# APP

该模块提供一系列函数，用于使用其他应用、与其他应用交互。例如发送意图、打开文件、发送邮件等。

同时提供了方便的进阶函数 `startActivity` 和 `sendBroadcast` ，用他们可完成app模块没有内置的和其他应用的交互。

## app.versionCode

- `return` {number}

当前软件版本号，整数值。例如160, 256等。

如果在Auto.js中运行则为Auto.js的版本号；在打包的软件中则为打包软件的版本号。

```js
toastLog(app.versionCode);
```

## app.versionName

- `return` {string}

当前软件的版本名称，例如"3.0.0 Beta"。

如果在Auto.js中运行则为Auto.js的版本名称；在打包的软件中则为打包软件的版本名称。

```js
toastLog(app.verionName);
```

## app.autojs.versionCode

- `return` {number}

Auto.js版本号，整数值。例如160, 256等。

## app.autojs.versionName

- `return` {string}

Auto.js版本名称，例如"3.0.0 Beta"。

## app.launchApp(appName)

- `appName` {string} 应用名称

通过应用名称启动应用。如果该名称对应的应用不存在，则返回false; 否则返回true。如果该名称对应多个应用，则只启动其中某一个。

该函数也可以作为全局函数使用。

```js
launchApp("Auto.js");
```

## app.launch(packageName)

- `packageName` {string} 应用包名

通过应用包名启动应用。如果该包名对应的应用不存在，则返回false；否则返回true。

该函数也可以作为全局函数使用。

```js
//启动微信
launch("com.tencent.mm");
```

## app.launchPackage(packageName)

- `packageName` {string} 应用包名

相当于 `app.launch(packageName)` 。

## app.getPackageName(appName)

- `appName` {string} 应用名称

获取应用名称对应的已安装的应用的包名，如果该找不到该应用，返回 `null` 。如果该名称对应多个应用，则只返回其中某一个的包名。

该函数也可以作为全局函数使用。

```js
var name = getPackageName("QQ"); //返回"com.tencent.mobileqq"
```

## app.getAppName(packageName)

- `packageName` {string} 应用包名

获取应用包名对应的已安装的应用的名称。如果该找不到该应用，返回null。

该函数也可以作为全局函数使用。

```js
var name = getAppName("com.tencent.mobileqq"); //返回"QQ"
```

## app.openAppSetting(packageName)

- `packageName` {string} 应用包名
- `return` {Boolean}

打开应用的详情页(设置页)。如果找不到该应用，返回false; 否则返回true。

该函数也可以作为全局函数使用。

## app.viewFile(path)

- `path` {string} 文件路径

用其他应用查看文件。文件不存在的情况由查看文件的应用处理。

如果找不出可以查看该文件的应用，则抛出 `ActivityNotException` 。

```js
//查看文本文件
app.viewFile("/sdcard/1.txt");
```

# app.editFile(path)

- `path` {string} 文件路径

用其他应用编辑文件。文件不存在的情况由编辑文件的应用处理。

如果找不出可以编辑该文件的应用，则抛出 `ActivityNotException` 。

```js
//编辑文本文件
app.editFile('/sdcard/1.txt/');
```

# app.uninstall(packageName)

- `packageName` {string} 应用包名

卸载应用。执行后会会弹出卸载应用的提示框。如果该包名的应用未安装，由应用卸载程序处理，可能弹出"未找到应用"的提示。

```js
//卸载QQ
app.uninstall("com.tencent.mobileqq");
```

# app.openUrl(url)

- `url` {string} 网站的Url，如果不以"http://"或"https://"开头则默认是"http://"。

用浏览器打开网站url。

如果没有安装浏览器应用，则抛出 `ActivityNotException` 。

## app.sendEmail(options)

- `options` {Object} 发送邮件的参数。包括:
  - `email` {string} | {Array} 收件人的邮件地址。如果有多个收件人，则用字符串数组表示
  - `cc` {string} | {Array} 抄送收件人的邮件地址。如果有多个抄送收件人，则用字符串数组表示
  - `bcc` {string} | {Array} 密送收件人的邮件地址。如果有多个密送收件人，则用字符串数组表示
  - `subject` {string} 邮件主题(标题)
  - `text` {string} 邮件正文
  - `attachment` {string} 附件的路径。

