目录

DeekeScript介绍

DeekeScript是什么?

DeekeScript是一款基于Android无障碍的模拟用户操作、控制用户手机的APP（框架）。用户可以通过DeekeScript快速开发出各种商用应用。

DeekeScript优势有哪些?

- 采用V8引擎，执行速度有较大提升
- 适配JavaScript，开发速度大幅提升
- 开发团队打造过自己的引流获客产品“嘀客APP”（现已迁移到DeekeScript中）
- 打包后的APP体验十分丝滑，界面高端美观大气（可参考“嘀客APP”）
- 支持Vscode开发与调试
- 为模拟操作和控制APP而生，全平台支持（无阉割）
- 个人版和企业版（多一个打包功能）功能完全一致
- 企业版支持后台管理系统（私有化部署，可以实现后台开账号，开代理商）
- DeekeScript将在未来提供强大的云市场，企业版用户可以自定义选择打包进自己的APP中

DeekeScript适合哪些场景?

在绝大多数需要取代人工，或者模拟人工进行的操作，都可以采用DeekeScript解决方案。比如：某音自动化获客，某信自动打招呼等等

- 如果你想获得一个稳定性好，执行速度快的第三方移动开发框架
- 如果你想快速开发出自己的商用模拟控制软件
- 如果你想快速开发出美观拿得出手的APP
- 如果你想私有化部署自己的程序，而不想被其他第三方控制
- 如果你不懂程序，但是懂销售，想在互联网行业创造一番天地
- 如果你想...

DeekeScript展望未来

未来，DeekeScript将会如何发展？我们在未来的6个月将会在以下几个地方发力：

- APP支持各种主题定制
- 丰富各种Api
- 开发应用市场
    - 开发者可以将开发的功能发布在应用市场获取收益
    - 用户只需要在DeekeScript软件内，即可使用软件内的使用插件）
    - 商家在后台即可选择应用市场的任意功能进行打包，打造自己的App（支持私有化部署）

案例篇

嘀客

嘀客（查看项目）作为DeekeScript工具开发的经典案例之一，具有参考价值；通过案例，可以让你了解到DeekeScript能做什么，以便你能够快速地做技术选型，开发“嘀客”需要做以下几步：

开发出这样的一款带App需要做哪些工作?

- 电脑端下载Vscode，安装DeekeScript开发插件
- 拉取DeekeScript开发的基础代码
- 编写JavaScript脚本
- 设计几个图标（当然，也可以在iconfont.cn平台下载几个）
- 手机安装DeekeScript App软件
- 不断调试运行，确保代码文档、无bug
- 编写激活码后端Api（需要配置在DeekeScript中，便于你的打包后的App激活）
- 上传开发脚本，DeekeScript后台打包成App

此时你可以将打包后的App提供给你的客户群体去使用，将你的激活码提供给客户，客户即可使用你开发的App工具了。

如果你是企业用户（或者具备企业权限），你可能还需要以下功能：

- 在DeekeScript开放平台注册管理后台
- 将相关接口配置到DeekeScript开发环境中

DeekeScript后台支持贴牌代理模式，后台有三种角色，分别是平台、代理商、用户；平台可以开通代理商账号（分配激活码数量），代理商可以开通普通用户（分配激活码数量）和激活码，普通用户可以创建激活码； 因此，你可以把“代理商”看作这是“城市合伙人”， “用户”看作是“区域合伙人”，区域合伙人创建激活码给到终端用户使用。

快速入门

认识DeekeScript

为什么会有DeekeScript？早些年，我曾服务于国内某互联网公司，主导引流相关的业务；当时因为公司业务需要，想采用手机端运行脚本的方式执行引流操作，经过几天时间的全面调研，最终选择了国内小有名气的某开源脚本软件。

后来，我自己基于此开源软件打造了一款引流获客软件 — 嘀客APP。当然，现在的嘀客APP已经被迭代了好几个版本，个中心酸无以言表。

在嘀客APP产品运作过程中，技术团队遇到了太多奇怪的问题、不稳定、闪退、息屏等等。这些问题为我们开拓市场埋下了很大的隐患，甚至影响到产品的未来。

再后来，我们决定对嘀客APP进行重构，解决当前所有问题。我们发现要想解决这个根本问题，必须得开发自己的底层框架。否则根本问题无法解决。最终我们决定在2023年年底启动了DeekeScript项目，历时3个多月的时间，我们完成了DeekeScript的1.0版本，并且把原先的嘀客APP也迁移进DeekeScript了。现在嘀客APP的体验提升了很多，运行速度也大幅提升。不再有运行一段时间挂掉，或者无法识别到界面的节点等情况发生。

为什么选择DeekeScript?

DeekeScript为稳定而生，脚本在充电状态下，可以持续挂机，不会出现离奇的中断事件；更不会出现一段时间后无法识别APP界面节点等问题。结合我们这几年在创业过程中积累的经验，我们十分了解开发者需要什么； 并且我们还把原有的嘀客后台提供给开发者使用，开发者现在只需提供几个图标和几个JavaScript脚本文件，就可以快速打造一个创业项目。**原本几个月才能完成的事情，现在只需要1-2天即可完成。**我们把开发者遇到的坑都走了一遍，我们更懂得开发者的诉求!

开发环境说明

Android版本支持

DeekeScript对绝大多数Android版本都提供支持，主要支持Android 8.0及以上版本；对应的Android Api版本是26及以上。

Android权限说明

使用DeekeScript可以获取Android的哪些权限呢？ 目前DeekeScript不限制应用获取Android权限，只要Android开放的权限，DeekeScript都会开放；DeekeScript默认支持了大多数常用权限，对于DeekeScript默认不支持的权限，可以通过自定义权限方式获取（注意此方式，需要将开发脚本打包成App）。

JavaScript支持哪些能力?

DeekeScript底层是基于V8引擎的，V8 是 Google 的开源高性能 JavaScript 和 WebAssembly 引擎，用 C++ 编写。它用于 Chrome 和 Node.js 等。 因此不用担心DeekeScript支持JavaScript的功能不全的问题。

基础函数支持

很多同学之前可能了解浏览器环境下的JavaScript和nodejs，因此可能习惯了使用setTimeout、setInterval、console.log等方法，这些DeekeScript都从底层进行了实现；因此可以放心使用这些函数。

是否支持模块化

另外关注点更多的可能是模块化部分，因为DeekeScript本身就是为了支持项目化（针对某个或者某些应用开发N个脚本）开发的工具（而不仅仅只是支持单个脚本的开发），所以对模块化支持也是很友好的，具体可以参阅模块化

异步支持

DeekeScript提供了异步支持，具体参考DeekeScript异步

是否支持多线程?

