**Интеграция Java плагина в среду разработки Unity с помощью AndroidJavaObject и AndroidJavaClass**

**Горбунов Никита Алексеевич**

**Аннотация.** В целях освоения новых инструментов разработки под операционную систему Android на мультиплатформенном движке Unity и дальнейшего их правильного применения в оной среде, автором были проанализированы и протестированы встроенные возможности интеграции языка программирования Java в среду разработки проекта. При написании Android приложений, довольно часто приходиться взаимодействовать с Android API. Большинство базовых используемых методов Android API уже встроено в инструментарий движка Unity, но в редких случаях его недостаточно, поэтому разработчики встроили возможность интеграции «JAR» и «AAR» плагинов с помощью классов «AndroidJavaObject» и «AndroidJavaClass», которые дают возможность вызова методов из скомпилированного файла. Для написания правильно работающего плагина, нужно придерживаться определенного паттерна проектирования, изменить и настроить Android Manifest при компиляции JAR или AAR файла и правильно использовать классы из метаданных Unity.

**Ключевые слова:** Разработка, Unity, Android Application on Unity, Android Plugin for Unity, Java in Unity, JAR & AAR, Android API in Unity, Добавление Java плагина в Unity, Unity Activity.

**Для цитирования:** Горбунов Н.А. Интеграция Java плагина в среду разработки Unity с помощью AndroidJavaObject и AndroidJavaClass. 2023.

Каждый разработчик сталкивался с задачей взаимодействия приложения и операционной системы, под которую оно компилируется.

Unity – кроссплатформенная среда разработки, с помощью которой можно создавать игры и приложения под разные платформы, но большая часть разработанных приложений на этом движке реализованы именно под операционную систему Android. Сам же Unity разработан на языке C#, поэтому многие не опытные разработчики могут задаваться вопросом – «Как же взаимодействовать с системными вызовами?». Большинство основных системных вызовов уже реализованы «под капотом» игрового движка, например: создание/удаление/запись файлов, запрос на разрешения приложения, геолокация, интернет запросы и так далее. Для написания простого приложения, которое должно будет сохранять что-либо или отправлять запросы на сервер встроенного инструментария хватит, но если приложение должно работать с Bluetooth устройствами, сервисами, ИК-портом (если присутствует в телефоне), то встроенных методов Unity недостаточно. Разработчики движка спрогнозировали это и дали возможность интеграции Java в приложение. Код написанный на Java можно импортировать только в уже скомпилированном файле с расширением AAR или JAR, а взаимодействие с этими «плагинами» будет осуществляться через классы «AndroidJavaClass» и «AndroidJavaObject». «AndroidJavaClass» — это представление Unity универсального экземпляра «java.lang.Class», а «AndroidJavaObject» – «java.lang.Object». Его можно использовать как бестиповый интерфейс для экземпляра любого класса Java внутри C# скрипта.

При написании кода на языке Java у вас открывается максимальный спектр возможностей взаимодействия с операционной системой Android. Но нужно придерживаться определенных правил, если код, написанный вами, используется лишь, как библиотека, к которой идёт обращение из основного приложения.

Первым делом нужно разобраться, как компилировать код на Java. При компиляции библиотеки под OC Windows, и дальнейшего взаимодействия с ней, на выходе получается файл с расширением DLL. При компиляции под ОС Android возможны два вида файлов:

* JAR – Java Archive. Содержит файлы классов, JAVA-ресурсы, зависимые библиотеки, и другие необходимые для приложения файлы. JAR может использоваться как в Android-приложении, так и в приложении на чистой Java.
* AAR – Android Archive. Так же, как и JAR является zip-архивом, но может использоваться только в Android-приложении. AAR содержит скомпилированный исходный код в файле classes.jar, Android -ресурсы, AndroidManifest, и другие файлы, входящие в состав APK.

При компиляции библиотеки для интеграции в Unity приложение подходят оба архива, но нужно учитывать особенности. В документации Unity разработчики советуют использовать AAR плагины из-за большей совместимости с движком. Если использовать API, которые требуют какие-либо разрешения, их нужно прописывать в «AndroidManifest.xml». Сторонние библиотеки добавляются в «build.Gradle». Если использовать AAR файл, то при компиляции не обязательно указывать все нужные ресурсы в плагине, достаточно будет добавить их в манифест вашего основного приложения, а движок сам сольёт всё в один Gradle файл. Также Unity не требует добавлять «3rd-party» в определенную папку, их можно будет расположить в любом удобном месте вашего приложения. JAR файл в свою очередь обязывает прописывать все разрешения в манифесте при сборке, а библиотеки обязательно должны лежать в папке «Assets/Plugins/Android».

