Webgum 项目

Android 开发文档

文档版本：V1.0.2

文档日期：2018 年 3 月 6 日

版权声明

【摘要】

关键词：

版本记录

版本编号	版本日期	修改者	说　明
V1.0.1	2018-01-30	李岩	创建
V1.0.2	2018-03-06	李岩	完善 2、3 两项

目　　录

1. 概述及使用

Android 端 webgum 项目主要完成 H5 与原生之间的交互。为了便于各功能各业务之间的解耦，Android 端的 webgum 在设计时采取了主体+插件的设计模式，既存在一个主体用于完成 webgum 最基本的交互。而与设备、网络等设备相关或者与分享、拨打电话、相册、定位等功能相关再或者与获取掌柜信息等项目和业务相关的交互 ,均通过插件的方式实现。在实际使用中 webgum 会提供一些与项目或业务无关的基础插件。在使用中宿主项目可以根据自身的需要选择是否使用这些插件，同样，宿主项目也可以根据自身需求而设计和开发新的插件。

H5 与原生交互的方式 webgum 大致分为了两种，**即时交互**和**回调交互**。

其使用方法（源码引用）大致如下：

1) 引入 webgum 模块，并在配置文件中设置：

将 webgum 模块导入到项目中并在 setting.gradle 配置文件中加入

```
include ':com.ule.webgum'
```

在主体项目的配置文件中引入

```
dependencies {
    compile project(':com.ule.webgum')
}
```

2) 在项目的 Application 类中对 webgum 进行初始化和设置插件

```
private void initWebgum(){
//下面两步应该放在 Application 中操作
Webgum.init(this);
Webgum.setMainPlugin(new MainPlugin());
```

```
Webgum.addPlugins(PluginsConfig.loadPlugins(this));
}
```

3) 在 Activity 中使用 webgum 已经提供的 webview

```
webView = (SystemWebView) findViewById(R.id.acty_sys_wv);
webView.init();
String url = "file:///android_asset/webgum/edit.html";
webView.loadUrl(url);
```

注意：在 activity 中要重写 onActivityResult()方法，便于跳转回置，基本逻辑如下：

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data){
    IReply reply = ReplyManager.getReply(requestCode);
    if(reply != null){
        reply.reply(data);
        ReplyManager.removeReply(reply);
    }
}
```

4) 使用自定义 webview

2. 主体设计及扩展

实际上，在设计 Android 端 webgum 时，也将主体设计成为了一个插件，既主体插件，主体插件在 webgum 初始化的时候就被添加进来，而且无法将主体插件删除。这样设计主要是为后期的扩展预留一定的空间，因为在以后有可能不同的宿主会要求 webgum 主体具有不同的功能，而这些功能放入其他插件又不合理。

在 webgum 的 webview 中，会注入一个 js 对象 `wg_android_native`。该对象主要用来管理和分发 H5 与原生交互的各种请求，鉴于此，它的实现必须由 Android 原生代码完成。

由于不同的浏览器内核各自的注入 js 对象的方式不尽相同，该对象实现交互的请求分发的方式也略有差别，所以针对不同浏览器会有不同的实现。目前 Android 端的 webgum 实现了 webview 和 crosswalk 两个内核的注入对象。它们暴露给 js 端使用的对象都是 `wg_android_native`，但各自的实现有所区别。也就是说它的存在主要还是为了让主体和插件不会因为浏览器内核的不同而不能通用，既不管是何种浏览器，只需要开发一个主体和插件都能正常使用。

js 层会分别通过调用该对象的 **`onJsCallWithListener()`**方法和 **`onJsCallWithResult()`** 方法来完成对有回调和无回调的请求。`wg_android_native` 通过这两个方法来完成分发，在分发时会对主体做特殊调用。

webgum 会带有一个默认的主体实现供宿主使用，当默认主体无法满足宿主需要时，宿主也可以自定义主体，并在 webgum 初始化的时候将其设置成为主体。以此来完成对主体的扩展。以下是一个扩展主体的示例：