相信富有经验的开发者，会发现很多工具都有提供多线程支持；因为很多时候，我们需要同时执行两个操作（比如，采集直播间弹幕的时候，又要实时获取在线人数）；这个时候使用多线程的方式可以更好地满足我们的需求。
DeekeScript本身也是支持多线程的；因为JavaScript是不支持多线程的，实际上底层是由Java在执行，具体可以参考多线程篇。

其他支持

类似http请求，websocket都有支持，具体可以参考文档对应部分

快速开始

创建你的第一个JavaScript脚本

等待后续更新

配置

deekeScript.json文件说明

示例请查看本页末尾部分

主体参数

参数名	类型	示例	说明
name	String	嘀客	App安装成功之后，在手机上的名称
icon	String	logo/dke.png	图标建议采用200*200像素的，清晰度大的
head	String	img/root.png	App中用户设置页面的头像，无设置页可以不填
settingTopBg	String	img/sett-top.png	App中用户设置页面的背景图，可以不设置，但是建议设置
methods	Json	methods参数	主界面的功能列表，每一个子节点对应一个功能
bottomMenus	Json	bottomMenus参数	App中底部菜单，可以使用系统内置的，也可以自定义
settingLists	Json	settingLists参数	App中设置页的列表项，可以使用系统内置的，也可以自定义
api	Json	api参数	设置相关api，比如激活码api，验证激活码是否有效api登

methods参数

参数名	类型	示例	说明
title	String	XX截流	App主界面的功能名称
icon	String	logo/fans.png	App主界面的功能图标
jsFile	String	tasks/task_dy_toker_fans.js	功能实际执行的代码所在文件
settingPage	Json	settingPage参数	功能对应的设置页面，如果为空，则直接执行jsFile脚本

bottomMenus参数

参数名	类型	示例	说明
title	String	首页	底部菜单名称名称
icon	String	logo/fans.png	底部菜单图标
banner	String	banner/banner.png	type为home的时候生效，首页顶部图片
type	String	home	目前支持home、setting、speech，分别表示首页、设置页、话术页

settingLists参数

参数名	类型	示例	说明
title	String	清理缓存	设置页标题名称
icon	String	logo/clear.png	底部菜单图标
url	String	https://script.deeke.top/upload/log	type为uploadLog的时候，必须，用于接受上传日志
type	String	clear	支持clear、uploadLog、customerService、update、custom，分别表示 清理缓存、上传日志、联系客服、更新App、自定义
jsFile	String	home	type为custom的时候必须，点击的时候，会执行对应的js文件
description	String	确定清理吗?	type为clear的时候，会弹出提示框，确认后执行清理；type为customerService可以设置为“客服微信：miniphper”

api参数

参数名	类型	示例	说明
url	String	https://script.deeke.top/api/login	接口地址，type为login，则为激活码激活地址；type为checkLogin，则为激活码验证地址
type	String	login	目前支持login、checkLogin ； 其中login用于激活码登录，checkLogin用于每30分钟检查一次状态，状态不对则软件需要重新激活

settingPage参数

参数名	类型	示例	说明
jsFile	String	执行文件，保留字段	保留字段
params	Json	params参数	需要用户设置的参数

params参数

参数名	类型	示例	说明
type	String	text	Form参数类型，有text、textArea、select、checkbox、radio、switch、number、numberRange、digitRange、digit
lable	String	用户账号	字段描述，控件的描述，用于告诉用户这个控件输入的内容
value	String	miniphper	初始值，可以为空
name	String	account	控件名称，后续获取值的时候，需要这个参数名称
min	int	0	最小值，当type为numberRange或者digitRange时生效
max	int	1000	最大值，当type为numberRange或者digitRange时生效
step	int	1	滑动最小单位，当type为numberRange或者digitRange时生效
options	Json	options参数	属性值，当type为select、checkbox、radio时生效

options参数

参数名	类型	示例	说明
selected	bool	false	是否默认选中
lable	String	男	描述，用于告诉用户这个控件的内容
value	String	1	值

```
{
  "name": "嘀客",
  "icon": "logo/dke.png",
  "head": "img/robot.png",
  "settingTopBg": "img/setting-top.png",
  "methods": [
    {
      "title": "XX截流",
      "icon": "img/fans.png",
      "jsFile": "tasks/task_dy_toker_fans.js",
      "settingPage": {
        "jsFile": "page/fans.js",
        "params": [
            {type:"text", lable:"账号", name:"account", "value":"miniphper"},
            {type:"digit", lable:"执行次数", name:"times", "value":100},
            {type:"numberRange", lable:"最小作品数", name:"works_count", "value":100, min: 0, max: 10000, step: 1},
            {type:"digitRange", lable:"私信评率", name:"pri_msg_rate", "value":0.5, max: 1, min: 0},
            {type:"checkbox", lable:"性别", name:"private_msg_rate", options: [
                {lable: "男", value: 1, selected: true},
                {lable: "女", value: 0, selected: true},
                {lable: "未知", value: 2, selected: false},
            ]},
            {type:"switch", lable:"点赞头像", name:"zan_avatar", value: true},
        ]
      }
    },
    {
      "title": "XXX截流",
      "icon": "img/anchor_focus.png",
      "jsFile": "tasks/task_dy_toker_focus.js"
    }
  ],
  "bottomMenus": [
    {
      "title": "嘀客",
      "icon": "img/home.png",
      "banner": "img/home-top.png",
      "type": "home"
    },
    {
      "title": "语术设置",
      "icon": "img/speech.png",
      "type": "speech"
    },
    {
      "title": "系统设置",
      "icon": "img/setting.png",
      "type": "setting"
    }
  ],
  "settingLists": [
    {
      "title": "联系客服",
```

```
      "icon": "img/kefu.png",
      "type": "customerService",
      "description": "客服微信: miniphper"
    },
    {
      "title": "上传日志",
      "icon": "img/upload.png",
      "type": "uploadLog",
      "url": "https://top.deeke.script/uploadLog"
    },
    {
      "title": "系统升级",
      "icon": "img/update.png",
      "type": "updateApp",
      "url": "https://top.deeke.script/updateApp"
    },
    {
      "title": "清理数据",
      "icon": "img/clear.png",
      "description": "确定清理嘛? ",
      "type": "clear"
    },
    {
      "title": "自定义功能",
      "icon": "img/clear.png",
      "type": "custom",
      "jsFile": "app/custom.js"
    }
  ],
  "api": [
    {
      "type": "login",
      "url": "https://top.deeke.script/login"
    },
    {
      "type": "checkLogin",
      "url": "https://top.deeke.script/isLogin"
    }
  ]
}
```

