本文主要介绍如何对常见的puppeteer的API进行二次封装，方便进行快速的UI自动化case编写.

封装原则：

1-元素都是基于自定义的Element模型作为入参

2-常见操作的日志新增

3-提升操作稳定性

4-解决特定场景：批量操作，文件

NewPuppeteer

新建：src/utils/NewPuppeteer.js

进行常见的方法封装,其他的封装可以按需

1-页面操作

fullScreen:滚动全屏截图

```
    /**
     * 滚动全屏截图
     * 文件名称可以自定义
     * 路径：test/report/screenshot/fullPage.png
     */
    fullScreen:async function(page,picName='fullPage') {
        console.log('滚动全屏截图=[%s]',page.url());
        await page.evaluate(() => {
            return new Promise((resolve, reject) => {
                //滚动的总高度
                let sumHeight = 0;
                //每次滚动高度 500 px
                let eachScroll = 500;
                let timer = setInterval(() => {
                    //页面的高度 包含滚动高度
                    let scrollHeight = document.body.scrollHeight;
                    //滚动条向下滚动 distance
                    window.scrollBy(0, eachScroll);
                    sumHeight += eachScroll;
                    //当滚动的总高度 大于 页面高度 说明滚到底了。也就是说到滚动条滚到底时，以上还会继续累加，直到超过页面高度
                    if (sumHeight >= scrollHeight) {
                        clearInterval(timer);
                        resolve();
                    }
                }, 1000);
            })
        });
```

```
        await page.screenshot({
            path: './test/report/screenShot/'+picName+'.png',
            type: 'png',
            fullPage:true
        });

    }
```

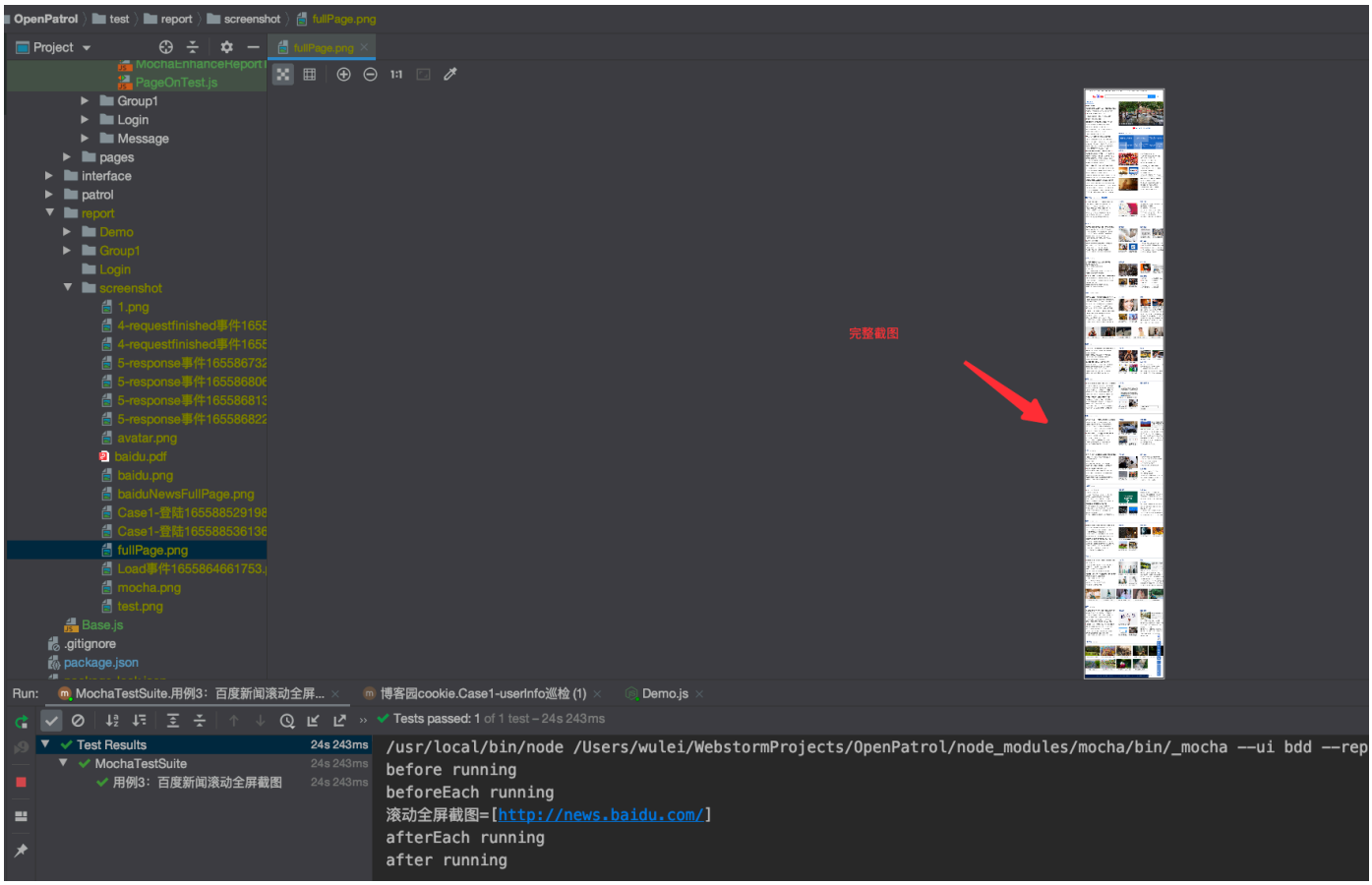
举个例子：百度新闻：http://news.baidu.com/，每次向下滚动后页面会懒加载，普通的截图只会截取固定大小。如果需要全屏截图，需要手动模拟滚动，然后全屏截图。

