Оглавление

1. Асинхронные методы

Основная часть методов в SpyBrowser JavaScript API работает асинхронно. Все асинхронные методы помечены суффиксами **Async**. Примеры асинхронных методов:

- spyBrowser.clipboard.getTextAsync
- spyBrowser.clipboard.setTextAsync
- spyBrowser.keyboard.sendTextAsync
- spyBrowser.session.getNameAsync
- spyBrowser.session.setNameAsync

и т.д.

Все асинхронные методы возвращают Promise (с результатом и без него).

2. Управление событиями

2.1. Виды событий

В текущей версии JavaScript API реализованы следующие события:

`"scriptLoadComplete"`	Вызывается один раз непосредственно после запуска скрипта пользователем. Во время навигации между страницами или обновления/перезагрузки страницы событие не вызывается.
`"mainWindowLoadComplete"`	Вызывается каждый раз после успешного завершения загрузки данных в главном фрейме страницы. Сюда относится как переход по другому адресу, так и перезагрузка/обновление текущего адреса.
`"mainWindowLoadError"`	Вызывается каждый раз после неудачного завершения загрузки данных в главном фрейме страницы. Событие "mainWindowLoadComplete" в этом случае не вызывается.
`"childWindowLoadComplete"`	Вызывается каждый раз после успешного завершения загрузки данных в главном фрейме страниц дочерних вкладок. Сюда относится как переход по другому адресу, так и перезагрузка/обновление текущего адреса.
`"childWindowLoadError"`	Вызывается каждый раз после неудачного завершения загрузки данных в главном фрейме страниц дочерних вкладок. Событие "childWindowLoadComplete" в этом случае не вызывается.
`"popupWindowLoadComplete"`	Вызывается каждый раз после успешного завершения загрузки данных в главном фрейме страниц всплывающих окон. Сюда относится как переход по другому адресу, так и перезагрузка/обновление текущего адреса.
`"popupWindowLoadError"`	Вызывается каждый раз после неудачного завершения загрузки данных в главном фрейме страниц всплывающих окон. Событие "popupWindowLoadComplete" в этом случае не вызывается.

2.2. spyBrowser.events.on

Метод осуществляет подписку на событие.

Сигнатура метода:

```
void spyBrowser.events.on(String name, Function callback)
```

Аргументы:

```
name – имя события (строка).
callback – функция обратного вызова, которая вызывается при наступлении события.
```

Возвращаемое значение:

```
(отсутствуют)
```

Пример использования:

```
// подписываемся на событие "scriptLoadComplete"
spyBrowser.events.on("scriptLoadComplete", async function () {

    // показываем текст в окне сообщения
    alert('Вызвано событие "scriptLoadComplete"');
});
```

2.3. spyBrowser.events.emit

Метод осуществляет вызов события.

Сигнатура метода:

```
void spyBrowser.events.emit(String name)
```

Аргументы:

```
name – имя события (строка).
```

Возвращаемое значение:

```
(отсутствуют)
```

Пример использования:

```
// подписываемся на событие "scriptLoadComplete"
spyBrowser.events.on("scriptLoadComplete", async function () {

    // вызываем событие "mainWindowLoadComplete"
    spyBrowser.events.emit("mainWindowLoadComplete");
});

// подписываемся на событие "mainWindowLoadComplete"
spyBrowser.events.on("mainWindowLoadComplete", async function () {

    // показываем текст в окне сообщения
    alert('Вызвано событие "scriptLoadComplete" или "mainWindowLoadComplete"');
});
```

3. Работа с буфером обмена Windows

3.1. spyBrowser.clipboard.getTextAsync

Метод получает текстовое содержимое из буфера обмена Windows.

Сигнатура метода:

```
Promise<String> spyBrowser.clipboard.getTextAsync()
```

Аргументы:

```
(отсутствуют)
```

Возвращаемое значение:

```
Promise с результатом String.
```

Пример использования:

```
spyBrowser.events.on("scriptLoadComplete", async function () {

    // сохраняем текстовое содержимое буфера обмена в переменную value
    var value = await spyBrowser.clipboard.getTextAsync();

    // показываем содержимое переменной value в окне сообщения
    alert(value);

    // завершаем работу скрипта
    spyBrowser.script.exit();
});
```

3.2. spyBrowser.clipboard.setTextAsync

Метод записывает текстовое содержимое в буфера обмена Windows.

Сигнатура метода:

```
Promise spyBrowser.clipboard.setTextAsync(String text)
```

Аргументы:

```
text – текст, который будет записан в буфер обмена.
```

Возвращаемое значение:

```
Promise
```

Пример использования:

```
spyBrowser.events.on("scriptLoadComplete", async function () {

    // задаём значение переменной value
    var value = "Clipboard test";

    // записываем значение переменной value в буфер обмена
    await spyBrowser.clipboard.setTextAsync(value);

    // завершаем работу скрипта
    spyBrowser.script.exit();
});
```

4. Работа с файлами

4.1. spyBrowser.io.assignFilesAsync

Метод сохраняет имена файлов во временный буфер. Эти имена файлов будут использоваться при перехвате окна выбора файлов.