VSCode开发篇

Vscode配置

等待更新

App

App对象提供了一些应用相关的数据获取能力，比如当前App的版本号、获取某个App的版本等。

currentPackageName

> 返回 {string}

获取当前应用的包名，获取你当前开发的App的包名，比如在DeekeScript中执行这个函数，将获取到“top.deeke.script”

```
console.log(App.currentPackageName());//输出 top.deeke.script
```

currentVersionCode

> 返回 {number} 整型

返回当前App的版本号，这个在App更新的时候判断当前App的版本号时很有用

currentVersionName

> 返回 {string}

返回当前App的版本名称

packageInfo

> 返回 {PackageInfo}

返回当前App的包信息，通过包信息你可以获取诸如“版本号，版本名称”等信息，可以遍历查看其属性和值

intent(json)

> json {json}
>
> 返回 {Intent}

创建Intent，也可以使用 new Intent();方式来创建

startActivity(object)

> options {object}
>
> 返回 {void}

根据选项构造一个Intent，并启动启动Activity

```
//打开应用来查看图片文件
var i = App.intent({
    action: "VIEW",
    type: "image/png",
    data: "file:///sdcard/1.png"
});
context.startActivity(i);
```

startService(options)

> options {object}
>
> 返回 {CompnentName}

根据选项构造一个Intent，并启动该服务。

sendBroadcast(options)

> options {object}
>
> 返回 {CompnentName}

根据选项构造一个Intent，并发送该广播。

launch(packageName)

> packageName {string}

通过包名打开应用

```
App.launch("top.deeke.script");//打开DeekeScript
```

openAppSetting(packageName)

> packageName {string}

通过包名，打开设置页面，在此页面可以无障碍点击卸载和强制停止按钮（停止应用的时候，很管用）

notifySuccess(title, content)

> title {string}
>
> content {string}
>
> 返回 {Notification}

通过Android的通知来告知用户，当前脚本已经执行完成，并且关闭当前启动的功能 如果只是通知完成，但是不关闭当前脚本，请自定义通知实现

系统函数 - System

System对象，提供一些常用的方法。

sleep(millisecond)

> millisecond {number} 毫秒

休眠millisecond毫秒

```
console.log('立即输出');
System.sleep(1000);
console.log('1秒钟后输出');
```

time()

> 返回 {string}

返回当前系统时间

```
console.log(System.time());//输出: 2024-03-07 12:12:12
```

currentActivity()

> 返回 {string}

返回最近一次监测到的正在运行的Activity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。

```
console.log(System.currentActivity());//输出: top.deeke.script
```

currentPackage()

> 返回 {string}

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

```
console.log(System.currentPackage());//输出: top.deeke.script
```

setClip(content)

> content {string}
>
> 返回 {void}

将内容写入到剪切板中

getClip()

> 返回 {string|null}

返回剪切板内容

toast(message)

> msg {string} 要显示的信息
>
> 返回 {void}

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。

toastLong(message)

> msg {string} 要显示的信息
>
> 返回 {void}

比toast(message)显示的更久一些，具体时长以开发机为准

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。

waitForActivity(activity, period, timeout)

> activity {string} 等待的Activity名称
>
> period {number} 等待的毫秒数
>
> timeout {number} 等待的总毫秒数
>
> 返回 {void}

等待Activity出现，period为检查Activity的间隔。如果timeout毫秒后未出现，则停止等待。

waitForPackage(package, period, timeout)

> package {string} 等待的Package名称
>
> period {number} 等待的毫秒数
>
> timeout {number} 等待的总毫秒数
>
> 返回 {void}

等待Package出现，period为检查Package的间隔。如果timeout毫秒后未出现，则停止等待。

exit(closeAll)

> closeAll {boolean} 是否关闭所有脚本引擎，为true则关闭所有，否则只关闭当前引擎外的其他脚本引擎
>
> 返回 {void}

关闭脚本引擎

设备 - Device

Device对象，提供一些设备相关的方法。

keepScreenOn(seconds)

> seconds {number} 屏幕常量秒数 返回 {void}

屏幕常量，如果seconds为0，则亮屏10分钟，否则亮屏seconds秒。seconds最大值可能在不同设备上有一定限制。保证屏幕常量，一般在操作（操作过程中，屏幕一般不会息屏）完相关步骤后，回到我们的App应用（在我们自己的应用上，可以保证长时间不息屏），即可保证不会息屏

```
Device.keepScreenOn(600);//常量10分钟
```

closeScreenLight()

> 返回 {void}

常量关闭

width()

> 返回 {number} 返回宽度 单位为px

获取屏幕真实的宽度（像素值）；手机旋转方向后，依然会返回真实的宽度

height()

> 返回 {number} 返回宽度 单位为px

获取屏幕真实的高度（像素值）；手机旋转方向后，依然会返回真实的高度

sdkInt()

> 返回 {number}

返回SDK的版本号

device()

> 返回 {string}
>
> 请注意，这个返回值并不是设备的市场名称（如“华为荣耀V30”）、品牌名（如“华为”）或型号名（如“V30”）。它更偏向于一个内部或技术名称，用于在Android框架和开发中识别不同的设备硬件。

获取设备的硬件标识，比如华为荣耀，可能返回"honor_xxx"的字符串

androidVersion()

> 返回 {string}

返回Android的版本号

createUuid()

> 返回 {string}
>
> 由于Android日益严格的权限管控，导致设备的唯一标识不太方便获取，本系统提供了一个uuid生成方案，来解决设备识别问题；建议在开发者在用户激活设备之后，使用此方法创建一个uuid（创建前，请使用getUuid检查是否存在）；后续将这个唯一标识和激活码绑定；
>
> 注意：App卸载后、设备恢复出厂等操作，将会让生成的uuid丢失

设置设备的随机的uuid

getUuid()

> 返回 {string}

获取设备的随机的uuid

isScreenOn()

> 返回 {boolean}

获取屏幕是否亮屏

brand()

> 返回 {string}

获取设备的品牌信息；对于华为（Huawei）的设备，它会返回 "huawei"。

os()

> 返回 {string}

获取设备操作系统信息；对于大多数标准的 Android 设备，Build.VERSION.BASE_OS 通常返回 "android"。然而，在某些设备或定制 Android 版本中，制造商可能会为其定制的基础操作系统设置不同的名称或标识符。

model()

> 返回 {string}

获取设备的model信息；如：荣耀特定设备的型号名称，如 "Honor V30" 或类似的字符串

codename()