test/automation/cases/Demo/MochaDemo.js 新增

```
const newPuppeteer = require('../../../../src/utils/NewPuppeteer');


it('用例3：百度新闻滚动全屏截图', async function () {
    const browser = await puppeteer.launch({    //启动浏览器
        headless: false,   //代码运行时打开浏览器方便观察
    });
    const page = await browser.newPage();  //打开浏览器的一个tab 页
    await page.goto('http://news.baidu.com/');  //访问网址 http://news.baidu.com/
    //滚动全屏截图
    await newPuppeteer.fullScreen(page);
})
```

运行后：



说明：滚动全屏截图可以按需放到 test/Base.js的afterEach方法中

goto：页面跳转

```
/**
 * 页面跳转URL
 * @param url
 * @param timeout
 * @returns {Promise<void>}
 */
goto: async function(url, timeout = global.config.timeout) {
    console.log('打开页面=[%s]', url);
    await page.goto(url, {
        timeout: timeout
    });
    //导航配置
    await page.reload({
        timeout: timeout,
        waitUntil: ['load','domcontentloaded']
    });
}
```

page.reload():导航配置

waitUntil:满足什么条件认为页面跳转完成，默认是 load 事件触发时。指定事件数组，那么所有事件触发后才认为是跳转完成。事件包括：

- load - 页面的load事件触发时
- domcontentloaded - 页面的 DOMContentLoaded 事件触发时
- networkidle0 - 不再有网络连接时触发（至少500毫秒后）
- networkidle2 - 只有2个网络连接时触发（至少500毫秒后）

switchLatestPage:切换页面

```
/**
 * 切换最新页面
 * @returns {Promise.<*>}
 */
switchLatestPage: async function() {
    await page.waitForTimeout(1000);

    try {
        let targets = await browser.targets();
        const targetPages = await targets.filter(target => target.type() === 'page');
        global.page = await targetPages[targetPages.length - 1].page();
    }catch (e) {
        console.log('切换页面句柄错误'+e);
    }
}
```

pageContent：获取页面信息

```
/**
 * 返回页面content
 * @returns {Promise<string>}
 */
pageContent: async function () {
    let pageContentDetail = await page.content();
    return pageContentDetail; //这里需要return才能获取到页面内容
}
```

scrollY：滑动到Y

默认滑动到（0，y），也可以参考滚动截图的实现

```
/**
 * 向下滑动元素位置
 * @returns {Promise.<void>}
 */
scrollY: async function (Y) {
    await page.evaluate( pos => {
        return window.scrollTo(0, pos==undefined?window.innerHeight:pos);
    },Y);
}
```

