常用命令：

安装codeceptjs的puppeteer

npm install codeceptjs puppeteer --save-dev

安装codeceptjs的puppeteer同时忽略掉chromium的安装

npm install codeceptjs puppeteer --save-dev --ignore-scripts

创建项目

npx codeceptjs init

新建测试案例

npx codeceptjs gt

运行测试案例每一步的步骤的结果都显示出来

npx codeceptjs run --steps

查看更详细的输出

codeceptjs run --debug

查看非常详细的输出系统

codeceptjs run --verbose

如果您提供此类文件的相对路径，则可以执行单个测试文件：

codeceptjs run github\_test.js

# or

codeceptjs run admin/login\_test.js

按名称使用--grep参数筛选测试。这将执行名称与正则表达式匹配的所有测试。

codeceptjs run --grep "slow"

代码：

更改使用的浏览器：

helpers: {

Puppeteer: {

show:true,

chrome:{

executablePath:'C://Users//19868//AppData//Local//Google//Chrome//Application//chrome'

}

}

},

打开网页（接受相对或绝对网址）

amOnPage

找到按钮或链接并单击它

click

在字段内输入文本

fillField

与表单进行交互

selectOption，checkOption

 等待页面的某些部分完全呈现（对于测试SPA很重要

wait\*

从页面源获取值

grab\*

检查页面上的文本

see，dontSee

跳过当前测试

xScenario

仅执行当前测试

Scenario.only

检查页面上的元素

seeElement，dontSeeElement

可以通过pause()调用在测试的任何位置暂停测试执行。

按ENTER继续测试执行。

要查看所有可用命令，请按两次TAB以查看I中包含的所有操作的列表。

如果测试失败，您可以通过将pause()命令After()挂钩来阻止浏览器关闭。这对调试失败的测试非常有帮助。这样，您可以保持相同的会话，并在页面上尝试不同的操作，以了解出错的原因。

After(pause);

可以通过运行在测试上下文之外启动交互式shell

codeceptjs shell

打开网页，登录用户等常见准备步骤可以放入Before或Background挂钩：

Feature('CodeceptJS Demonstration');Before((I) => { // or Background

I.amOnPage('/documentation');

});Scenario('test some forms', (I) => {

I.click('Create User');

I.see('User is valid');

I.dontSeeInCurrentUrl('/documentation');

});Scenario('test title', (I) => {

I.seeInTitle('Example application');

});

如果您需要在所有测试之前运行复杂的设置并在之后拆除它，您可以使用BeforeSuite和AfterSuite 运行。BeforeSuite并且AfterSuite可以访问I对象，但是BeforeSuite/AfterSuite没有访问浏览器，因为此时它还没有运行。您可以使用它们来执行将设置环境的处理程序。BeforeSuite/AfterSuite将仅适用于声明它的文件（因此您可以声明文件的不同设置）

BeforeSuite((I) => {

I.syncDown('testfolder');

});AfterSuite((I) => {

I.syncUp('testfolder');

I.clearDir('testfolder');

});

要指定可以执行操作的页面上的确切区域，可以使用within函数。在其上下文中执行的所有内容都将缩小到locator指定的上下文：

用法： within('section', ()=>{})

例：

I.amOnPage('https://github.com');within('.js-signup-form', () => {

I.fillField('user[login]', 'User');

I.fillField('user[email]', 'user@user.com');

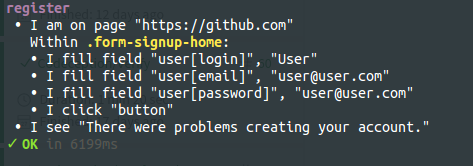
I.fillField('user[password]', 'user@user.com');

I.click('button');

});I.see('There were problems creating your account.');

within也可以使用[iframe](https://codecept.io/acceptance/" \l "iframes)

当在内部块中运行步骤时，将显示一个班次：



在within内可以返回一个可以在场景中使用的值：

// inside async functionconst val = await within('#sidebar', () => {

return I.grabTextFrom({ css: 'h1' });

});I.fillField('Description', val);

## **评论**

有一种简单的方法可以为测试场景添加其他注释。使用say命令将信息打印到屏幕：

I.say('I am going to publish post');I.say('I enter title and body');I.say('I expect post is visible on site');

使用第二个参数传入颜色值（ASCII）。

1. say('This is red', 'red'); //red is used
2. say('This is blue', 'blue'); //blue is used

I.say('This is by default'); //cyan is used