Сигнатура метода:

```
Promise spyBrowser.io.assignFilesAsync(Array names)
```

Аргументы:

```
names – массив с именами файлов.
```

Возвращаемое значение:

```
Promise
```

Пример использования:

```
// перед запуском скрипта создаём в корне диска C папку "Test" и помещаем в эту папку два файла:
// 1.png
// 2.png
// страница, на которой запускается скрипт: https://www.freeconvert.com/png-converter
spyBrowser.events.on("scriptLoadComplete", async function () {

    try {

        // задаём значение переменной names
        // в нашем случае используется массив из двух имён файлов
        var names = ["C:\\Test\\1.png", "C:\\Test\\2.png"];

        // сохраняем имена файлов во временный буфер
        await spyBrowser.io.assignFilesAsync(names);

        // находим на странице элемент управления, отвечающий за выбор файлов
        var input = document.querySelector("input[name=file]");

        // кликаем по элементу управления (в результате чего будет вызвано и перехвачено диалоговое окно выбора файла)
        await spyBrowser.mouse.lButton.clickAsync(input);
    } catch (e) {
        // что-то пошло не по плану - отправляем информацию об исключении в консоль
        console.dir(e);
    }

    // завершаем работу скрипта
    spyBrowser.script.exit();
});
```

5. Эмуляция ввода текста с клавиатуры

5.1. spyBrowser.keyboard.sendEnterAsync

Метод эмулирует нажатие клавиши «Enter».

Сигнатура метода:

```
Promise spyBrowser.keyboard.sendEnterAsync()
```

Аргументы:

```
(отсутствуют)
```

Возвращаемое значение:

```
Promise
```

Пример использования:

```
// запускаем скрипт на одной из страниц: https://ya.ru/ или https://www.google.com/
spyBrowser.events.on("scriptLoadComplete", async function () {

    try {

        // пауза 500 мс
        await spyBrowser.script.sleepAsync(500);

        // находим элемент управления, который используется для ввода поискового запроса
        var input = document.querySelector("input[name=text]") || document.querySelector("input[name=q]");

        // эмулируем клик левой клавишей мыши по элементу управления (для того, чтобы передать ему фокус ввода)
        await spyBrowser.mouse.lButton.clickAsync(input);

        // пауза 500 мс
        await spyBrowser.script.sleepAsync(500);

        // эмулируем ввод с клавиатуры текста "Test"
        await spyBrowser.keyboard.sendTextAsync("Test");

        // заранее сообщаем о завершении работы скрипта
        spyBrowser.script.exit();

        // пауза 500 мс
        await spyBrowser.script.sleepAsync(500);

        // эмулируем нажатие клавиши "Enter"
        await spyBrowser.keyboard.sendEnterAsync();
    } catch (e) {

        // что-то пошло не по плану - отправляем информацию об исключении в консоль
        console.dir(e);
    }
});
```

5.2. spyBrowser.keyboard.sendTextAsync

Метод эмулирует ввод текста с клавиатуры.

Сигнатура метода:

```
Promise spyBrowser.keyboard.sendTextAsync(String text)
```

Аргументы:

```
text – текст, который будет использоваться во время эмуляции.
```

Возвращаемое значение:

```
Promise
```

Пример использования:

```
// запускаем скрипт на одной из страниц: https://ya.ru/ или https://www.google.com/
spyBrowser.events.on("scriptLoadComplete", async function () {

    try {
        // пауза 500 мс
        await spyBrowser.script.sleepAsync(500);

        // находим элемент управления, который используется для ввода поискового запроса
        var input = document.querySelector("input[name=text]") || document.querySelector("input[name=q]");

        // эмулируем клик левой клавишей мыши по элементу управления (для того, чтобы передать ему фокус ввода)
        await spyBrowser.mouse.lButton.clickAsync(input);

        // пауза 500 мс
        await spyBrowser.script.sleepAsync(500);

        // эмулируем ввод с клавиатуры текста "Test"
        await spyBrowser.keyboard.sendTextAsync("Test");

        // сообщаем о завершении работы скрипта
        spyBrowser.script.exit();
    } catch (e) {

        // что-то пошло не по плану - отправляем информацию об исключении в консоль
        console.dir(e);
    }
});
```

6. Эмуляция функций мыши

6.1. spyBrowser.mouse.lButton.clickAsync

Метод эмулирует одинарный клик левой клавишей мыши.

Сигнатура метода:

```
Promise spyBrowser.mouse.lButton.clickAsync(Int x, Int y)
Promise spyBrowser.mouse.lButton.clickAsync(HTMLElement element)
```

Аргументы:

```
x и y – координаты мыши, которые будут использоваться во время клика.
element – HTML-элемент, по которому будет совершён клик. Элемент должен находиться в видимой области экрана.
```

Возвращаемое значение:

```
Promise
```

Пример использования:

```
// запускаем скрипт на странице: https://cpstest.us/mouse-tester/
// для остановки скрипта обновляем страницу
spyBrowser.events.on("scriptLoadComplete", async function () {

    try {
        // находим элемент, по которому будут осуществляться клики клавишами мыши
        var element = document.querySelector("div.mouse-section");

        // выполняем цикл до тех пор, пока скрипт не будет завершён
        while (spyBrowser.script.isRunning) {

            // кликаем по элементу левой клавишей мыши
            await spyBrowser.mouse.lButton.clickAsync(element);

            // пауза 200 мс
            await spyBrowser.script.sleepAsync(200);

            // кликаем по элементу правой клавишей мыши
            await spyBrowser.mouse.rButton.clickAsync(element);

            // пауза 200 мс
            await spyBrowser.script.sleepAsync(200);
        }
    } catch(e) {
        // что-то пошло не по плану - отправляем информацию об исключении в консоль
        console.dir(e);
    }
});

spyBrowser.events.on("mainWindowLoadComplete", async function () {

    // сообщаем о завершении работы скрипта
    spyBrowser.script.exit();
});
```

6.2. spyBrowser.mouse.lButton.dblClickAsync

Метод эмулирует двойной клик левой клавишей мыши.