page.evaluate参数传递：注意此时是执行JS函数，所以console输出是在页面

单参数：

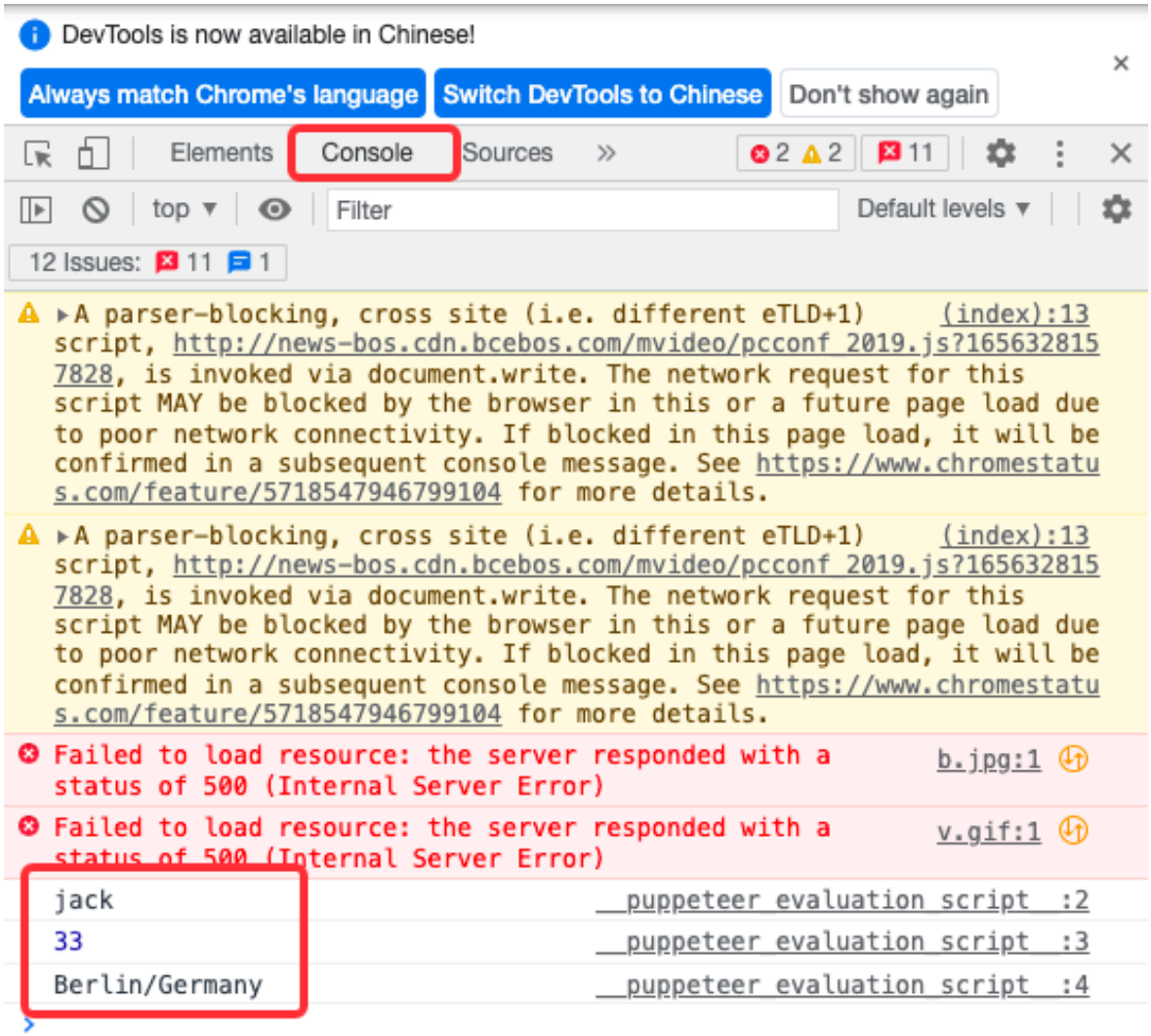
```
await page.evaluate( example => { code}, example );
```

两参数：

```
await page.evaluate( ( example_1, example_2 ) => {code }, example_1, example_2 );
```

多参数：

```
let name = 'jack';
let age  = 33;
let location = 'Berlin/Germany';
await page.evaluate(({name, age, location}) => {
    console.log(name);
    console.log(age);
    console.log(location);
},{name, age, location});
```



2-元素操作

click:点击元素

```
/**
 * 点击元素
 * @param element
 * @param timeout
 * @returns {Promise<void>}
 */
click: async function (element, timeout = global.config.timeout) {
    console.log('点击 [%s]', element.content);
    let selector = element.selector.split('>>');
    if (selector.length > 1){
        try {
            // 点击第一个
            await page.click(selector[0].selector);
        }catch (e) {
            console.log(e);
            throw new Error('元素 [' + element.selector + '] 不存在');
        }
    }else {
        await page.waitForSelector(element.selector, { timeout }).then(() =>
page.tap(element.selector));
    }
    await page.waitForTimeout(1000);
}
```

find:获取元素

```
/**
     * 获取元素
     * @returns {Promise.<ElementHandle>}
     */
    find: async function (element, timeout = global.config.timeout) {
        let el;
        if (element.selector.startsWith("/") || element.selector.startsWith("//")) {
            el = await page.waitForXPath(element.selector, { timeout });
        }else {
            el = await page.waitForSelector(element.selector, { timeout });
        }
        return el;
    }
```

