();
    assert.equal(title, pageTitle);
    await page.screenshot({ path: `2.${pageTitle}.full.png`, fullPage: true });
  });
```

实战

写一个爬虫爬图片

```
it('6.实战下载图片', async function () {
    this.timeout(50000);
    const page = await browser.newPage();
    await page.goto('https://m.lechebang.com/webapp/car/brand', { timeout: 0 }).catch(e => {
      console.log('111error', e);
    });
    const res = await page.waitForResponse((response) => {
      return response.url().indexOf('gateway/car/getAllFirstLevelBrandType') > 0 && response.status() == 200;
    });
    // 下载图片
    const results = await res.json();
    let imgs = [];
    results.result.forEach(e => {
      imgs = imgs.concat(e.results.map(r => r.imgUrl));
    });

    console.log('111length', imgs.length);
    for (var i = 0, length = imgs.length; i < length; i++) {
      const url = imgs[i];
      var viewSource = await page.goto(url, { timeout: 0 }).catch
```

```
  (e => {
21          console.log('1error', url, e);
22        });
23        const buffer = await viewSource.buffer();
24        await fse.outputFile(`../images/${url.slice(-10)}`, buffer)
  ; //下载
25      }
26      assert.ok(res.ok());
27    });
```

持续集成&部署

云构建

自动化部署

githubhooks

> Git Hooks就是那些在Git执行特定事件（如commit、push、receive等）后触发运行的脚本。

https://github.com/kangaoxioashi/front/settings/hooks

jekins

> jekins 是

403问题： https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/Fix-No-Valid-Crumb-Error-Jenkins-GitHub-WebHook-Included
文档： https://www.jenkins.io/zh/doc/pipeline/tour/getting-started/

1. jekins 的安装

https://segmentfault.com/a/1190000023072976
启动： service jenkins start

2. jekins 的配置
   - 2.1： git hook 配置、插件安装等基础配置；
   - 2.2： 动态获取git 分支等信息，根据分支信息打包构建并发布；
   - 2.3: 邮箱配置 https://juejin.cn/post/6844904119707123719
3. 修改shell脚本实现自己的打包和发布逻辑

实战

http://60.205.211.38:8082/login?from=%2F

```
const program = require('commander');
const AdmZip = require('adm-zip');
const fse = require('fs-extra');

program.option('-b, --branch <string>', 'Build branch')
  .option('-m, --message <string>', 'Commit Message');
program.parse(process.argv);
const options = program.opts();
console.log('111112', options);

//1. 把代码打包 zip
console.log(process.cwd())
const zip = new AdmZip('');
zip.addLocalFolder(`${process.cwd()}/src/`);
zip.writeZip('./code.zip');
//2. 拷贝到发布目录
fse.copySync('./code.zip', '../code.zip');
//3. unzip
var unzip = new AdmZip('../code.zip');
unzip.extractAllTo("../code/", true);
```

总结

1. 所谓的自动化就是借助一些工具比如无头浏览器、Jenkins等将原本人工的工作用机器按流程去完成。
2.

https://segmentfault.com/a/1190000021898738