根据选项 `options` 调用邮箱应用发送邮件。这些选项均是可选的。

如果没有安装邮箱应用，则抛出 `ActivityNotException` 。

```js
//发送邮件给10086@qq.com和10001@qq.com。
app.sendEmail({
    email: ["10086@qq.com", "10001@qq.com"],
    subject: "这是一个邮件标题",
    text: "这是邮件正文"
});
```

## app.startActivity(name)

- `name` {string} 活动名称，可选的值为:
  - `console` 日志界面
  - `settings` 设置界面

启动Auto.js的特定界面。该函数在Auto.js内运行则会打开Auto.js内的界面，在打包应用中运行则会打开打包应用的相应界面。

```js
app.startActivity("console");
```

# 进阶: 意图Intent

Intent(意图) 是一个消息传递对象，您可以使用它从其他应用组件请求操作。尽管 Intent 可以通过多种方式促进组件之间的通信，但其基本用例主要包括以下三个：

- 启动活动(Activity)： Activity 表示应用中的一个"屏幕"。例如应用主入口都是一个Activity，应用的功能通常也以Activity的形式独立，例如微信的主界面、朋友圈、聊天窗口都是不同的Activity。通过将 Intent 传递给 startActivity()，您可以启动新的 Activity 实例。Intent 描述了要启动的 Activity，并携带了任何必要的数据。

- 启动服务(Service)： Service 是一个不使用用户界面而在后台执行操作的组件。通过将 Intent 传递给 startService()，您可以启动服务执行一次性操作（例如，下载文件）。Intent 描述了要启动的服务，并携带了任何必要的数据。

- 传递广播： 广播是任何应用均可接收的消息。系统将针对系统事件（例如：系统启动或设备开始充电时）传递各种广播。通过将 Intent 传递给 sendBroadcast()、sendOrderedBroadcast() 或 sendStickyBroadcast()，您可以将广播传递给其他应用。

本模块提供了构建Intent的函数( `app.intent()` ), 启动Activity的函数 `app.startActivity()` , 发送广播的函数 `app.sendBroadcast()` 。

使用这些方法可以用来方便的调用其他应用。例如直接打开某个QQ号的个人卡片页，打开某个QQ号的聊天窗口等。

```js
var qq = "2732014414";
app.startActivity({
    action: "android.intent.action.VIEW",
    data:"mqq://im/chat?chat_type=wpa&version=1&src_type=web&uin
    packageName: "com.tencent.mobileqq",
});
```

# app.intent(options)

[v4.1.0新增]

- `options` {Object} 选项，包括：
  - `action` {string} 意图的Action，指意图要完成的动作，是一个字符串常量，比如"android.intent.action.SEND"。当action以"android.intent.action"开头时，可以省略前缀，直接用"SEND"代替。参见**Actions**。
  - `type` {string} 意图的MimeType，表示和该意图直接相关的数据的类型，表示比如"text/plain"为纯文本类型。
  - `data` {string} 意图的Data，表示和该意图直接相关的数据，是一个Uri, 可以是文件路径或者Url等。例如要打开一个文件, action为"android.intent.action.VIEW", data为"file:///sdcard/1.txt"。
  - `category` {Array} 意图的类别。比较少用。参见**Categories**。
  - `packageName` {string} 目标包名
  - `className` {string} 目标Activity或Service等组件的名称
  - `extras` {Object} 以键值对构成的这个Intent的Extras(额外信息)。提供该意图的其他信息，例如发送邮件时的邮件标题、邮件正文。参见**Extras**。
  - `flags` {Array} intent的标识，字符串数组，例如 `["activity_new_task", "grant_read_uri_permission"]` 。参见**Flags**。
  - `root` {Boolea} 是否以root权限启动、发送该intent。使用该参数后，不能使用`context.