```
final public class MainPlugin extends MainPluginAbs{


    public static final String WEBGUM_VERSION = "1.0.0";
    public static final String WEBGUM_OS = "Android";
    public static final String PLUGIN_VERSION = "1.0.0";


    public MainPlugin() {
        super(PLUGIN_VERSION);
    }


    @Override
    public String getOsInfo(JSRequest request) {
        JSResult<String> jsResult = new JSResult<>();
```

```
        jsResult.put("osName", WEBGUM_OS);
        jsResult.put("osVersion", SystemTool.getSystemVersion());
        jsResult.put("brand", SystemTool.getBrand());
        jsResult.put("model", SystemTool.getModel());
        jsResult.put("wgVersion", WEBGUM_VERSION);
        return jsResult.toJsonString();
    }

    @Override
    public String getPlugins(JSRequest request) {

        Log.e("MainPlugin","getPlugins");

        JSResult result = new JSResult();

        List<String> list = new ArrayList<>();

        for(IWebgumPlugin plugin :
request.webgumView.getPluginManager().pluginMap.values()){
            if(!"main".equals(plugin.getName())){
                list.add(plugin.getName());
            }

        }
        result.put("plugins",list);
        return result.toJsonString();
    }


    //这是一个扩展的主体方法
    public void testCall(JSRequest request, JsResponse response){
        JSRequest.Parameter parameter = request.getParam(0);
        JSResult result = new JSResult();
        result.put("name","testCall");
        response.send(parameter, result);
    }

}
```

3. 插件开发及管理

在**开发插件**时：

首先，每一个插件都要继承自 `IWebgumPlugin` 抽象类。为了便于对插件的管理和维护，每一个插件都有一个插件名和插件版本。在对插件进行命名时要严禁名称重复。

其次，在为插件添加方法时，需要在方法上添加`@JSMethod()`注解，只有添加了这个注解的方法才能被 H5 层调用到，该注解可以传递一个参数 name，既供 H5 调用时的方法名称，如果不传，默认与插件本身的方法名一致。如下示例：

```
@JSMethod(name="open")
public void openAlbum(JSRequest request, JsResponse response){
    ...
}
```

以上示例是 camera 插件中打开相册的方法，在 H5 页面上调用时代码为：

```
wg.camera.open(function(data){
    ...
})
```

如果在注解中没有传递 name 参数，则调用如下：

```
wg.camera.openAlbum(function(data){
    ...
})
```

再次，方法必须是 `public` 修饰，当方法为**即时交互**时，方法必须有且只能有一个 `JSRequest` 类型的参数，返回值也必须为 `String` 类型；当方法为**回调交互**时方法必须有 `JSRequest` 类型和 `JsResponse` 类型的参数，没有返回值。既，插件的方法分为两类，其规定格式如下：

```
//即时交互
```

```
@JSMethod([name="method"])
public String methodName(JSRequest request){
    ...
}

//回调交互
@JSMethod([name="method"])
public void methodName(JSRequest request, JsResponse response){
    ...
}
```

最后，在方法中需要获取 H5 端传递过来的参数，可以通过 `request.getParam(index)` 方法获取，需要传递一个 `int` 参数，既获取第几个参数，返回值为 `Parameter` 类型的对象，当取不到对应位置的参数时返回 `null`。在**回调交互**中，如果需要回调 js，在回调前，需要先构造一个 `JSResult` 类型的对象，并将返回结果封装在里面，通过 `response.send(param, result)` 方法回调 js。`send` 方法需要两个参数，既 `Parameter` 对象和 `JSResult` 对象。

以下是一个完整的插件示例：

```
public class CameraPlugin extends IWebgumPlugin {



    public CameraPlugin(String pluginName, String pluginVersion){
        super(pluginName, pluginVersion);
    }

    /**
     * 打开相册
     * @param request
     * @param response
     */
    @JSMethod
    public void openAlbum(JSRequest request, JsResponse response){
        Intent openAlbumIntent = new Intent("android.intent.action.GET_CONTENT");
```

```
        openAlbumIntent.setDataAndType(MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
"image/*");
        Activity acty = (Activity)request.webgumView.getContext();
        AlbumReply albumReply = new AlbumReply(acty,request,response);
        ReplyManager.addReply(albumReply);
        acty.startActivityForResult(openAlbumIntent, albumReply.getId());
    }

