全局变量与函数

全局变量和函数在所有模块中均可使用。 但以下变量的作用域只在模块内，详见 module：

- exports
- module
- require() 以下的对象是特定于 Auto.js 的。 有些内置对象是 JavaScript 语言本身的一部分，它们也是全局的。

一些模块中的函数为了使用方便也可以直接全局使用，这些函数在此不再赘述。例如timers模块的setInterval, setTimeout等函数。

sleep(n)

- n {number} 毫秒数

暂停运行n**毫秒**的时间。1秒等于1000毫秒。

```js
//暂停5秒
sleep(5000);
```

currentPackage()

- 返回 {string}

返回最近一次监测到的正在运行的应用的包名，一般可以认为就是当前正在运行的应用的包名。

此函数依赖于无障碍服务，如果服务未启动，则抛出异常并提示用户启动。

currentActivity()

- 返回 {string}

返回最近一次监测到的正在运行的Activity的名称，一般可以认为就是当前正在运行的Activity的名称。

此函数依赖于无障碍服务，如果服务未启动，则抛出异常并提示用户启动。

setClip(text)

- `text` {string} 文本

设置剪贴板内容。此剪贴板即系统剪贴板，在一般应用的输入框中"粘贴"既可使用。

```js
setClip("剪贴板文本");
```

getClip()

- 返回 {string}

返回系统剪贴板的内容。

```js
toast("剪贴板内容为:" + getClip());
```

toast(message)

- message {string} 要显示的信息

以气泡显示信息message几秒。(具体时间取决于安卓系统，一般都是2秒)

注意，信息的显示是"异步"执行的，并且，不会等待信息消失程序才继续执行。如果在循环中执行该命令，可能出现脚本停止运行后仍然有不断的气泡信息出现的情况。 例如:

```js
for(var i = 0; i < 100; i++){
  toast(i);
```

```
}
```

运行这段程序以后，会很快执行完成，且不断弹出消息，在任务管理中关闭所有脚本也无法停止。 要保证气泡消息才继续执行可以用：

```js
for(var i = 0; i < 100; i++){
  toast(i);
  sleep(2000);
}
```

或者修改toast函数：

```js
var _toast_ = toast;
toast = function(message){
  _toast_(message);
  sleep(2000);
}
for(var i = 0; i < 100; i++){
  toast(i);
}
```

toastLog(message)

- message {string} 要显示的信息

相当于 `toast(message);log(message)` 。显示信息message并在控制台中输出。参见console.log。

waitForActivity(activity[, period = 200])

- `activity` Activity名称
- `period` 轮询等待间隔（毫秒）

等待指定的Activity出现，period为检查Activity的间隔。

waitForPackage(package[, period = 200])

- `package` 包名
- `period` 轮询等待间隔（毫秒）

等待指定的应用出现。例如 `waitForPackage("com.tencent.mm")` 为等待当前界面为微信。

exit()

立即停止脚本运行。

立即停止是通过抛出 `ScriptInterruptedException` 来实现的，因此如果用 `try...catch` 把exit()函数的异常捕捉，则脚本不会立即停止，仍会运行几行后再停止。

random(min, max)

- `min` {number} 随机数产生的区间下界
- `max` {number} 随机数产生的区间上界
- 返回 {number}

返回一个在[min...max]之间的随机数。例如random(0, 2)可能产生0, 1, 2。

random()

- 返回 {number}

返回在[0, 1)的随机浮点数。

requiresApi(api)

- `api` Android版本号

表示此脚本需要Android API版本达到指定版本才能运行。例如 `requiresApi(19)` 表示脚本需要在Android 4.4以及以上运行。

调用该函数时会判断运行脚本的设备系统的版本号，如果没有达到要求则抛出异常。

可以参考以下Android API和版本的对照表:

平台版本： API级别

Android 7.0： 24

Android 6.0： 23

Android 5.1： 22

Android 5.0： 21

Android 4.4W： 20

Android 4.4： 19

Android 4.3： 18

requiresAutojsVersion(version)

- `version` {string} | {number} Auto.js的版本或版本号

表示此脚本需要Auto.js版本达到指定版本才能运行。例如 `requiresAutojsVersion("3.0.0 Beta")` 表示脚本需要在Auto.js 3.0.0 Beta以及以上运行。

调用该函数时会判断运行脚本的Auto.js的版本号，如果没有达到要求则抛出异常。

version参数可以是整数表示版本号，例如 `requiresAutojsVersion(250)` ；也可以是字符串格式表示的版本，例如"3.0.0 Beta", "3.1.0 Alpha4", "3.2.0"等。

可以通过 `app.autojs.versionCode` 和 `app.autojs.versionName` 获取当前的Auto.js版本号和版本。

runtime.requestPermissions(permissions)

- `permissions` {Array} 权限的字符串数组

动态申请安卓的权限。例如：

```
js
//请求GPS权限
```

```
runtime.requestPermissions(["access_fine_location"]);
```

尽管安卓有很多权限，但必须写入Manifest才能动态申请，为了防止权限的滥用，目前Auto.js只能额外申请两个权限：

- `access_fine_location` GPS权限
- `record_audio` 录音权限

您可以通过APK编辑器来增加Auto.js以及Auto.js打包的应用的权限。

安卓所有的权限列表参见Permissions Overview。（并没有用）

runtime.loadJar(path)

- `path` {string} jar文件路径

加载目标jar文件，加载成功后将可以使用该Jar文件的类。

```js
// 加载jsoup.jar
runtime.loadJar("./jsoup.jar");
// 使用jsoup解析html
importClass(org.jsoup.Jsoup);
log(Jsoup.parse(files.read("./test.html")));
```

(jsoup是一个Java实现的解析Html DOM的库，可以在Jsoup下载)

runtime.loadDex(path)

- `path` {string} dex文件路径

加载目标dex文件，加载成功后将可以使用该dex文件的类。

因为加载jar实际上是把jar转换为dex再加载的，因此加载dex文件会比jar文件快得多。可以使用Android SDK的build tools的dx工具把jar转换为dex。

context

全局变量。一个android.content.Context对象。

注意该对象为ApplicationContext，因此不能用于界面、对话框等的创建。

上一页
设备 - Device
基础

下一页
基于控件的操作
基础