Сигнатура метода:

```
Promise spyBrowser.mouse.lButton.dblClickAsync(Int x, Int y)
Promise spyBrowser.mouse.lButton.dblClickAsync(HTMLElement element)
```

Аргументы:

```
x и y – координаты мыши, которые будут использоваться во время клика.
element – HTML-элемент, по которому будет совершён клик. Элемент должен находиться в видимой области экрана.
```

Возвращаемое значение:

```
Promise
```

Пример использования:

```
// запускаем скрипт на странице: https://cpstest.us/mouse-tester/
// для остановки скрипта обновляем страницу
spyBrowser.events.on("scriptLoadComplete", async function () {

    try {

        // находим элемент, по которому будут осуществляться клики клавишами мыши
        var element = document.querySelector("div.mouse-section");

        // выполняем цикл до тех пор, пока скрипт не будет завершён
        while (spyBrowser.script.isRunning) {

            // дважды кликаем по элементу левой клавишей мыши
            await spyBrowser.mouse.lButton.dblClickAsync(element);

            // пауза 200 мс
            await spyBrowser.script.sleepAsync(200);

            // дважды кликаем по элементу правой клавишей мыши
            await spyBrowser.mouse.rButton.dblClickAsync(element);

            // пауза 200 мс
            await spyBrowser.script.sleepAsync(200);
        }
    } catch (e) {

        // что-то пошло не по плану - отправляем информацию об исключении в консоль
        console.dir(e);
    }
});

spyBrowser.events.on("mainWindowLoadComplete", async function () {

    // сообщаем о завершении работы скрипта
    spyBrowser.script.exit();
});
```

6.3. spyBrowser.mouse.rButton.clickAsync

Метод эмулирует одинарный клик правой клавишей мыши.

Сигнатура метода:

```
Promise spyBrowser.mouse.rButton.clickAsync(Int x, Int y)
Promise spyBrowser.mouse.rButton.clickAsync(HTMLElement element)
```

Аргументы:

```
x и y – координаты мыши, которые будут использоваться во время клика.
element – HTML-элемент, по которому будет совершён клик. Элемент должен находиться в видимой области экрана.
```

Возвращаемое значение:

```
Promise
```

Пример использования:

```
// запускаем скрипт на странице: https://cpstest.us/mouse-tester/
// для остановки скрипта обновляем страницу
spyBrowser.events.on("scriptLoadComplete", async function () {

    try {

        // находим элемент, по которому будут осуществляться клики клавишами мыши
        var element = document.querySelector("div.mouse-section");

        // выполняем цикл до тех пор, пока скрипт не будет завершён
        while (spyBrowser.script.isRunning) {

            // кликаем по элементу левой клавишей мыши
            await spyBrowser.mouse.lButton.clickAsync(element);

            // пауза 200 мс
            await spyBrowser.script.sleepAsync(200);

            // кликаем по элементу правой клавишей мыши
            await spyBrowser.mouse.rButton.clickAsync(element);

            // пауза 200 мс
            await spyBrowser.script.sleepAsync(200);
        }
    } catch (e) {

        // что-то пошло не по плану - отправляем информацию об исключении в консоль
        console.dir(e);
    }
});

spyBrowser.events.on("mainWindowLoadComplete", async function () {

    // сообщаем о завершении работы скрипта
    spyBrowser.script.exit();
});
```

6.4. spyBrowser.mouse.rButton.dblClickAsync

Метод эмулирует двойной клик правой клавишей мыши.

Сигнатура метода:

```
Promise spyBrowser.mouse.rButton.dblClickAsync(Int x, Int y)
Promise spyBrowser.mouse.rButton.dblClickAsync(HTMLElement element)
```

Аргументы:

```
x и y – координаты мыши, которые будут использоваться во время клика.
element – HTML-элемент, по которому будет совершён клик. Элемент должен находиться в видимой области экрана.
```

Возвращаемое значение:

```
Promise
```

Пример использования:

```
// запускаем скрипт на странице: https://cpstest.us/mouse-tester/
// для остановки скрипта обновляем страницу
spyBrowser.events.on("scriptLoadComplete", async function () {

    try {

        // находим элемент, по которому будут осуществляться клики клавишами мыши
        var element = document.querySelector("div.mouse-section");

        // выполняем цикл до тех пор, пока скрипт не будет завершён
        while (spyBrowser.script.isRunning) {

            // дважды кликаем по элементу левой клавишей мыши
            await spyBrowser.mouse.lButton.dblClickAsync(element);

            // пауза 200 мс
            await spyBrowser.script.sleepAsync(200);

            // дважды кликаем по элементу правой клавишей мыши
            await spyBrowser.mouse.rButton.dblClickAsync(element);

            // пауза 200 мс
            await spyBrowser.script.sleepAsync(200);
        }
    } catch (e) {

        // что-то пошло не по плану - отправляем информацию об исключении в консоль
        console.dir(e);
    }
});

spyBrowser.events.on("mainWindowLoadComplete", async function () {

    // сообщаем о завершении работы скрипта
    spyBrowser.script.exit();
});
```

6.5. spyBrowser.mouse.wheelAsync

Метод эмулирует прокрутку экрана колёсиком мыши.

Сигнатура метода:

```
Promise spyBrowser.mouse.wheelAsync(Int x, Int y, Int deltaX, Int deltaY)
```

Аргументы:

```
x и y – координаты мыши, которые будут использоваться во время прокрутки (на странице может находиться несколько прокручиваемых областей).
deltaX и deltaY – количество прокручиваемых пикселей по горизонтали и по вертикали.
```

Возвращаемое значение:

```
Promise
```

Пример использования:

```
spyBrowser.events.on("scriptLoadComplete", async function () {

    // повторяем цикл до тех пор, пока скрипт не будет остановлен пользователем
    while (spyBrowser.script.isRunning) {

        // повторяем цикл 100 раз
        for (var i = 0; i < 100; i++) {

            // прокручиваем экран по вертикали вниз на 16 пикселей
            await spyBrowser.mouse.wheelAsync(0, 0, 0, -16);

            // пауза 5 мс
            await spyBrowser.script.sleepAsync(5);
        }

        // повторяем цикл 100 раз
        for (var i = 0; i < 100; i++) {

            // прокручиваем экран по вертикали вверх на 16 пикселей
            await spyBrowser.mouse.wheelAsync(0, 0, 0, 16);

            // пауза 5 мс
            await spyBrowser.script.sleepAsync(5);
        }
    }
});
```

6.6. spyBrowser.mouse.verticalHumanWheelAsync

Метод эмулирует вертикальную прокрутку экрана колёсиком мыши (на заданное количество делений колёсика) человеком. Для реализации метода использовались данные, которые были записаны с реальных устройств.