> 返回 {string}

获取设备的codename信息；例如 "REL" 表示正式发布的版本

选择器 - UiSelector

基本介绍

UiSelector 即选择器，用于通过各种条件选取屏幕上的控件，再对这些控件进行点击、长按等动作。这里需要先简单介绍一下控件和界面的相关知识。

Android中的界面是由一个个控件构成的，例如图片部分是一个图片控件(ImageView)，文字部分是一个文字控件(TextView)；同时，通过各种布局来决定各个控件的位置，例如，线性布局(LinearLayout)里面的控件都是按水平或垂直一次叠放的，列表布局(AbsListView)则是以列表的形式显示控件。

控件有各种属性，包括文本(text), 描述(desc), 类名(className)，是否可以点击（clickable）, id 等等。我们通常用一个控件的属性来找到这个控件，例如，想要点击某软件聊天窗口的"发送"按钮，我们就可以通过他的文本属性为"发送"来找到这个控件并点击他，具体代码为:

> 获取控件后，即可对控件进行点击，滑动，输入文本等操作。控件操作请阅读 **控件操作** 部分

```
//这里的sendButton就是一个控件对象，可以对控件对象进行各种操作
let sendButton = new UiSelector().text("发送").findOne();
if(sendButton){
    sendButton.click();
}
```

id(name)

> name {string}
>
> 返回 {UiSelector} 返回选择器自身以便链式调用

为当前选择器附加控件"id 等于字符串 name"的筛选条件。

className(name)

> name {string}
>
> 返回 {UiSelector} 返回选择器自身以便链式调用

为当前选择器附加控件"className 等于字符串 name"的筛选条件。

bounds(left, top, right, bottom)

> left {number} 控件左边缘与屏幕左边的距离
>
> top {number} 上边距
>
> right {number} 右边距
>
> bottom {number} 下边距
>
> 返回 {UiSelector} 返回选择器自身以便链式调用

为当前选择器附加控件"bounds范围"的筛选条件。

text(content)

> content {string}
>
> 返回 {UiSelector} 返回选择器自身以便链式调用

为当前选择器附加控件"text 等于字符串 content"的筛选条件。

desc(content)

> content {string}
>
> 返回 {UiSelector} 返回选择器自身以便链式调用

为当前选择器附加控件"contentDescribe 等于字符串 content"的筛选条件。

clickable(canClick)

> canClick {boolean}
>
> 返回 {UiSelector} 返回选择器自身以便链式调用

为当前选择器附加控件"clickable 等于 canClick"的筛选条件。

selected(isSelected)

> isSelected {boolean}
>
> 返回 {UiSelector} 返回选择器自身以便链式调用

为当前选择器附加控件"selected 等于 isSelected"的筛选条件。

checked(isChecked)

> isChecked {boolean}
>
> 返回 {UiSelector} 返回选择器自身以便链式调用

为当前选择器附加控件"checked 等于字符串 isChecked"的筛选条件。

enabled(isEnabled)

> isEnabled {boolean}
>
> 返回 {UiSelector} 返回选择器自身以便链式调用

为当前选择器附加控件"enabled 等于字符串 isEnabled"的筛选条件。

scrollable(canScrollable)

> canScrollable {boolean}
>
> 返回 {UiSelector} 返回选择器自身以便链式调用

为当前选择器附加控件"scrollable 等于字符串 canScrollable"的筛选条件。

checkable(isCheckable)

> isCheckable {boolean}
>
> 返回 {UiSelector} 返回选择器自身以便链式调用

为当前选择器附加控件"checkable 等于字符串 isCheckable"的筛选条件。

textContains(content)

> content {string}
>
> 返回 {UiSelector} 返回选择器自身以便链式调用

为当前选择器附加控件"text 包含字符串 content"的筛选条件。

textMatches(content)

> content {string}
>
> 返回 {UiSelector} 返回选择器自身以便链式调用

为当前选择器附加控件"text 正则匹配 content"的筛选条件。

descContains(content)

> content {string}
>
> 返回 {UiSelector} 返回选择器自身以便链式调用

为当前选择器附加控件"contentDescribe 正则匹配 content"的筛选条件。

descMatches(content)

> content {string}
>
> 返回 {UiSelector} 返回选择器自身以便链式调用

为当前选择器附加控件"contentDescribe 正则匹配 content"的筛选条件。

filter(callback)

> callback {function}
>
> 返回{UiObject[]}

对当前查找到的UiObject数组进行过滤，过滤的时候执行callback方法，该方法返回false，则对应的UiObject被过滤掉

exist()

> content {string}
>
> 返回 {boolean} 返回是否存在

判断当前选择器是否能匹配到UiObject控件信息。

find()

> 返回 {UiObject[]}

获取当前选择器筛选的所有UiObject控件。

find(uiSelector)

> uiSelector {UiSelector}
>
> 返回 {UiObject[]}

获取当前选择器筛选的所有UiObject控件（a）中查找符合uiSelector选择器的UiObject；从a集合查找，然后遍历它们的子控件、子控件的子控件，直到a下面的所有控件都被查找一遍才结束。

find(timeout)

> timeout {number}
>
> 返回 {UiObject[]}

获取当前选择器筛选的所有UiObject控件，查找timeout时间，时间结束后不管是否找到，都会结束查找，并且返回。

findOne()

> 返回 {UiObject}

获取当前选择器筛选的第一个UiObject控件。

findOnce()

> 返回 {UiObject}

获取当前选择器筛选的第一个UiObject控件。

findOne(uiSelector)

> uiSelector {UiSelector}
>
> 返回 {UiObject}

获取当前选择器筛选的第一个UiObject控件（a）中查找符合uiSelector选择器的UiObject；从a集合查找，然后遍历它们的子控件、子控件的子控件，直到a下面的所有控件都被查找一遍才结束。

find(timeout)

> timeout {number}
>
> 返回 {UiObject}

获取当前选择器筛选的第一个UiObject控件，查找timeout时间，时间结束后不管是否找到，都会结束查找，并且返回。

定时器

用于在未来某个未来时间执行函数；计时器函数实现了与 Web 浏览器提供的定时器类似的 API。

setTimeout(callback, delay)

> callback {callback} 要执行的函数
>
> delay {number} 延时 毫秒数
>
> 示例：

```
setTimeout(()=>{
    console.log('2秒后执行');
}, 2000);
```

setInterval(callback, delay)

> callback {callback} 要执行的函数
>
> delay {number} 延时 毫秒数
>
> 示例：

```
setInterval(()=>{
    console.log('每间隔2秒执行一次');
}, 2000);
```

setImmediate(callback)

> callback {callback} 要执行的函数

在Looper循环的当前回合结束时要调用的函数

> 示例：

```
setImmediate(()=>{
    console.log('脚本最后执行');
});
```

clearTimeout(id)