getProp:获取元素属性

```
/**
 * 获取元素的目标值
 * @param element
 * @param targetValue
 * @returns {Promise.<*>}
 */
getProp: async function(element, targetValue) {
        let selector = element.selector.split('>>');
        if (selector.length > 1) {
            const value = await page.$$eval(selector[0], (anchors, textContent,
targetValue) => {
                return anchors.filter(anchor => {
                    return anchor.textContent == textContent;
                }).map(anchor => {
                    return anchor[targetValue];
                });
            }, selector[1], targetValue);

            if (value == undefined || value.length == 0) {
                console.log('未查找到指定属性' + targetValue);
                return value;
            } else{
                if (value.length > 1 && selector.length ==3) {
                    var tValue = value[selector[2] -1];
                    return (tValue == undefined || typeof tValue !== 'string' )
                        ? tValue
                        : tValue.replace('/\n/g', '').trim();
                }
                return (value[0] !== undefined || typeof value[0] == 'string' ) ?
value[0].replace('/\n/g', '').trim() : undefined;
            }
        } else {
            await page.waitForSelector(element.selector, {
                'timeout': global.config.timeout
            });
            let value = await page.$eval(element.selector, (el, targetValue) => {
                return el[targetValue];
            }, targetValue);

            if (value == undefined || value == '' || value == null) {
                console.log('未查找到指定属性' + targetValue);
                return value;
            }else {
                return (typeof value !== 'string') ? value : value.replace('/\n/g',
'').trim();
            }
        }
```

```
    }
```

基于获取元素属性的方法，可以新增：

getValue:获取value

```
/**
 * 获取元素的value
 * @param page
 * @param element
 * @param property
 * @returns {Promise.<Promise.<Object|undefined>|*>}
 */
getValue: async function (element) {
    console.log('获取元素 [%s] value', element.content);
    let i = 0;
    while (i < global.config.retry) {
        try {
            let content = await this.getProp(element, 'value');
            return content;
        } catch (e) {
            console.log('Find element [%s] error for [%d] time', element.selector, i+1)
            console.log(e);
            i++;
            continue;
        }
    }
},
```

getInnerText:获取innerText

```
/**
 * 获取元素的内容
 * @param page
 * @param element
 * @returns {Promise.<*>}
 */
getInnerText: async function (element) {
    console.log('获取元素 [%s] 内容', element.content);
    let i = 0;
    while (i < global.config.retry){
        try {
            let content = await this.getProp(element, 'innerText');
            if (content !== null && content !== undefined) {
                return content;
            }else {
                console.log('Find element [%s] error for [%d] time', element.selector,
i+1);
```

```
                i++;
                continue;
            }
        } catch (e) {
            console.log('Find element [%s] error for [%d] time', element.selector,
i+1);
            console.log(e);
            i++;
            continue;
        }
    }
}
```

getHref:获取href

```
/**
 * 获取元素的href信息
 * @param page
 * @param element
 * @returns {Promise.<Promise.<Object|undefined>|*>}
 */
getHref: async function (element) {
    console.log('获取元素 [%s] href', element.content);
    let i = 0;
    while (i < global.config.retry) {
        try {
            var href = await this.getProp(element, 'href');
            return href;
        } catch (e) {
            console.log('Find element [%s] error for [%d] time', element.selector,
i+1);
            console.log(e);
            i++;
            continue;
        }
    }
}
```

getSrc:获取src

```
/**
 * 获取元素的src信息
 * @param page
 * @param element
 * @returns {Promise.<Promise.<Object|undefined>|*>}
 */
getSrc: async function (element) {
    console.log('获取元素 [%s] src 链接', element.content);
```

```
    let i = 0;
    while (i < global.config.retry) {
        try {
            let content = await this.getProp(element, 'src');
            return content;
        } catch (e) {
            console.log('Find element [%s] error for [%d] time', element.selector, i+1)
            console.log(e);
            i++;
            continue;
        }
    }
}
```

getStyle:获取style

```
/**
 * 获取元素的style
 * @param element
 * @returns {Promise.<Promise.<Object|undefined>|*>}
 */
getStyle: async function (element) {
    console.log('获取元素 [%s] style', element.content);
    let i = 0;
    while (i < global.config.retry) {
        try {
            let content = await page.$eval(element.selector,
                (x) => {return
JSON.parse(JSON.stringify(window.getComputedStyle(x)))});
            return content;
        } catch (e) {
            console.log('Find element [%s] error for [%d] time', element.selector,
i+1);
            console.log(e);
            i++;
            continue;
        }
    }
}
```

isExist:是否存在

```
/**
 * 检查页面元素是否存在，存在返回true，不存在返回false
 * @param page
 * @param element
 * @returns {Promise.<boolean>}
 */
```

```
isExist: async function (element) {
    let isExist = false;
    try {
        console.log('查找元素 [%s]', element.content);
        let el = await this.getProp(element, 'outerHTML');
        if (el != undefined) {
            isExist = true;
            return isExist;
        }
    }catch (e) {
        console.log('页面元素 [%s] 不存在', element.selector)
        console.log(e);
        isExist = false;
        return isExist;
    }
}
```