根据选项，构造一个意图Intent对象。

例如：

```js
//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);
```

需要注意的是，除非应用专门暴露Activity出来，否则在没有root权限的情况下使用intent是无法跳转到特定Activity、应用的特定界面的。例如我们能通过Intent跳转到QQ的分享界面，是因为QQ对外暴露了分享的Activity；而在没有root权限的情况下，我们无法通过intent跳转到QQ的设置界面，因为QQ并没有暴露这个Activity。

但如果有root权限，则在intent的参数加上 `"root": true` 即可。例如使用root权限跳转到Auto.js的设置界面为：

```js
app.startActivity({
    packageName: "org.autojs.autojs",
    className: "org.autojs.autojs.ui.settings.SettingsActivity_"
    root: true
});
```

另外，关于intent的参数如何获取的问题，一些intent是意外发现并且在网络中传播的（例如跳转QQ聊天窗口是因为QQ给网页提供了跳转到客服QQ的方法），如果要自己获取活动的intent的参数，可以通过例如"intent记录"，"隐式启动"等应用拦截内部intent或者查询暴露的intent。其中拦截内部intent需要XPosed框架，或者可以通过反编译等手段获取参数。总之，没有简单直接的方法。

更多信息，请百度**安卓Intent**或参考**Android指南: Intent**。

# app.startActivity(options)

- `options` {Object} 选项

根据选项构造一个Intent，并启动该Activity。

```js
app.startActivity({
    action: "SEND",
    type: "text/plain",
    data: "file:///sdcard/1.txt"
});
```

# app.sendBroadcast(options)

- `options` {Object} 选项

根据选项构造一个Intent，并发送该广播。

# app.startService(options)

- `options` {Object} 选项

根据选项构造一个Intent，并启动该服务。

# app.sendBroadcast(name)

**[v4.1.0新增]**

- `name` {string} 特定的广播名称，包括：
  - `inspect_layout_hierarchy` 布局层次分析
  - `inspect_layout_bounds` 布局范围

发送以上特定名称的广播可以触发Auto.js的布局分析，方便脚本调试。这些广播在Auto.js发送才有效，在打包的脚本上运行将没有任何效果。

```js
app.sendBroadcast("inspect_layout_bounds");
```

# app.intentToShell(options)

**[v4.1.0新增]**

- `options` {Object} 选项

根据选项构造一个Intent，转换为对应的shell的intent命令的参数。

例如:

```js
shell("am start " + app.intentToShell({
    packageName: "org.autojs.autojs",
    className: "org.autojs.autojs.ui.settings.SettingsActivity_"
}), true);
```

参见**intent参数的规范**。

# app.parseUri(uri)

**[v4.1.0新增]**

- `uri` {string} 一个代表Uri的字符串，例如"file:///sdcard/1.txt", "https://www.autojs.org"
- `return` {Uri} 一个代表Uri的对象，参见android.net.Uri。

解析uri字符串并返回相应的Uri对象。即使Uri格式错误，该函数也会返回一个Uri对象，但之后如果访问该对象的scheme, path等值可能因解析失败而返回 `null` 。

需要注意的是，在高版本Android上，由于系统限制直接在Uri暴露文件的绝对路径，因此如果uri字符串是文件 `file://...` ，返回的Uri会是诸如 `content://...` 的形式。

## app.getUriForFile(path)

**[v4.1.0新增]**

- `path` {string} 文件路径，例如"/sdcard/1.txt"
- `return` {Uri} 一个指向该文件的Uri的对象，参见android.net.Uri。

从一个文件路径创建一个uri对象。需要注意的是，在高版本Android上，由于系统限制直接在Uri暴露文件的绝对路径，因此返回的Uri会是诸如 `content://...` 的形式。

上一页
用户界面 - UI
基础

下一页
设备 - Device
基础