取消一个由setTimeout(callback, delay)创建的定时任务

> 示例：

```
let timer = setTimeout(()=>{
    console.log(123);
}, 1000);

clearTimeout(timer);
```

clearInterval(id)

取消一个由setInterval(callback, delay)创建的定时任务

> 示例：

```
let timer = setInterval(()=>{
    console.log(123);
}, 1000);

clearInterval(timer);
```

clearImmediate(id)

取消一个由setImmediate(callback, delay)创建的定时任务

> 示例：

```
let timer = setImmediate(()=>{
    console.log(123);
});

clearImmediate(timer);
```

网络请求-Http

Http模块提供一些进行http请求的函数。

get(url, headers)

> url {string} 请求地址
>
> headers {json} 请求头 可以为 null

```
let res = Http.get('https://script.deeke.top/api/userInfo');
console.log(res);//输出 {code: 0, msg: "成功", data: {name: "miniphper", weixin: "miniphper"}}
```

post(url, params, headers)

> url {string} 请求地址
>
> params {json} 请求参数
>
> headers {json} 请求头 可以为 null
>
> post请求的Content-type为"application/json; charset=utf-8"

```
let params = {
    account: "miniphper",
    password: "miniphper"
}

let res = Http.post('https://script.deeke.top/api/login', params, null);
console.log(res);//输出 {code: 0, msg: "成功", data: []}
```

postFile(url, files, params, callback)

> url {string} 请求地址
>
> files {File[]} 请求文件列表
>
> params {json} 请求参数
>
> headers {json} 请求头 可以为 null

```
let files = [new File("文件地址"), new File("文件地址2")];
let params = {username: "miniphper"};

let res = Http.postFile('https://script.deeke.top/api/fileUpload', files, params, {
    success: (response)=>{
        //处理逻辑
        let js = response.json();
        console.log(js);
    },
    fail: (response)=>{
        //失败逻辑
    }
});
```

webSocket

create

后续更新

控制台 - console

控制台模块提供了一个和Web浏览器中相似的用于调试的控制台。用于输出一些调试信息、中间结果等。 console模块中的一些函数也可以直接作为全局函数使用，例如log, info等。

> 如果你不仅仅需要打印数据到控制台，还需要记录日志到文件，请使用 **日志-Log** 相关的方法
>
> 注意：console是DeekeScript中唯一一个首字母小写的对象（为了适应用户习惯）

log([data][, ...args])

打印到控制台，并带上换行符。 可以传入多个参数，第一个参数作为主要信息，其他参数作为类似于 printf(3) 中的代替值（参数都会传给 util.format()）。

```
console.log("输出的内容", 324, {name: "张三"});
```

info([data][, ...args])

与console.log类似，但输出结果以绿色字体显示。输出优先级高于log, 用于输出重要信息。

warn([data][, ...args])

与console.log类似，但输出结果以蓝色字体显示。输出优先级高于info, 用于输出警告信息。

error([data][, ...args])

与console.log类似，但输出结果以红色字体显示。输出优先级高于warn, 用于输出错误信息。

日志 - Log

日志模块提供了一些日志写入方法，例如log, info等。

> 如果你不仅仅需要打印数据到控制台，还需要记录日志到文件，请使用Log相关的方法

setFile(filename);

设置日志输出文件

```
//未打包: 最终存储在 /data/data/top.deeke.script/files/log/myfile.log
//打包后: 最终存储在 /data/data/com.example.myapp/files/log/myfile.log
Log.setFile("myfile.log");
```

getFileDir(filename);

获取日志文件完整路径

```
let filename = Log.getFileDir("myfile.log");

//未打包: 返回内容为 /data/data/top.deeke.script/files/log/myfile.log
//打包后: 返回内容为 /data/data/com.example.myapp/files/log/myfile.log
Log.log(filename);
```

log([data][, ...args])

与console.log类似，区别是会记录输出内容到文件。

info([data][, ...args])

与console.info类似，区别是会记录输出内容到文件。

warn([data][, ...args])

与console.warn类似，区别是会记录输出内容到文件。

error([data][, ...args])

与console.error，区别是会记录输出内容到文件。

本地存储

本地存储模块提供了一些数据写入和读取的功能。

> 存储模块底层使用的Android DataStore实现

createDataStore(namespace);

> namespace {string}
>
> 返回 {boolean}

设置存储文件，不同模块可以设置不同的namespace 【对应Android中的filename】

> 注意：前缀为"deekeScript:XXX"的namespace被系统使用，用户只能读取不能写入

```
Storage.createDataStore(namespace);
```

put(key, value);

> key {string}
>
> value {any}
>
> 返回 {boolean}

```
let user = {
    name: "张三",
    age: 22
};

Storage.put("user", user);
```

get(key);

> key {string}
>
> 返回 {any}

输出键为key的值，不存在的时候返回null

```
let user = Storage.get("user");
Log.log(user);//输出json数据  {name: "张三", age: 22}
```

remove(key);

> key {string}
>
> 返回 {boolean}

存在则删除，不存在则什么都不做

```
let user = Storage.remove("user");
Log.log(Storage.get("user"));//输出 null
```

clear();

> 返回 {boolean}

清空所有内容

```
Log.log(Storage.clear());//输出 true
```

Files

Files模块提供了一些常见的文件处理，包括文件读写、移动、复制、删掉等。

一次性的文件读写可以直接使用Files.read(), Files.write(), Files.append()等方便的函数;

isFile(path)

> path {string} 路径
>
> 返回 {boolean}

返回路径path是否是文件。

```
console.log(Files.isDir("/sdcard/文件夹/")); //返回false
console.log(Files.isDir("/sdcard/文件.txt")); //返回true
```

isDir(path)

> path {string} 路径
>
> 返回 {boolean}

返回路径path是否是文件夹。

```
console.log(Files.isDir("/sdcard/文件夹/")); //返回true
console.log(Files.isDir("/sdcard/文件.txt")); //返回false
```

isEmptyDir(path)

> path {string} 路径
>
> 返回 {boolean}

返回文件夹path是否为空文件夹。如果该路径并非文件夹，则直接返回false。

join(parent, child)

> parent {string} 父目录路径
>
> child {string} 子路径
>
> 返回 {string}

连接两个路径并返回，例如Files.join("/sdcard/", "1.txt")返回"/sdcard/1.txt"。

create(path)

> path {string} 路径
>
> 返回 {boolean}

创建一个文件或文件夹并返回是否创建成功。如果文件已经存在，则直接返回false。

createWithDirs(path)

> path {string} 路径
>
> 返回 {boolean}

创建一个文件或文件夹并返回是否创建成功。如果文件所在文件夹不存在，则先创建他所在的一系列文件夹。如果文件已经存在，则直接返回false。

exists(path)