isShow：是否展示

```
/**
 * 检查元素是否展示
 * @param element
 * @returns {Promise.<boolean>}
 */
isShow: async function (element) {
    let isShow = false;
    try {
        console.log('查找元素 [%s] 是否展示', element.content);
        let elStyle = await this.getProp(element, 'style');
        if (elStyle != undefined
            && elStyle.display !== 'none') {
            isShow = true;
            return isShow;
        }
    }catch (e) {
        console.log('页面元素 [%s] 未展示', element.selector)
        console.log(e);
        isShow = false;
        return isShow;
    }
}
```

clearInput：清空输入

```
/**
 * 清空输入框内容
 * @param element
 * @returns {Promise.<void>}
```

```
 */
clearInput: async function (element) {
    let textValue = await this.getValue(element);
    if(textValue !== null && textValue !== undefined && textValue.length>0) {
        // 光标聚焦到输入框
        await page.focus(element.selector);
        for (let i = 0; i < textValue.length; i++) {
            // 兼容光标定位在最左的情况
            await page.keyboard.press('ArrowRight');
            await page.keyboard.press('Backspace');
        }
    }
}
```

tap:点击元素

```
/**
 * tap元素
 * @param element
 * @param timeout
 * @returns {Promise.<void>}
 */
tap: async function (element, timeout = global.config.timeout) {
    console.log('轻触 [%s]', element.content);
    await page.waitForSelector(element.selector, { timeout }).then(() =>
page.tap(element.selector));
    await page.waitForTimeout(1000);
}
```

hover:悬停元素

```
/**
 * hover 悬停元素
 * @param element
 * @param timeout
 * @returns {Promise.<void>}
 */
hover: async function (element, timeout = global.config.timeout) {
    console.log('hover [%s]', element.content);
    let selector = element.selector.split('>>');

    if (selector.length > 1){
        await page.$$eval(selector[0], (anchors, text) => {
            anchors.map(anchor => {
                if (anchor.textContent == text) {
                    anchor.hover();
                    return;
                }
```

```
            })
        }, selector[1]);
    }else {
        await page.waitForSelector(element.selector, { timeout }).then(() =>
page.hover(element.selector));
    }
    await page.waitForTimeout(500);
}
```

search:输入并Enter

```
/**
 * 搜索文本，默认支持Enter键搜索
 * @param element
 * @param text
 * @returns {Promise.<void>}
 */
search: async function (element, text) {
    await this.type(element, text);
    await page.keyboard.press('Enter');
}
```

type:输入信息

```
/**
 * 输入信息
 * @param element
 * @param inputText
 * @param timeout
 * @returns {Promise<void>}
 */
type: async function (element, inputMsg, timeout = global.config.timeout) {
    console.log('[%s] 输入内容 [%s]', element.content, inputMsg);
    let selector = element.selector.split('>>');

    await this.clearInput(element);

    if (selector.length > 1){
        await page.$$eval(selector[0], (anchors, textContent, inputText) => {
            anchors.map(anchor => {
                if (anchor.placeholder == textContent) {
                    anchor.value = inputText;
                    var evt = document.createEvent("HTMLEvents");
                    evt.initEvent("change", false, true); // adding this created a
magic and passes it as if keypressed
                    anchor.dispatchEvent(evt);

                    return;
```

```
                }
            })
        }, selector[1], inputMsg);
    }else {
        await page.waitForSelector(element.selector, { timeout }).then(() =>
page.tap(element.selector)).then(() => page.keyboard.type(inputMsg));
    }
}
```

focusAndType:聚焦并输入

```
    /**
     * 输入框聚焦并输入内容
     * @param element 元素
     * @param inputText 输入内容
     * @returns {Promise.<void>}
     */
    focusAndType: async function(element, inputText) {
        console.log('[%s] 输入内容 [%s]', element.content, inputText);
        await page.focus(element.selector);
        await page.type(element.selector, inputText);
    }
```

3-文件操作

uploadFile:上传

```
/**
 * 文件上传
 * @param element 需要上传的input元素
 * @param filePath 以/开头的绝对路径，或者以./开头的相对工程根路径
 * @returns {Promise.<void>}
 */
uploadFile: async function (element, filePath, timeout = global.config.timeout) {
    if (filePath == undefined || filePath == null) {
        throw new Error('上传文件路径不能为空');
    }
    if (!(filePath.startsWith('/') || filePath.startsWith('./') ||
filePath.startsWith('../'))) {
        throw new Error(filePath +' 非绝对路径或相对路径，请检查');
    }
    console.log('元素 [%s] 上传文件 [%s]', element.content, filePath);
    await page.waitForSelector(element.selector,  { timeout }).then(x =>
x.uploadFile(filePath));
    await page.waitForTimeout(500);
}
```