Сигнатура метода:

```
Promise spyBrowser.mouse.verticalHumanWheelAsync(Int x, Int y, Int ticks)
```

Аргументы:

```
x и y – координаты мыши, которые будут использоваться во время прокрутки (на странице может находиться несколько прокручиваемых областей).
ticks – количество делений колёсика мыши. Для прокрутки вниз указывается число в диапазоне -6...-1 (100-600 пикселей). Для прокрутки вверх указывается число в диапазоне 1...6 (100-600 пикселей). Значение 0 будет проигнорировано.
```

Возвращаемое значение:

```
Promise
```

Пример использования:

```
spyBrowser.events.on("scriptLoadComplete", async function () {

    // объявляем функцию, которая будет возвращать случайные числа из диапазона min...max
    function randomInteger(min, max) {
        var rnd = min + Math.random() * (max + 1 - min);
        return Math.floor(rnd);
    }

    // объявляем функцию, которая будет прокручивать страницу
    // при direction == -1 прокрутка будет осуществляться вниз
    // при direction == 1 прокрутка будет осуществляться вверх
    async function scroll(direction) {

        // запоминаем текущее положение прокрутки
        var lastPageYOffset = window.pageYOffset;

        // используем бесконечный цикл (остановим его в нужный момент)
        while (true) {

            // прокручиваем страницу на 4...6 делений колёсика мыши (400-600 пикселей)
            await spyBrowser.mouse.verticalHumanWheelAsync(150, 150, randomInteger(4, 6) * direction);

            // пауза в диапазоне 100...200 мс
            await spyBrowser.script.sleepAsync(randomInteger(100, 200));

            // если положение страницы не изменилось (скорее всего достигнута граница), то останавливаем цикл
            if (window.pageYOffset == lastPageYOffset)
                break;

            // запоминаем текущее положение прокрутки
            lastPageYOffset = window.pageYOffset;
        }
    }

    await scroll(-1);
    await scroll(1);

    // сообщаем о завершении работы скрипта
    spyBrowser.script.exit();
});
```

7. Работа с сетью

7.1. spyBrowser.net.fetchTextAsync

Метод осуществляет загрузку текста по указанному адресу.

Сигнатура метода:

```
Promise<String> spyBrowser.net.fetchTextAsync(String address)
```

Аргументы:

```
address – адрес.
```

Возвращаемое значение:

```
Promise с результатом в виде строки.
```

Пример использования:

```
spyBrowser.events.on("scriptLoadComplete", async function () {

    try {

        // загружаем страницу http://api.ipaddress.com/myip?format=json и сохраняем полученный текст в переменную text
        var text = await spyBrowser.net.fetchTextAsync("http://api.ipaddress.com/myip?format=json");

        // выводим содержимое переменной text в консоль
        console.log(text);
    } catch(e) {

        // что-то пошло не по плану - отправляем информацию об исключении в консоль
        console.dir(e);
    }

    // сообщаем о завершении работы скрипта
    spyBrowser.script.exit();
});
```

Метод spyBrowser.net.fetchTextAsync может использоваться так же для загрузки локального файла с текстом. Пример использования:

```
spyBrowser.events.on("scriptLoadComplete", async function () {

    try {

        // загружаем файл "C:\Test\Test.txt" и сохраняем полученный текст в переменную text
        var text = await spyBrowser.net.fetchTextAsync("file:///C:/Test/Test.txt");

        // выводим содержимое переменной text в консоль
        console.log(text);
    } catch(e) {

        // что-то пошло не по плану - отправляем информацию об исключении в консоль
        console.dir(e);
    }

    // сообщаем о завершении работы скрипта
    spyBrowser.script.exit();
});
```

8. Управление скриптом

8.1. spyBrowser.script.setInterval

Метод **spyBrowser.script.setInterval** является аналогом нативного метода **window.setInterval**. В отличие от нативного метода, таймер, созданный вызовом **spyBrowser.script.setInterval** будет остановлен в следующих случаях:

- при вызове **spyBrowser.script.clearAllIntervals**;
- при вызове **spyBrowser.script.exit**;
- при завершении работы скрипта пользователем.

Таймер можно остановить вызовом **window.clearInterval** или **window.clearTimeout**.

Сигнатура метода:

```
Int spyBrowser.script.setInterval(String code)
Int spyBrowser.script.setInterval(String code, Int timeout)
Int spyBrowser.script.setInterval(Function callback)
Int spyBrowser.script.setInterval(Function callback, Int timeout)
Int spyBrowser.script.setInterval(Function callback, Int timeout, Object arg0)
Int spyBrowser.script.setInterval(Function callback, Int timeout, Object arg0, Object arg1)
Int spyBrowser.script.setInterval(Function callback, Int timeout, Object arg0, Object arg1, /* … ,*/ Object argN)
```

Аргументы:

```
Для получения информации по аргументам метода обратитесь к официальной документации по нативному методу window.setInterval.
```

Возвращаемое значение:

```
Как и при вызове нативного метода window.setInterval метод spyBrowser.script.setInterval возвращает идентификатор таймера.
```

Пример использования:

```
spyBrowser.events.on("scriptLoadComplete", async function () {

    // инициализируем счётчик
    var counter = 0;

    // запускаем таймер с интервалом 100 мс
    spyBrowser.script.setInterval(function () {

        // увеличиваем значение счётчика на единицу
        counter++;

        // записываем содержимого счётчика в заголовок страницы
        document.title = counter;
    }, 100);

    // запускаем таймер с таймаутом 10000 мс (10 с)
    spyBrowser.script.setTimeout(function () {

        // сообщаем о завершении работы скрипта
        spyBrowser.script.exit();
    }, 10000);
});
```

8.2. spyBrowser.script.setTimeout

Метод **spyBrowser.script.setTimeout** является аналогом нативного метода **window. setTimeout**. В отличие от нативного метода, таймер, созданный вызовом **spyBrowser.script. setTimeout** будет остановлен в следующих случаях:

- при вызове **spyBrowser.script.clearAllIntervals**;
- при вызове **spyBrowser.script.exit**;
- при завершении работы скрипта пользователем.