> path {string} 路径
>
> 返回 {boolean}

返回在路径path处的文件是否存在。

ensureDir(path)

> path {string} 路径
>
> 确保路径path所在的文件夹存在。如果该路径所在文件夹不存在，则创建该文件夹。

例如对于路径"/sdcard/Download/ABC/1.txt"，如果/Download/文件夹不存在，则会先创建Download，再创建ABC文件夹。

read(path[, encoding = "utf-8"])

> path {string} 路径
>
> encoding {string} 字符编码，可选，默认为utf-8
>
> 返回 {string}

读取文本文件path的所有内容并返回。如果文件不存在，则抛出FileNotFoundException。

readBytes(path)

> path {string} 路径
>
> 返回 {byte[]}

读取文件path的所有内容并返回一个字节数组。如果文件不存在，则抛出FileNotFoundException。

write(path, text[, encoding = "utf-8"])

> path {string} 路径
>
> text {string} 要写入的文本内容
>
> encoding {string} 字符编码

把text写入到文件path中。如果文件存在则覆盖，不存在则创建。

writeBytes(path, bytes)

> path {string} 路径
>
> bytes {byte[]} 字节数组，要写入的二进制数据

把bytes写入到文件path中。如果文件存在则覆盖，不存在则创建。

append(path, text[, encoding = 'utf-8'])

> path {string} 路径
>
> text {string} 要写入的文本内容
>
> encoding {string} 字符编码

把text追加到文件path的末尾。如果文件不存在则创建。

appendBytes(path, text[, encoding = 'utf-8'])

> path {string} 路径
>
> bytes {byte[]} 字节数组，要写入的二进制数据

把bytes追加到文件path的末尾。如果文件不存在则创建。

copy(fromPath, toPath)

> fromPath {string} 要复制的原文件路径
>
> toPath {string} 复制到的文件路径
>
> 返回 {boolean}

复制文件，返回是否复制成功。例如Files.copy("/sdcard/1.txt", "/sdcard/Download/1.txt")。

move(fromPath, toPath)

> fromPath {string} 要移动的原文件路径
>
> toPath {string} 移动到的文件路径
>
> 返回 {boolean}

移动文件，返回是否移动成功。例如Files.move("/sdcard/1.txt", "/sdcard/Download/1.txt")会把1.txt文件从sd卡根目录移动到Download文件夹。

rename(path, newName)

> path {string} 要重命名的原文件路径
>
> newName {string} 要重命名的新文件名
>
> 返回 {boolean}

重命名文件，并返回是否重命名成功。例如Files.rename("/sdcard/1.txt", "2.txt")。

renameWithoutExtension(path, newName)

> path {string} 要重命名的原文件路径
>
> newName {string} 要重命名的新文件名
>
> 返回 {boolean}

重命名文件，不包含拓展名，并返回是否重命名成功。例如Files.rename("/sdcard/1.txt", "2")会把"1.txt"重命名为"2.txt"。

getName(path)

> path {string} 路径
>
> 返回 {string}

返回文件的文件名。例如Files.getName("/sdcard/1.txt")返回"1.txt"。

getNameWithoutExtension(path)

> path {string} 路径
>
> 返回 {string}

返回不含拓展名的文件的文件名。例如## getName("/sdcard/1.txt")返回"1"。

getExtension(path)

> path {string} 路径
>
> 返回 {string}

返回文件的拓展名。例如Files.getExtension("/sdcard/1.txt")返回"txt"。

remove(path)

> path {string} 路径
>
> 返回 {boolean}

删除文件或空文件夹，返回是否删除成功。

removeDir(path)

> path {string} 路径
>
> path {string} 路径
>
> 返回 {boolean}

删除文件夹，如果文件夹不为空，则删除该文件夹的所有内容再删除该文件夹，返回是否全部删除成功。

getSdcardPath()

> 返回 {string}

返回SD卡路径。所谓SD卡，即外部存储器。

cwd()

> 返回 {string}

返回脚本的"当前工作文件夹路径"。该路径指的是，如果脚本本身为脚本文件，则返回这个脚本文件所在目录；否则返回null获取其他设定路径。

例如，对于脚本文件"/sdcard/脚本/1.js"运行Files.cwd()返回"/sdcard/脚本/"。

path(relativePath)

> relativePath {string} 相对路径
>
> 返回 {string}

返回相对路径对应的绝对路径。例如Files.path("./1.png")，如果运行这个语句的脚本位于文件夹"/sdcard/脚本/"中，则返回"/sdcard/脚本/1.png"。

listDir(path[, filter])

> path {string} 路径
>
> filter {Function} 过滤函数，可选。接收一个string参数（文件名），返回一个boolean值。

列出文件夹path下的满足条件的文件和文件夹的名称的数组。如果不加filter参数，则返回所有文件和文件夹。

手势

click(x, y)

> x {number} x轴坐标
>
> y {number} y轴坐标
>
> 返回 {boolean}

点击屏幕位置

```
click(100, 200);
```

longClick(x, y)

> x {number} x轴坐标
>
> y {number} y轴坐标
>
> 返回 {boolean}

长按屏幕位置

```
longClick(100, 200);
```

press(x, y, duration)

> x {number} x轴坐标
>
> y {number} y轴坐标
>
> duration {number} 按压时间 毫秒
>
> 返回 {boolean}

按压屏幕位置一段时间

```
press(100, 200, 10);//每次按压10毫秒
```

swipe(startX, startY, endX, endY, duration)

> startX {number} 开始位置的x轴坐标
>
> startX {number} 开始位置的y轴坐标
>
> endX {number} 结束位置的x轴坐标
>
> endY {number} 结束位置的y轴坐标
>
> duration {number} 滑动时间 毫秒
>
> 返回 {boolean}

滑动手势

```
swipe(100, 200, 500, 400, 200);//从坐标(100,200)滑动到(500, 400)，执行时间为200毫秒
```

back()

> 返回 {boolean}

点击返回按键

```
back();
```

home()

> 返回 {boolean}

点击Home按键，返回到主界面

```
home();
```

recents()

> 返回 {boolean}

显示最近任务

```
back();
```

控件操作

基本介绍

UiObject 即控件对象，可以对控件进行点击、长按等动作。这里需要先简单介绍一下控件的相关知识。

Android中的界面是由一个个控件构成的，例如图片部分是一个图片控件(ImageView)，文字部分是一个文字控件(TextView)；同时，通过各种布局来决定各个控件的位置。

> 获取控件后，即可对控件进行点击，滑动，输入文本等操作。获取控件操作请阅读 **获取控件操作** 部分

```
//这里的sendButton就是一个控件对象，可以对控件对象进行各种操作
let sendButton = new UiSelector().text("发送").findOne();
if(sendButton){
    sendButton.click();
}
```

click()