Прежде чем приступать к написанию кода, нужно решить, какие задачи будет решать плагин, написанный вами на Java. Если вам нужно управлять уведомлениями, вызывать «AlertDialog» или «Pop-up» окна, то нужно будет обязательно взаимодействовать с Android Activity вашего приложения. Android Activity – ключевой компонент для создания визуализации интерфейса, форма приложения с которой взаимодействует пользователь. Активность вашего приложения нужно будет передать, как параметр в методе инициализации плагина из скрипта C#. Далее идет взаимодействие с базовыми функциями приложения, например: доступ к файловой системе, запрос на разрешения, доступ к WIFI, Bluetooth, USB. Вам нужно будет обращаться к контексту активности. Context – Интерфейс для глобальной информации о среде приложения. Это абстрактный класс, реализация которого обеспечивается системой Android. Он обеспечивает доступ к ресурсам и классам, специфичным для приложения, а также вызовы для операций уровня приложения, таких как запуск действий, широковещательная рассылка, получение намерений (Intent) и так далее. И для опытных разработчиков есть возможность переопределения активности основного приложения. «UnityPlayerActivity» Unity Android приложения отвечает за базовое взаимодействие между операционной системой Android и приложением. Вы можете использовать плагины, чтобы создать свою собственную активность, которая расширяет и переопределяет стандартную «UnityPlayerActivity», например для старта каких-либо сервисов после блокировки экрана телефона.

После конкретизации поставленной задачи нужно определиться с архитектурой написания плагина.

* Если вам важна точность исполнения всех вызовов плагина, сохранение состояний объектов и возврат результата, например постоянное обращение к файловой системе или отправка данных по Bluetooth с сохранением подключения к устройству, то стоит использовать паттерн проектирования Singleton.
* Если вам не важно сохранение состояния, и нужно лишь иногда обращаться к Android API, например вывести pop-up об какой-либо ошибке или вести «LogCat» с пользовательскими тегами, то достаточно будет реализовать всё в статических методах.
* Если нужно единожды вызвать API, которая не отвечает за важные решения операционной системы, то Java плагин создавать не обязательно. Можно объявить и инициализировать какой-либо класс Android API в классе «AndroidJavaClass», вызвать нужный метод, после чего Garbage collector очистит выделенную память. Этот способ самый не оптимизированный, но быстрый.

1. Примечание – выбор архитектуры зависит от API, которые вы будете использовать.

Наконец можно приступить к интеграции плагина в C# скрипты. Для начала нужно выбрать какого типа будет переменная, которая хранит в себе библиотеку. Если вы решили использовать паттерн Singleton, то вам нужно использовать «AndroidJavaObject», так как нам нужно хранить экземпляр класса и вызывать его методы. Иначе, если вам достаточно вызывать только статические методы класса, используйте «AndroidJavaClass».

2. Примечание – во втором случае можно использовать «AndroidJavaObject», но это логически будет не верно.

Далее, если у вас присутствует метод инициализации в плагине, нужно его вызвать передав в аргументах активность. Чтобы это сделать используется конструктор «AndroidJavaClass», который в аргументах принимает строку с названием пространства имён, которое нужно использовать.

1. Инициализируем Unity Activity (Если вы её не переопределили).
2. Объявляем переменную, в которой будет наш плагин.
3. Инициализируем плагин, передавая в параметры активность нашего приложения.
4. Возвращаем экземпляр класса нашей библиотеки в отдельную переменную.

Пример кода:

**private** AndroidJavaClass unityClass;

**private** AndroidJavaObject unityActivity;

**private** AndroidJavaClass customClass;

**private** AndroidJavaObject instance;

**private** **const** **string** PackageName = “com.myPlugin.plugin.Manager”;

**private** **const** **string** UnityDefaultJavaClassName = “com.unity3d.player.UnityPlayer”;

**public** **void** **Init**()

{

AndroidJNI.AttachCurrentThread();

unityClass = **new** AndroidJavaClass(UnityDefaultJavaClassName);

unityActivity = unityClass.GetStatic<AndroidJavaObject>(“currentActivity”);

customClass = **new** AndroidJavaClass(PackageName);

customClass.CallStatic(“InitInstance”, unityActivity);

instance = customClass.GetStatic<AndroidJavaObject>(“instance”);

}

3. Примечание – Метод «AndroidJNI.AttachCurrentThread()» нужно использовать при каждом вызове методов из плагина, если скрипт работает в основном потоке.

Все методы плагина будут вызываться с помощью «Call» и «CallStatic», для обычных и static методов соответственно, которые принимают 2 параметра:

1. Имя метода из плагина (типом string)
2. Массив базовых классов object

При передаче параметров, для наибольшей производительности, следует использовать простые типы, такие как: int, float, string.

В результате проделанной работы на выходе получается рабочий Java плагин, который был успешно интегрирован в проект. Для его реализации понадобилось:

* Настроить сборку Java кода в AAR файл.
* Проанализировать задачи, которые должна реализовывать библиотека.
* Исходя из задач, построить верную структуру программы.
* Интегрировать плагин в среду проекта и инициализировать его в скриптах.
* Использовать методы через методы «Call» и «CallStatic», передавая базовые типы, чтобы не нагружать приложение.

Список литературы.

1. <https://docs.unity3d.com/ScriptReference/>
2. <https://habr.com/ru/companies/pixonic/articles/353444/>