Таймер можно остановить вызовом **window.clearInterval** или **window.clearTimeout**.

Сигнатура метода:

```
Int spyBrowser.script.setTimeout(String code)
Int spyBrowser.script.setTimeout(String code, Int timeout)
Int spyBrowser.script.setTimeout(Function callback)
Int spyBrowser.script.setTimeout(Function callback, Int timeout)
Int spyBrowser.script.setTimeout(Function callback, Int timeout, Object arg0)
Int spyBrowser.script.setTimeout(Function callback, Int timeout, Object arg0, Object arg1)
Int spyBrowser.script.setTimeout(Function callback, Int timeout, Object arg0, Object arg1, /* … ,*/ Object argN)
```

Аргументы:

```
Для получения информации по аргументам метода обратитесь к официальной документации по нативному методу window.setTimeout.
```

Возвращаемое значение:

```
Как и при вызове нативного метода window.setTimeout метод spyBrowser.script.setTimeout возвращает идентификатор таймера.
```

Пример использования:

```
spyBrowser.events.on("scriptLoadComplete", async function () {

    // инициализируем счётчик
    var counter = 0;

    // запускаем таймер с интервалом 100 мс
    spyBrowser.script.setInterval(function () {

        // увеличиваем значение счётчика на единицу
        counter++;

        // записываем содержимого счётчика в заголовок страницы
        document.title = counter;
    }, 100);

    // запускаем таймер с таймаутом 10000 мс (10 с)
    spyBrowser.script.setTimeout(function () {

        // сообщаем о завершении работы скрипта
        spyBrowser.script.exit();
    }, 10000);
});
```

8.3. spyBrowser.script.clearAllIntervals

При вызове метода осуществляется остановка всех таймеров, которые были запущены с помощью **spyBrowser.script.setInterval** и **spyBrowser.script.setTimeout**.

Сигнатура метода:

```
void spyBrowser.script.clearAllIntervals()
```

Аргументы:

```
(отсутствуют)
```

Возвращаемое значение:

```
(отсутствует)
```

Пример использования:

```
spyBrowser.events.on("scriptLoadComplete", async function () {

    // инициализируем счётчик
    var counter = 0;

    // запускаем таймер с интервалом 100 мс
    spyBrowser.script.setInterval(function () {

        // увеличиваем значение счётчика на единицу
        counter++;

        // записываем содержимого счётчика в заголовок страницы
        document.title = counter;
    }, 100);

    // запускаем таймер с таймаутом 10000 мс (10 с)
    spyBrowser.script.setTimeout(function () {

        // останавливаем все таймеры
        spyBrowser.script.clearAllIntervals();
    }, 10000);
});
```

8.4. spyBrowser.script.sleepAsync

Метод используется для организации задержек в различных участках кода.

Сигнатура метода:

```
Promise spyBrowser.script.sleepAsync(Int timeout)
```

Аргументы:

```
timeout – таймаут в миллисекундах.
```

Возвращаемое значение:

```
Promise
```

Пример использования:

```
spyBrowser.events.on("scriptLoadComplete", async function () {

    // инициализируем счётчик
    var counter = 0;

    // повторяем цикл до тех пор, пока он не будет остановлен пользователем
    while (spyBrowser.script.isRunning) {

        // увеличиваем значение счётчика на единицу
        counter++;

        // записываем содержимого счётчика в заголовок страницы
        document.title = counter;

        // пауза 200 мс
        await spyBrowser.script.sleepAsync(200);
    }
});
```

8.5. spyBrowser.script.exit

Метод отправляет сообщение о том, что работу скрипта следует завершить. При этом выполнение кода, который следует за вызовом, будет продолжено. Все таймеры, созданные с помощью вызовов **spyBrowser.script.setInterval** и **spyBrowser.script.setTimeout** будут остановлены. Свойство **spyBrowser.script.isRunning** получит значение **false**. Все подписки на события с помощью **spyBrowser.events.on** будут удалены.

Сигнатура метода:

```
void spyBrowser.script.exit()
```

Аргументы:

```
(отсутствуют)
```

Возвращаемое значение:

```
(отсутствует)
```

Пример использования:

```
spyBrowser.events.on("scriptLoadComplete", async function () {

    // инициализируем счётчик
    var counter = 0;

    // запускаем таймер с интервалом 100 мс
    spyBrowser.script.setInterval(function () {

        // увеличиваем значение счётчика на единицу
        counter++;

        // записываем содержимого счётчика в заголовок страницы
        document.title = counter;
    }, 100);

    // запускаем таймер с таймаутом 10000 мс (10 с)
    spyBrowser.script.setTimeout(function () {

        // сообщаем о завершении работы скрипта
        spyBrowser.script.exit();
    }, 10000);
});
```

8.6. spyBrowser.script.isRunning

Сразу же после запуска скрипта свойство **spyBrowser.script.isRunning** получает значение **true**. Значение **false** будет получено в одном из случаев:

- при вызове **spyBrowser.script.exit**;
- при завершении работы скрипта пользователем.