> 返回 {Boolean} 返回是否点击成功

点击控件

```
let sendButton = new UiSelector().text("发送").findOne();
if(sendButton){
    sendButton.click();
}
```

longClick()

> 返回 {Boolean} 返回是否点击成功

点击控件

```
let sendButton = new UiSelector().text("发送").findOne();
if(sendButton){
    sendButton.longClick();
}
```

scrollForward()

> 返回 {Boolean} 返回是否滑动成功

向前滑动控件，将会让界面下方的节点往上滚动或者右边的节点往左滚动

scrollBackward()

> 返回 {Boolean} 返回是否滑动成功

向后滑动控件，将会让界面上方的节点往下滚动或者左边的节点往右滚动

setSelection(startPosition, endPosition)

> startPosition {number} 起始位置
>
> endPosition {number} 结束位置
>
> 返回 {Boolean} 返回是否成功

选中文本内容

copy()

> 返回 {Boolean} 返回是否成功

复制选中的文本内容

```
let et = new UiSelector().className("EditText").findOne();
//选中前两个字
et.setSelection(0, 2);
//对选中内容进行复制
if(et.copy()){
    toast("复制成功");
}else{
    toast("复制失败");
}
```

cut()

> 返回 {boolean} 返回是否成功

对输入框文本的选中内容进行剪切，并返回是否操作成功。

该函数只能用于输入框控件，并且当前输入框控件有选中的文本。可以通过setSelection()函数来设置输入框选中的内容。

paste()

> 返回 {boolean} 返回是否成功

粘贴内容到文本框

```
System.setClip("DeekeScript");//将字符串“DeekeScript”粘贴到文本框
let obj = new UiSelector().className("EditText").findOne();
obj.paste();
```

focus()

> 返回 {boolean} 返回是否成功

获取焦点。

```
let obj = new UiSelector().className("EditText").findOne();
obj.focus();
```

setText(text)

> text {string} 输入参数 返回 {boolean} 返回是否成功

将字符串“text”输入到文本框

```
let obj = new UiSelector().className("EditText").findOne();
obj.setText("DeekeScript");//将字符串“DeekeScript”输入到文本框
```

find(uiSelector)

> uiSelector {UiSelector} 要查找的内容 返回 {UiObject[]|null} 返回查找到的控件对象

在当前的控件下查找某些控件，查找条件通过UiSelector定义

```
let obj = new UiSelector().className("name").findOne();
let uiObjects = obj.find(new UiSelector().className("EditText"));//查找obj下面的所有输入框控件
```

findOne(uiSelector)

> uiSelector {UiSelector} 要查找的内容 返回 {UiObject|null} 返回查找到的控件对象

在当前的控件下查找某个控件，查找条件通过UiSelector定义

```
let obj = new UiSelector().className("name").findOne();
let uiObject = obj.findOne(new UiSelector().className("EditText"));//查找obj下面的第一个输入框控件
```

bounds()

> 返回 {Rect}

返回当前控件的位置信息

```
let obj = new UiSelector().className("EditText").findOne();
let rect = obj.bounds();
//输出  左边距、上边距、右边距、下边距、高度、宽度
console.log(rect.left, rect.top, rect.right, rect.bottom, rect.height(), rec.width());
```

id()

> 返回 {string}

返回当前控件的id

```
let obj = new UiSelector().id("p2").findOne();
let str = obj.id();//输出p2
```

text()

> 返回 {string}

返回当前控件的text内容

```
let obj = new UiSelector().id("p2").findOne();
let str = obj.text();//控件的text属性获取
```

desc()

> 返回 {string}

返回当前控件的contentDescription内容

```
let obj = new UiSelector().id("p2").findOne();
let str = obj.desc();//控件的contentDescription属性获取
```

children()

> 返回 {UiObject[]|null}

返回当前控件的所有子控件对象

```
let obj = new UiSelector().id("p2").findOne();
let childs = obj.children();//子控件内容
```

parent()

> 返回 {UiObject}

返回当前控件的父控件对象

```
let obj = new UiSelector().id("p2").findOne();
let childs = obj.parent();//子控件内容
```

getDrawingOrder()

> 返回 {number} 层级

返回当前控件的 绘制层级

```
let obj = new UiSelector().id("p2").findOne();
let index = obj.getDrawingOrder();//控件绘制层级
```

isClickable()

> 返回 {boolean}

是否可以点击，相关的方法还有：
isFocusable/isScrollable/isLongClickable/isEnable/isPassword/isEditable/isVisibleToUser/isCheckable/isChecked/isSelected

setClickable(bool)

> 返回 {void}

设置是否可以点击，相关的方法还有：
setFocusable/setScrollable/setLongClickable/setEnable/setPassword/setEditable/setVisibleToUser/setCheckable/setChecked/setSelected

Engines

首先有必要介绍一下什么是Engines。你可以把Engines理解为JavaScript运行时的管理器。默认情况下，你的JavaScript运行在一个V8环境中。 在某些情况下，你可能需要在你的JavaScript代码中执行一段代码，但是希望这段代码是独立于当前环境（并且不阻塞当前的JavaScript运行）， 这个时候你就会使用Engines来创建一个新的JavaScript运行时，代码将会在一个新的V8环境中运行。

当然，在一些时候，如果你想彻底关闭JavaScript运行时，也是通过Engines来关闭的。

> 注意，如果你需要使用多线程，可以采用Engines来实现

exec(filename)

> filename {string} 要执行的JS文件地址 必须为相对项目根目录的地址，非当前文件的相对地址
>
> 返回 {Thread}

在新的环境中执行JavaScript，新的环境会开启新的堆栈空间，不会和当前的脚本有任何关联，是完全独立的。 比如：你想执行当前脚本的时候，又要定时与服务器进行沟通或者做一些其他事情，可以采用这种方式实现。

```
//heart.js主要启动一个定时器setInterval，每间隔几秒钟往服务器发送当前App的状态
Engines.exec("js/heart.js");//心跳脚本，监测App的在线状态

//特别说明，即使当前脚本终止运行了，heart.js依旧会继续运行
//如果需要关闭heart.js，可以使用 Engines.closeAll方法来操作;
```

closeAll(bool)

> bool {boolean} 是否关闭当前脚本，为true则关闭当前脚本，否则只关闭当前脚本以外的脚本
>
> 建议在脚本运行完成后进行关闭，否则可能会出现内存泄漏

关闭JavaScript运行时和运行时所在的线程

```
Engines.closeAll(true);//关闭所有脚本运行时，当前脚本所在运行时也会被结束
```