下载操作本质是GET接口，忽略

4-经纬度

setCurrentGeoLocation:设置经纬度

```
/**
 * 设置当前的经纬度
 * @returns {Promise.<void>}
 */
setCurrentGeoLocation: async function() {
    await page.evaluateOnNewDocument(function() {
        navigator.geolocation.getCurrentPosition = function (cb) {
            setTimeout(() => {
                cb({
                    'coords': {
                        accuracy: 21,
                        altitude: null,
                        altitudeAccuracy: null,
                        heading: null,
                        latitude: 33.25924446,
                        longitude: 127.21937542,
                        speed: null
                    }
                })
            }, global.config.timeout)
        }
    });
}
```

5-完整代码

新建：src/utils/NewPuppeteer.js

```
module.exports = {
    //######### 页面操作 ############
    /**
     * 页面跳转URL
     * @param url
     * @param timeout
     * @returns {Promise<void>}
     */
    goto: async function(url, timeout = global.config.timeout) {
        console.log('打开页面=[%s]', url);
        await page.goto(url, {
            timeout: timeout
        });
```

```
        //导航配置
        await page.reload({
            timeout: timeout,
            waitUntil: ['load','domcontentloaded']
        });
    },
    /**
     * 切换最新页面
     * @returns {Promise.<*>}
     */
    switchLatestPage: async function() {
        await page.waitForTimeout(1000);

        try {
            let targets = await browser.targets();
            const targetPages = await targets.filter(target => target.type() ===
'page');
            global.page = await targetPages[targetPages.length - 1].page();
        }catch (e) {
            console.log('切换页面句柄错误'+e);
        }
    },
    /**
     * 关闭页面
     * @param page
     * @returns {Promise.<void>}
     */
    close: async function() {
        console.log('关闭页面 [%s]', page.url());
        await page.close();
    },
    /**
     * 返回页面content
     * @returns {Promise<string>}
     */
    pageContent: async function () {
        let pageContentDetail = await page.content();
        return pageContentDetail; //这里需要return才能获取到页面内容
    },
    /**
     * 向下滑动元素位置
     * @returns {Promise.<void>}
     */
    scrollY: async function (Y) {
        await page.evaluate( pos => {
            return window.scrollTo(0, pos==undefined?window.innerHeight:pos);
        },Y);
    },
    //######### 元素操作 ###########
```

```
    /**
     * 点击元素
     * @param element
     * @param timeout
     * @returns {Promise<void>}
     */
    click: async function (element, timeout = global.config.timeout) {
        console.log('点击元素 [%s]', element.content);
        let selector = element.selector.split('>>');
        if (selector.length > 1){
            try {
                // 点击第一个
                await page.click(selector[0].selector);
            }catch (e) {
                console.log(e);
                throw new Error('元素 [' + element.selector + '] 不存在');
            }
        }else {
            await page.waitForSelector(element.selector, { timeout }).then(() =>
page.tap(element.selector));
        }
    },
    /**
     * 获取元素
     * @returns {Promise.<ElementHandle>}
     */
    find: async function (element, timeout = global.config.timeout) {
        let el;
        if (element.selector.startsWith("/") || element.selector.startsWith("//")) {
            el = await page.waitForXPath(element.selector, { timeout });
        }else {
            el = await page.waitForSelector(element.selector, { timeout });
        }
        return el;
    },
    /**
     * 获取元素的目标值
     * @param element
     * @param targetValue
     * @returns {Promise.<*>}
     */
    getProp: async function(element, targetValue) {
        let selector = element.selector.split('>>');
        if (selector.length > 1) {
            const value = await page.$$eval(selector[0], (anchors, textContent,
targetValue) => {
                return anchors.filter(anchor => {
                    return anchor.textContent == textContent;
                }).map(anchor => {
```

```
                    return anchor[targetValue];
                });
            }, selector[1], targetValue);

            if (value == undefined || value.length == 0) {
                console.log('未查找到指定属性' + targetValue);
                return value;
            } else{
                if (value.length > 1 && selector.length ==3) {
                    var tValue = value[selector[2] -1];
                    return (tValue == undefined || typeof tValue !== 'string' )
                        ? tValue
                        : tValue.replace('/\n/g', '').trim();
                }
                return (value[0] !== undefined || typeof value[0] == 'string' ) ?
value[0].replace('/\n/g', '').trim() : undefined;
            }
        } else {
            await page.waitForSelector(element.selector, {
                'timeout': global.config.timeout
            });
            let value = await page.$eval(element.selector, (el, targetValue) => {
                return el[targetValue];
            }, targetValue);

            if (value == undefined || value == '' || value == null) {
                console.log('未查找到指定属性' + targetValue);
                return value;
            }else {
                return (typeof value !== 'string') ? value : value.replace('/\n/g',
'').trim();
            }
        }
    },
    /**
     * 获取元素的value
     * @param page
     * @param element
     * @param property
     * @returns {Promise.<Promise.<Object|undefined>|*>}
     */
    getValue: async function (element) {