Возвращаемое значение:
`false или true.`

Пример использования:

```
spyBrowser.events.on("scriptLoadComplete", async function () {

    // инициализируем счётчик
    var counter = 0;

    // повторяем цикл до тех пор, пока он не будет остановлен пользователем
    while (spyBrowser.script.isRunning) {

        // увеличиваем значение счётчика на единицу
        counter++;

        // записываем содержимого счётчика в заголовок страницы
        document.title = counter;

        // пауза 200 мс
        await spyBrowser.script.sleepAsync(200);
    }
});
```

9. Взаимодействие с сессией

9.1. spyBrowser.session.getIdAsync

Метод получает идентификатор текущей сессии.

Сигнатура метода:

```
Promise<Int> spyBrowser.session.getIdAsync()
```

Аргументы:

```
(отсутствуют)
```

Возвращаемое значение:

```
Promise с результатом, который содержит идентификатор текущей сессии.
```

Пример использования:

```
spyBrowser.events.on("scriptLoadComplete", async function () {

    // получаем идентификатор текущей сессии и сохраняем его в переменную id
    var id = await spyBrowser.session.getIdAsync();

    // отображаем содержимое переменной id в окне сообщения
    alert(id);

    // сообщаем о завершении работы скрипта
    spyBrowser.script.exit();
});
```

9.2. spyBrowser.session.getNameAsync

Метод получает имя текущей сессии.

Сигнатура метода:

```
Promise<String> spyBrowser.session.getNameAsync()
```

Аргументы:

```
(отсутствуют)
```

Возвращаемое значение:

```
Promise с результатом, который содержит имя текущей сессии.
```

Пример использования:

```
spyBrowser.events.on("scriptLoadComplete", async function () {

    // получаем имя текущей сессии и сохраняем его в переменную name
    var name = await spyBrowser.session.getNameAsync();

    // отображаем содержимое переменной name в окне сообщения
    alert(name);

    // сообщаем о завершении работы скрипта
    spyBrowser.script.exit();
});
```

9.3. spyBrowser.session.setNameAsync

Метод устанавливает имя текущей сессии.

Сигнатура метода:

```
Promise spyBrowser.session.setNameAsync(String value)
```

Аргументы:

```
value – новое имя сессии.
```

Возвращаемое значение:

```
Promise
```

Пример использования:

```
spyBrowser.events.on("scriptLoadComplete", async function () {

    // меняем имя сессии
    await spyBrowser.session.setNameAsync("Новое имя сессии");

    // сообщаем о завершении работы скрипта
    spyBrowser.script.exit();
});
```

9.4. spyBrowser.session.getGroupAsync

Метод получает группу текущей сессии.

Сигнатура метода:

```
Promise<String> spyBrowser.session.getGroupAsync()
```

Аргументы:

```
(отсутствуют)
```

Возвращаемое значение:

```
Promise с результатом, который содержит группу текущей сессии.
```

Пример использования:

```
spyBrowser.events.on("scriptLoadComplete", async function () {

    // получаем группу текущей сессии и сохраняем его в переменную group
    var group = await spyBrowser.session.getGroupAsync();

    // отображаем содержимое переменной group в окне сообщения
    alert(group);

    // сообщаем о завершении работы скрипта
    spyBrowser.script.exit();
});
```

9.5. spyBrowser.session.setGroupAsync

Метод устанавливает группу текущей сессии.

Сигнатура метода:

```
Promise spyBrowser.session.setGroupAsync(String value)
```

Аргументы:

```
value – новая группа сессии.
```

Возвращаемое значение:

```
Promise
```

Пример использования:

```
spyBrowser.events.on("scriptLoadComplete", async function () {

    // меняем группу сессии
    await spyBrowser.session.setGroupAsync("Новая группа сессии");

    // сообщаем о завершении работы скрипта
    spyBrowser.script.exit();
});
```

9.6. spyBrowser.session.getCommentsAsync

Метод получает комментарии к текущей сессии.

Сигнатура метода:

```
Promise<String> spyBrowser.session.getCommentsAsync()
```

Аргументы:

```
(отсутствуют)
```

Возвращаемое значение:

```
Promise с результатом, который содержит комментарии к текущей сессии.
```

Пример использования:

```
spyBrowser.events.on("scriptLoadComplete", async function () {

    // получаем комментарии к текущей сессии и сохраняем его в переменную comments
    var comments = await spyBrowser.session.getCommentsAsync();

    // отображаем содержимое переменной comments в окне сообщения
    alert(comments);

    // сообщаем о завершении работы скрипта
    spyBrowser.script.exit();
});
```

9.7. spyBrowser.session.setCommentsAsync

Метод устанавливает комментарии к текущей сессии.

Сигнатура метода:

```
Promise spyBrowser.session.setCommentsAsync(String value)
```

Аргументы:

```
value – новые комментарии к сессии.
```

Возвращаемое значение:

```
Promise
```

Пример использования:

```
spyBrowser.events.on("scriptLoadComplete", async function () {

    // меняем комментарии к сессии
    await spyBrowser.session.setCommentsAsync("Новые комментарии к сессии");

    // сообщаем о завершении работы скрипта
    spyBrowser.script.exit();
});
```

9.8. spyBrowser.session.enumMarkersAsync

Метод возвращает массив с именами маркеров, которые зарегистрированы в программе.