多线程

相信富有经验的开发者，会发现很多工具都有提供多线程支持；因为很多时候，我们需要同时执行两个操作（比如，采集直播间弹幕的时候，又要实时获取在线人数）；这个时候使用多线程的方式可以更好地满足我们的需求。DeekeScript本身也是支持多线程的；

> 特别说明一点，因为DeekeScript底层是基于V8的，而V8是基于单线程的
>
> 如果确实需要使用真正意义上的多线程，可以使用Engines来实现

Promise实现异步效果

当然，实际上，还可以通过异步的方式来实现你需要的效果，代码如下：

```
function task1() {
  return new Promise((resolve) => {
    setTimeout(() => {
      console.log('任务1执行完毕');
      resolve();// 当任务1完成时，resolve这个Promise
    }, 1000);
  });
}

function task2() {
  return new Promise((resolve) => {
    setTimeout(() => {
      console.log('任务2执行完毕');
      resolve();
    }, 1000);
  });
}

task1();
task2();
```

如果你需要循环执行某些动作，可以采用setInterval定时器来实现。

module - 模块

在很多时候，我们的应用可能不止1个JavaScript文件就能实现的，尤其是在构建复杂应用的时候。

在DeekeScript中，可以轻松通过import方法来加载JavaScript文件。下面是一个实例：

import与export

```
//项目文件结构如下：项目跟目录下有一个js文件夹和一个task文件夹
//js文件夹中包含了我们需要import的文件，task文件里面有一个dy_toker.js文件
//下面我们在dy_toker.js文件中引用js文件夹下面的a.js和b.js
// |> js
// | - a.js
// | - b.js
// |> task
// | - dy_toker.js
// |DeekeScript.json

// js/a.js文件代码如下
export let a = {
    name: "我是a.js",
    getName(){
        return this.name;
    }
}

//dy_toker.js代码如下  特别说明，路径是相对于项目根目录的路径
import {a} from "js/a.js";//非相对于dy_toker.js文件的路径
console.log(a.getName());//输出“我是a.js”
```

加解密

本模块主要囊如了一些场景的加密方式，和一些编码方式

base64Encode(str)

> str {string} 需要编码的字符串
>
> 返回 {string}

base64编码字符串

base64Decode(str)

> str {string} base64Encode后的字符串
>
> 返回 {string}

base64解密后的字符串

md5(str)

> str {string} 加密前的字符串
>
> 返回 {string} 加密后的字符串

md5加密

sha1(str)

> str {string} 加密前的字符串
>
> 返回 {string} 加密后的字符串

sha1加密

sha256(str)

> str {string} 加密前的字符串
>
> 返回 {string} 加密后的字符串
>
> sha256加密

aesCbcEncode(key, iv, str)

> key {string}
>
> iv {string}
>
> str {string} 需要加密的内容
>
> 返回 {string} 加密后的内容

采用AES-CBC加密

```
let iv = Encrypt.generateIv();
let key = "sdfsdl";
let encodeStr = Enctypt.aesCbcEncode(key, iv, str);
console.log(encodeStr);//输出加密后的内容
```

aesCbcDecode(key, iv, encodeStr)

> key {string}
>
> iv {string}
>
> str {string} 需要被解密的内容
>
> 返回 {string} 解密后的内容

采用AES-CBC模式解密

```
let iv = Encrypt.generateIv();//替换成实际的iv
let key = "sdfsdl";
let str = Enctypt.aesCbcEncode(key, iv, encodeStr);
console.log(encodeStr);//输出加密后的内容
```

图片与颜色

图片识别

等待后续更新

颜色识别

等待后续更新

扩展

DeekeScript将JavaScript运行在Android环境中，很多时候我们需要使用Java的一些类来实现一些能力， 从而弥补JavaScript的一些不足。

默认情况下，DeekeScript已经帮助用户注册了哪些基础的Java类呢?

> 下面的类，都默认导入，可以在JavaScript中直接使用

- import android.content.Intent;
- import java.nio.file.Files;
- import java.io.File;

Files类使用

> 完整例子，请阅读 文件系统-Files

```
console.log(Files.isDir("/sdcard/文件夹/")); //返回false
```

如何添加其他扩展?

等待后续更新

后端接口

激活码

需要在DeekeScript.json配置文件中配置Uri请求地址

DeekeScript请求：

```
Request Method: POST
Content-Type: application/json

请求参数:
{
    "activeCode": "用户输入的激活码",
    "androidId": "系统生成的唯一ID" //用户卸载App或者恢复出厂设置后，此ID将会变化
}
```

你的接口需要返回如下json内容：

```
//激活成功
{"code": 0, "status":"success"}

//激活失败
{"code": 1, "status":"fail", "msg": "激活失败"}
```

激活状态

需要在DeekeScript.json配置文件中配置Uri请求地址，DeekeScript每5分钟请求一次此接口，如果连续2次返回失败，则会强制退出App 并且会发生一个通知给客户（通知内容为接口返回的msg字段内容）

DeekeScript请求：

```
Request Method: POST
Content-Type: application/json

请求参数:
{
    "androidId": "激活时生成的AndroidId"
}
```

你的接口需要返回如下json内容：

```
//当前设备具备App使用权限
{"code": 0, "status":"success"}

//当前设备不具备App使用权限
{"code": 1, "status":"fail", "msg": "激活码已过期"}
```

在线打包

在线打包

等待更新

管理后台支持

DeekeScript给企业用户提供了后台管理系统支持，可以轻松管理App的激活码，开通代理商等。

界面

等待更新

云市场

云市场作为未来发力的重要模块之一，是DeekeScript接下来着重发展的一部分；但是目前我们团队缺少相关人才，以至于云市场功能还不能在短期内实现，预计在2024年年底能发布上线。

> 如果有同学关注这个方向，并且愿意贡献一部分力量，请联系作者（微信：DeekeScript）

Deeke

Deeke地址

可以访问地址：点击访问

问题汇总

DeekeScript未来会支持其他模式吗?

答：是的，DeekeScript目前对UI部分进行了强制的要求，后续会逐步推出自定义ui的模式

DeekeScript未来会一直维护更新吗?

答：是的，DeekeScript于2023年年底问世，主要是为了解决底层框架不稳定的问题，后续会持续增加功能，并且会修复历史存在的问题

DeekeScript官方如何联系?

答：DeekeScript目前是个人在维护与升级，如需联系作者，请添加微信号：DeekeScript；作者可以为你解答一些问题

DeekeScript未来会支持nodejs模式吗?

答：短期内，DeekeScript不会去兼容nodejs模式

DeekeScript未来会支持Typescript吗?

答：在未来的一段时间，会逐步提供支持