        console.log('获取元素 [%s] value', element.content);
        let i = 0;
        while (i < global.config.retry) {
            try {
                let content = await this.getProp(element, 'value');
                return content;
            } catch (e) {
```

```
                console.log('Find element [%s] error for [%d] time', element.selector,
i+1)
                console.log(e);
                i++;
                continue;
            }
        }
    },

    /**
     * 获取元素的内容
     * @param page
     * @param element
     * @returns {Promise.<*>}
     */
    getInnerText: async function (element) {
        console.log('获取元素 [%s] 内容', element.content);
        let i = 0;
        while (i < global.config.retry){
            try {
                let content = await this.getProp(element, 'innerText');
                if (content !== null && content !== undefined) {
                    return content;
                }else {
                    console.log('Find element [%s] error for [%d] time',
element.selector, i+1);
                    i++;
                    continue;
                }
            } catch (e) {
                console.log('Find element [%s] error for [%d] time', element.selector,
i+1);
                console.log(e);
                i++;
                continue;
            }
        }
    },

    /**
     * 获取元素的href信息
     * @param page
     * @param element
     * @returns {Promise.<Promise.<Object|undefined>|*>}
     */
    getHref: async function (element) {
        console.log('获取元素 [%s] href', element.content);
        let i = 0;
        while (i < global.config.retry) {
```

```
            try {
                var href = await this.getProp(element, 'href');
                return href;
            } catch (e) {
                console.log('Find element [%s] error for [%d] time', element.selector,
i+1);
                console.log(e);
                i++;
                continue;
            }
        }
    },

    /**
     * 获取元素的src信息
     * @param page
     * @param element
     * @returns {Promise.<Promise.<Object|undefined>|*>}
     */
    getSrc: async function (element) {
        console.log('获取元素 [%s] src 链接', element.content);
        let i = 0;
        while (i < global.config.retry) {
            try {
                let content = await this.getProp(element, 'src');
                return content;
            } catch (e) {
                console.log('Find element [%s] error for [%d] time', element.selector,
i+1)
                console.log(e);
                i++;
                continue;
            }
        }
    },

    /**
     * 获取元素的style
     * @param element
     * @returns {Promise.<Promise.<Object|undefined>|*>}
     */
    getStyle: async function (element) {
        console.log('获取元素 [%s] style', element.content);
        let i = 0;
        while (i < global.config.retry) {
            try {
                let content = await page.$eval(element.selector,
                    (x) => {return
JSON.parse(JSON.stringify(window.getComputedStyle(x)))});
```

```
                return content;
            } catch (e) {
                console.log('Find element [%s] error for [%d] time', element.selector,
i+1);
                console.log(e);
                i++;
                continue;
            }
        }
    },

    /**
     * 检查页面元素是否存在，存在返回true，不存在返回false
     * @param page
     * @param element
     * @returns {Promise.<boolean>}
     */
    isExist: async function (element) {
        let isExist = false;
        try {
            console.log('查找元素 [%s]', element.content);
            let el = await this.getProp(element, 'outerHTML');
            if (el != undefined) {
                isExist = true;
                return isExist;
            }
        }catch (e) {
            console.log('页面元素 [%s] 不存在', element.selector)
            console.log(e);
            isExist = false;
            return isExist;
        }
    },

    /**
     * 检查元素是否展示
     * @param element
     * @returns {Promise.<boolean>}
     */
    isShow: async function (element) {
        let isShow = false;
        try {
            console.log('查找元素 [%s] 是否展示', element.content);
            let elStyle = await this.getProp(element, 'style');
            if (elStyle != undefined
                && elStyle.display !== 'none') {
                isShow = true;
                return isShow;
            }
```

```
        }catch (e) {
            console.log('页面元素 [%s] 未展示', element.selector)
            console.log(e);
            isShow = false;
            return isShow;
        }
    },
    /**
     * 清空输入框内容
     * @param element
     * @returns {Promise.<void>}
     */
    clearInput: async function (element) {
        let textValue = await this.getValue(element);
        if(textValue !== null && textValue !== undefined && textValue.length>0) {
            // 光标聚焦到输入框
            await page.focus(element.selector);
            for (let i = 0; i < textValue.length; i++) {
                // 兼容光标定位在最左的情况
                await page.keyboard.press('ArrowRight');
                await page.keyboard.press('Backspace');
            }
        }
    },
    /**
     * tap元素
     * @param element
     * @param timeout
     * @returns {Promise.<void>}
     */
    tap: async function (element, timeout = global.config.timeout) {
        console.log('轻触 [%s]', element.content);
        await page.waitForSelector(element.selector, { timeout }).then(() =>
page.tap(element.selector));
        await page.waitForTimeout(1000);
    },

    /**