Сигнатура метода:

```
Promise<Array> spyBrowser.session.enumMarkersAsync()
```

Аргументы:

```
(отсутствуют)
```

Возвращаемое значение:

```
Promise с результатом, который содержит массив с именами маркеров.
```

Пример использования:

```
spyBrowser.events.on("scriptLoadComplete", async function () {

    try {

        // получаем массив с зарегистрированными именами маркеров
        var array = await spyBrowser.session.enumMarkersAsync();

        // повторяем цикл до тех пор, пока скрипт не будет завершён пользователем
        while (spyBrowser.script.isRunning) {

            // извлекаем из массива array первый элемент и помещаем его в переменную marker
            var marker = array.shift();

            // добавляем значение переменной marker в конец массива array
            array.push(marker);

            // меняем маркер текущей сессии
            await spyBrowser.session.setMarkerAsync(marker);

            // пауза 200 мс
            await spyBrowser.script.sleepAsync(200);
        }
    } catch (e) {

        // что-то пошло не по плану - отправляем информацию об исключении в консоль
        console.dir(e);
    }
});
```

9.9. spyBrowser.session.getMarkerAsync

Метод получает имя маркера текущей сессии.

Сигнатура метода:

```
Promise<String> spyBrowser.session.getMarkerAsync()
```

Аргументы:

```
(отсутствуют)
```

Возвращаемое значение:

```
Promise с результатом, который содержит имя маркера текущей сессии.
```

Пример использования:

```
spyBrowser.events.on("scriptLoadComplete", async function () {

    // получаем имя маркера текущей сессии и сохраняем его в переменную marker
    var marker = await spyBrowser.session.getMarkerAsync();

    // отображаем содержимое переменной marker в окне сообщения
    alert(marker);

    // сообщаем о завершении работы скрипта
    spyBrowser.script.exit();
});
```

9.10. spyBrowser.session.setMarkerAsync

Метод устанавливает маркер текущей сессии.

Сигнатура метода:

```
Promise spyBrowser.session.setMarkerAsync(String value)
```

Аргументы:

```
value – имя нового маркера.
```

Возвращаемое значение:

```
Promise
```

Пример использования:

```
spyBrowser.events.on("scriptLoadComplete", async function () {

    try {

        // получаем массив с зарегистрированными именами маркеров
        var array = await spyBrowser.session.enumMarkersAsync();

        // повторяем цикл до тех пор, пока скрипт не будет завершён пользователем
        while (spyBrowser.script.isRunning) {

            // извлекаем из массива array первый элемент и помещаем его в переменную marker
            var marker = array.shift();

            // добавляем значение переменной marker в конец массива array
            array.push(marker);

            // меняем маркер текущей сессии
            await spyBrowser.session.setMarkerAsync(marker);

            // пауза 200 мс
            await spyBrowser.script.sleepAsync(200);
        }
    } catch (e) {

        // что-то пошло не по плану - отправляем информацию об исключении в консоль
        console.dir(e);
    }
});
```

10. Хранение данных

В текущей версии программы используется два типа хранилища: глобальное и локальное.

	Глобальное хранилище	Локальное хранилище
Количество хранилищ	1	Равно количеству сессий
Доступ к хранилищу	Из любой сессии	Из сессии-родителя
Место хранения	Файл «storage.json» во временной папке **браузера**.	Файл «storage.json» во временной папке **сессии**.
Тип сохраняемых данных	Объект, массив, строка, число или boolean	Объект, массив, строка, число или boolean

10.1. Работа с глобальным хранилищем

10.1.1. spyBrowser.storage.global.getValueAsync

Метод осуществляет получение данных из глобального хранилища.

Сигнатура метода:

```
Promise<Object> spyBrowser.storage.global.getValueAsync(String key)
```

Аргументы:

```
key – ключ, по которому осуществляется получение данных из хранилища.
```

Возвращаемое значение:

```
Promise с результатом в виде объекта, массива, строки, числа или boolean. Если в качестве результата пришло null, значит данных с указанным ключом не существует.
```

Пример использования:

```
spyBrowser.events.on("scriptLoadComplete", async function () {

    try {

        // сохраняем в переменную key уникальный ключ
        // (используем уникальный для того, чтобы другой скрипт не затёр наши данные)
        var key = "581f5ff6027d102e";

        // создаём тестовый объект
        var value1 = {
            string1: "строка 1",
            string2: "строка 2",
            boolean1: false,
            boolean2: true,
            int1: 1,
            int2: 2,
            array1: ["один", "два", "три"],
            array2: ["три", "два", "один"]
        };

        // сохраняем данные в хранилище
        await spyBrowser.storage.global.setValueAsync(key, value1);

        // получаем данные из хранилища
        var value2 = await spyBrowser.storage.global.getValueAsync(key);

        // выводим результат в консоль
        console.dir(value2);

        // удаляем данные из хранилища
        await spyBrowser.storage.global.setValueAsync(key, null);

        // сообщаем о завершении работы скрипта
        spyBrowser.script.exit();
    } catch (e) {

        // что-то пошло не по плану - отправляем информацию об исключении в консоль
        console.dir(e);
    }
});
```

10.1.2. spyBrowser.storage.global.setValueAsync

Метод осуществляет запись данных в глобальное хранилище.

Сигнатура метода:

```
Promise spyBrowser.storage.global.setValueAsync(String key, Object value)
```

Аргументы:

```
key – ключ, по которому осуществляется запись в хранилище.
value – данные (объект, массив, строка, число или boolean). Если в качестве данных передать null, то запись с указанным ключом будет удалена из хранилища.
```

Возвращаемое значение:

```
Promise
```

Пример использования:

```
spyBrowser.events.on("scriptLoadComplete", async function () {

    try {

        // сохраняем в переменную key уникальный ключ
        // (используем уникальный для того, чтобы другой скрипт не затёр наши данные)
        var key = "581f5ff6027d102e";

        // создаём тестовый объект
        var value1 = {
            string1: "строка 1",
            string2: "строка 2",
            boolean1: false,
            boolean2: true,
            int1: 1,
            int2: 2,
            array1: ["один", "два", "три"],
            array2: ["три", "два", "один"]
        };

        // сохраняем данные в хранилище
        await spyBrowser.storage.global.setValueAsync(key, value1);

        // получаем данные из хранилища
        var value2 = await spyBrowser.storage.global.getValueAsync(key);

        // выводим результат в консоль
        console.dir(value2);

        // удаляем данные из хранилища
        await spyBrowser.storage.global.setValueAsync(key, null);

        // сообщаем о завершении работы скрипта
        spyBrowser.script.exit();
    } catch (e) {

        // что-то пошло не по плану - отправляем информацию об исключении в консоль
        console.dir(e);
    }
});
```

10.2. Работа с локальным хранилищем

10.2.1. spyBrowser.storage.local.getValueAsync

Метод осуществляет получение данных из локального хранилища.