    /**
     * 打开摄像头
     * @param request
     * @param response
     */
    @JSMethod
    public void openCamera(JSRequest request, JsResponse response){
        Intent openCameraIntent = new Intent("android.media.action.IMAGE_CAPTURE");
        Activity acty = (Activity)request.webgumView.getContext();
        CameraReply cameraReply = new CameraReply(acty,request,response);
        ReplyManager.addReply(cameraReply);
        openCameraIntent.putExtra("output", cameraReply.getImgUri());
        acty.startActivityForResult(openCameraIntent, cameraReply.getId());
    }

}
```

在 CameraReply 类中的处理如下：

```
public class CameraReply implements IReply {

    private Activity activity;
    private Uri imgUri;
    private int id = IDFactory.generateID();
    private JsResponse response;
    private JSRequest request;

    public CameraReply(Activity activity,JSRequest request, JsResponse response){
        this.activity = activity;
        this.request = request;
        this.response = response;
        File imageStorageDir = new
File(Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DCIM),
```

```
"YZGHD");
            if (! imageStorageDir.exists()){
                imageStorageDir.mkdirs();
            }
            File file = new File(imageStorageDir + File.separator +
String.valueOf(System.currentTimeMillis()) + ".jpg");
            imgUri = Uri.fromFile(file);
        }

        @Override
        public int getId() {
            return id;
        }

        public Uri getImgUri(){
            return imgUri;
        }

        @Override
        public void reply(final Intent data) {

            new Thread(new Runnable() {
                @Override
                public void run() {
                    File file = new File(getImgUri().getPath());
                    JSRequest.Parameter param = request.getParam(0);
                    try{
                        if (file.exists()) {
                            Bitmap bitmap = ImageTool.getBitmap(file.getPath());
                            bitmap = ImageTool.compressAndGenImage(bitmap,500);
                            String imageBase64 = ImageTool.bitmapToBase64(bitmap,50);
                            imageBase64 = imageBase64.replaceAll("\n","");
                            JSResult result = new JSResult();
                            result.put("imagePath", getImgUri().getPath());
                            result.put("imageBase64", imageBase64);
                            response.send(param, result);
                        } else {
                            response.send(param, new JSResult(false,"9990","用户取消"));
                        }
                    }catch (Exception e){
```

```
                    response.send(param, new JSResult(false,"9999","处理异常"));
                }
            }
        }).start();
    }
}
```

补充，主体插件作为一个特殊的插件 ,其提供给 H5 调用的方法除了不需要`@JSMethod()`注解之外，其他几点规则也必须遵循。`wg_android_native` 在调用主体插件的方法时，不会检查时候有`@JSMethod()`注解，这也就意味着在 H5 调用主体方法时，必须与原生方法名一致。如在上述 `MainPlugin` 类示例中，H5 的对 `getOsInfo()`方法的调用代码如下：

```
var os_info = wg.getOsInfo();
```

另外， H5 在调用主体插件时，只需要通过 `wg.methodName()`即可，而调用插件时则需要通过 `wg.pluginName.method()`的方式。

在**管理插件**时：

webgum 并没有暴露太多关于插件管理的方法，只在其初始化时提供了 `setMainPlugin(MainPluginAbs mainPlugin)`方法用于设置主体，`addPlugin(IWebgumPlugin plugin)`方法和 `addPlugins(List<IWebgumPlugin> plugins)` 方法用于添加一个或一组插件。而在 webgum 内部则存在一个 `PluginManager` 类做为插件管理器来管理插件，原则上对于宿主项目来说不需要关心 `PluginManager` 类，也无需对其做任何的直接操作。但是如果有特殊情况需要对其做直接操作的话，也可以通过 `request.webgumView.getPluginManager()`获取。这里再次重申，每一个插件都必须继承自 `IWebgumPlugin` 类 ,并提供 **`pluginName`** 和 **`pluginVersion`** ,而且 **`pluginName`** 不能重复。