     * hover 元素
     * @param element
     * @param timeout
     * @returns {Promise.<void>}
     */
    hover: async function (element, timeout = global.config.timeout) {
        console.log('hover [%s]', element.content);
        let selector = element.selector.split('>>');

        if (selector.length > 1){
            await page.$$eval(selector[0], (anchors, text) => {
```

```
                anchors.map(anchor => {
                    if (anchor.textContent == text) {
                        anchor.hover();
                        return;
                    }
                })
            }, selector[1]);
        }else {
            await page.waitForSelector(element.selector, { timeout }).then(() =>
page.hover(element.selector));
        }
        await page.waitForTimeout(500);
    },

    /**
     * 搜索文本，默认支持Enter键搜索
     * @param element
     * @param text
     * @returns {Promise.<void>}
     */
    search: async function (element, text) {
        await this.type(element, text);
        await page.keyboard.press('Enter');
    },

    /**
     * 输入信息
     * @param element
     * @param inputText
     * @param timeout
     * @returns {Promise<void>}
     */
    type: async function (element, inputMsg, timeout = global.config.timeout) {
        console.log('[%s] 输入内容 [%s]', element.content, inputMsg);
        let selector = element.selector.split('>>');

        await this.clearInput(element);

        if (selector.length > 1){
            await page.$$eval(selector[0], (anchors, textContent, inputText) => {
                anchors.map(anchor => {
                    if (anchor.placeholder == textContent) {
                        anchor.value = inputText;
                        var evt = document.createEvent("HTMLEvents");
                        evt.initEvent("change", false, true); // adding this created a
magic and passes it as if keypressed
                        anchor.dispatchEvent(evt);

                        return;
```

```
                    }
                })
            }, selector[1], inputMsg);
        }else {
            await page.waitForSelector(element.selector, { timeout }).then(() =>
page.tap(element.selector)).then(() => page.keyboard.type(inputMsg));
        }
    },

    /**
     * 输入框聚焦并输入内容
     * @param element 元素
     * @param inputText 输入内容
     * @returns {Promise.<void>}
     */
    focusAndType: async function(element, inputText) {
        console.log('[%s] 输入内容 [%s]', element.content, inputText);
        await page.focus(element.selector);
        await page.type(element.selector, inputText);
    },
    /**
     * 元素定位相同的输入框输入相同内容
     * @param element 获取所有满足selector的元素
     * @param text 输入相同的指定内容
     * @returns {Promise.<void>}
     */
    iteratorInputs: async function(element, text) {
        let elementHandles = await page.$$(element.selector);

        if (elementHandles !== null && elementHandles !== undefined &&
elementHandles.length > 0) {
            for (let i = 0; i < elementHandles.length; i++) {
                await elementHandles[i].asElement();

                // 如果有内容，先删除
                let elementValue = await elementHandles[i].getProperty('value');
                let context = elementValue._remoteObject.value;

                if (context !== undefined && context.length > 0) {

                    await elementHandles[i].focus();

                    for (let j = 0; j<context.length; j++) {
                        await elementHandles[i].press('ArrowRight');
                        await elementHandles[i].press('Backspace');
                    }
                }

                // 输入内容
```

```
                await elementHandles[i].type(text);

                await elementHandles[i].dispose();
            }
        }
    },

    //######### 文件操作 ###########
    /**
     * 文件上传
     * @param element 需要上传的input元素
     * @param filePath 以/开头的绝对路径，或者以./开头的相对工程根路径
     * @returns {Promise.<void>}
     */
    uploadFile: async function (element, filePath, timeout = global.config.timeout) {
        if (filePath == undefined || filePath == null) {
            throw new Error('上传文件路径不能为空');
        }
        if (!(filePath.startsWith('/') || filePath.startsWith('./') ||
filePath.startsWith('../'))) {
            throw new Error(filePath +' 非绝对路径或相对路径，请检查');
        }
        console.log('元素 [%s] 上传文件 [%s]', element.content, filePath);
        await page.waitForSelector(element.selector,  { timeout }).then(x =>
x.uploadFile(filePath));
        await page.waitForTimeout(500);
    },
    //######### cookie操作 ###########
    /**
     * 清空cookie
     * @returns {Promise.<void>}
     */
    clearCookie: async function () {
        global.cookies = undefined;
    },
    //######### 其他操作 ###########
    /**
     * 设置当前的经纬度
     * @returns {Promise.<void>}
     */
    setCurrentGeoLocation: async function() {
        await page.evaluateOnNewDocument(function() {
            navigator.geolocation.getCurrentPosition = function (cb) {
                setTimeout(() => {
                    cb({
                        'coords': {
                            accuracy: 21,
                            altitude: null,
                            altitudeAccuracy: null,
```

```
                            heading: null,
                            latitude: 33.25924446,
                            longitude: 127.21937542,
                            speed: null
                        }
                    })
                }, global.config.timeout)
            }
        });
    }
};
```

6-UI自动化

在PO模式的分层下，可以快速编写UI自动化的场景代码，此处省略具体的场景