Сигнатура метода:

```
Promise<Object> spyBrowser.storage.local.getValueAsync(String key)
```

Аргументы:

```
key – ключ, по которому осуществляется получение данных из хранилища.
```

Возвращаемое значение:

```
Promise с результатом в виде объекта, массива, строки, числа или boolean. Если в качестве результата пришло null, значит данных с указанным ключом не существует.
```

Пример использования:

```
spyBrowser.events.on("scriptLoadComplete", async function () {

    try {

        // сохраняем в переменную key уникальный ключ
        // (используем уникальный для того, чтобы другой скрипт не затёр наши данные)
        var key = "581f5ff6027d102e";

        // создаём тестовый объект
        var value1 = {
            string1: "строка 1",
            string2: "строка 2",
            boolean1: false,
            boolean2: true,
            int1: 1,
            int2: 2,
            array1: ["один", "два", "три"],
            array2: ["три", "два", "один"]
        };

        // сохраняем данные в хранилище
        await spyBrowser.storage.local.setValueAsync(key, value1);

        // получаем данные из хранилища
        var value2 = await spyBrowser.storage.local.getValueAsync(key);

        // выводим результат в консоль
        console.dir(value2);

        // удаляем данные из хранилища
        await spyBrowser.storage.local.setValueAsync(key, null);

        // сообщаем о завершении работы скрипта
        spyBrowser.script.exit();
    } catch (e) {

        // что-то пошло не по плану - отправляем информацию об исключении в консоль
        console.dir(e);
    }
});
```

10.2.2. spyBrowser.storage.local.setValueAsync

Метод осуществляет запись данных в локальное хранилище.

Сигнатура метода:

```
Promise spyBrowser.storage.local.setValueAsync(String key, Object value)
```

Аргументы:

```
key – ключ, по которому осуществляется запись в хранилище.
value – данные (объект, массив, строка, число или boolean). Если в качестве данных передать null, то запись с указанным ключом будет удалена из хранилища.
```

Возвращаемое значение:

```
Promise
```

Пример использования:

```
spyBrowser.events.on("scriptLoadComplete", async function () {

    try {

        // сохраняем в переменную key уникальный ключ
        // (используем уникальный для того, чтобы другой скрипт не затёр наши данные)
        var key = "581f5ff6027d102e";

        // создаём тестовый объект
        var value1 = {
            string1: "строка 1",
            string2: "строка 2",
            boolean1: false,
            boolean2: true,
            int1: 1,
            int2: 2,
            array1: ["один", "два", "три"],
            array2: ["три", "два", "один"]
        };

        // сохраняем данные в хранилище
        await spyBrowser.storage.local.setValueAsync(key, value1);

        // получаем данные из хранилища
        var value2 = await spyBrowser.storage.local.getValueAsync(key);

        // выводим результат в консоль
        console.dir(value2);

        // удаляем данные из хранилища
        await spyBrowser.storage.local.setValueAsync(key, null);

        // сообщаем о завершении работы скрипта
        spyBrowser.script.exit();
    } catch (e) {

        // что-то пошло не по плану - отправляем информацию об исключении в консоль
        console.dir(e);
    }
});
```

11. Примеры скриптов

11.1. Пример скрипта, который открывает адреса по списку (Вариант №1)

В этом примере список адресов помещается в комментарий к сессии, каждый адрес с новой строки. Адрес должен начинаться на http:// или на https://

```
spyBrowser.events.on("scriptLoadComplete", async function () {

    // объявляем функцию, которая получает комментарии к сессии и преобразует их в массив адресов
    async function getArray() {

        // получаем комментарии к сессии
        var comments = await spyBrowser.session.getCommentsAsync();

        // разбиваем полученные комментарии на массив строк
        var array = comments.split(/\r|\n/);

        // удаляем из массива пустые строки
        array = array.filter(function (line) {
            return line.trim() != "";
        });

        // возвращаем массив адресов
        return array;
    }

    // объявляем функцию, которая преобразует массив адресов в текст и заменяет полученным текстом комментарии к сессии
    async function setArray(array) {

        // преобразуем массив в текст
        var comments = array.join("\r\n");

        // устанавливаем комментарии к сессии
        await spyBrowser.session.setCommentsAsync(comments);
    }

    // получаем массив с адресами
    var array = await getArray();

    // если массив пустой, то сообщаем о завершении работы скрипта
    if (array.length == 0) {

        spyBrowser.script.exit();
    } else {

        // получаем первый адрес из массива адресов
```

```
        var address = array.shift();

        // сохраняем массив с адресами в комментарии к сессии
        await setArray(array);

        // удаляем из адреса пробельные символы
        var address = address.trim();

        // открываем адрес address в текущей вкладке
        spyBrowser.window.open(address, false, false);
    }
});

spyBrowser.events.on("mainWindowLoadComplete", async function () {

    // пауза 3 секунды
    await spyBrowser.script.sleepAsync(3000);

    // вызываем событие "scriptLoadComplete"
    spyBrowser.events.emit("scriptLoadComplete");
});

spyBrowser.events.on("mainWindowLoadError", async function () {

    // пауза 3 секунды
    await spyBrowser.script.sleepAsync(3000);

    // вызываем событие "scriptLoadComplete"
    spyBrowser.events.emit("scriptLoadComplete");
});
```