WebView

**WebView** — это компонент, который позволяет встраивать веб-страницы в приложения, своеобразный мини-браузер. Находится в разделе **Containers**.

В старых версиях Android **WebView** использовал движок **WebKit**. В Android 4.4 он стал использовать движок **Chromium** или **Blink**. В Android 5 появилось отдельное приложение **System WebView**, которое можно скачать из Play Market. Такое разделение позволило обновлять движок без обновления системы. На этом приключения не закончились. В Android 7.0 уже используется движок **Chrome**, а если этого браузера на устройстве нет, то используется **System WebView**. Подобные выкрутасы не прошли даром, программисты часто жалуются, что какой-то кусок кода не работает. Приходится проверять работу на разных устройствах. Общий принцип работы сохранился.

Надеюсь, вы уже познакомились с [базовым примером по созданию собственного браузера](http://developer.alexanderklimov.ru/android/mybrowser.php). Рассмотрим дополнительные возможности элемента **WebView**.

Загружаем локальные страницы и картинки

Если вы хотите загружать в **WebView** страницы не из интернета, а со своего приложения, то разместите нужные файлы в папке [assets](http://developer.alexanderklimov.ru/android/theory/assets.php), например, **assets/mypage.html**. Доступ к файлу вы можете получить через конструкцию **file://android\_asset**:

myBrowser = (WebView)findViewById(R.id.mybrowser);

myBrowser.loadUrl("file:///android\_asset/mypage.html");

Аналогично поступаем с картинками, которые встречаются в html-файле

<img src="file:///android\_asset/myimage.gif">

Также можно загрузить файл из папки **res/raw**:

webView.loadUrl("file:///android\_res/raw/cat.html");

Если картинка находится на внешнем накопителе, то попробуйте вариант:

WebView webView = (WebView) findViewById(R.id.webView);

String imageName = "cutecat.png";

String catUrl = "file://"

+ Environment.getExternalStorageDirectory().getAbsolutePath()

.toString() + "/" + imageName;

webView.loadUrl(catUrl);

Загружаем данные при помощи loadData() и loadDataWithBaseURL()

Данные можно загрузить с помощью метода **loadData()**:

String htmlText = "<html><body>Percent test: 100% </body></html>";

WebView webView = (WebView) findViewById(R.id.webView);

webView.loadData(htmlText, "text/html", "en\_US");

Если текст простой, то этот способ подойдёт. Но в данном примере встречается символ процента, который относится к спецсимволам и часть текста может оказаться недоступной. Если в тексте встречаются подобные символы, то лучше использовать метод **loadDataWithBaseURL()**:

webView.loadDataWithBaseURL(null, htmlText, "text/html", "en\_US", null);

Если вам приходится использовать **loadData()**, то спецсимволы можно заменить при помощи метода **replace()**:

String webData = stringBuffer.toString(); // поступающие данные

webData = webData.replace("#", "%23");

webData = webData.replace("%", "%25");

webData = webData.replace("\\", "%27");

webData = webData.replace("?", "%3f");

webView.loadData(webData, "text/html", "UTF-8");

Проблемы с кодировкой

У меня есть программа в Google Play, использующая **WebView**. К моему удивлению, некоторые пользователи жаловались, что текст нечитаем, так как они видят только кракозябры. Особенно много жалоб было от пользователей с планшетами. Оказалось, что проблема довольна распространённая и обсуждается на форумах. Танцы с бубнами (установка явной кодировки UTF-8) не помогают. Нашёл один ответ, который у некоторых заработал, на всякий случай я его здесь оставлю.

// перед загрузкой данных (load...)

WebSettings settings = mWebView.getSettings();

settings.setDefaultTextEncodingName("utf-8");

Но я рекомендую просто использовать метод **loadDataWithBaseURL()**. Работает стабильно.

Методы

У **WebView** есть множество методов, которые позволяют добиваться полной функциональности как у обычного браузера - обновить страницу, перейти на предыдущую страницу и т.д. Часть методов представлена ниже:

* reload()
* goForward()
* goBack()

Используем зум для просмотра

Не забывайте, что **WebView** можно использовать не только для просмотра html-страниц, но и для просмотра изображений. Поэтому данный компонент вполне можно использовать как вьювер картинок, к тому же вы можете включить встроенный механизм масштабирования:

mWebView = (WebView) findViewById(R.id.webView1);

// устанавливаем Zoom control

mWebView.getSettings().setBuiltInZoomControls(true);

// загружаем картинку (не забудьте установить разрешение на интернет)

mWebView.loadUrl("http://netsources.narod.ru/friday/alkocat.jpg");

this.setTitle("WebView");



Прозрачность

Устанавливать прозрачность лучше программно. Встречал жалобы, что через XML это свойство не работает.

webView.setBackgroundColor(0x00000000);

WebView в Lollipop

В Android 5.0 компонент доступен в Google Play ([Android System WebView](https://play.google.com/store/apps/details?id=com.google.android.webview)) и его можно обновлять на устройстве.

Компонент теперь основывается на движке Chromium и поддерживает следующие новинки.

* WebRTC
* WebAudio
* WebGL

Можно ознакомиться с некоторыми примерами - [GoogleChrome/chromium-webview-samples](https://github.com/GoogleChrome/chromium-webview-samples). Там есть примеры с WebRTC, полноэкранным режимом, касаниями экрана, выбора файла, работой с JavaScript-сценариями.

Кроме того, стал доступен **Safe Browsing** - механизм, предупреждающий об опасных ссылках. Включается через манифест.

<manifest>

<meta-data android:name="android.webkit.WebView.EnableSafeBrowsing"

android:value="true" />

. . .

<application> . . . </application>

</manifest>

Советы

Фон

Если вы заметили, что экран мерцает во время загрузки **WebView**, то поменяйте фон. Мерцание происходит из-за смены фона приложения (темы), на белый фон по умолчанию для **WebView**, а потом на фон, который прописан на странице.

mWebView.setBackgroundColor(Color.parseColor("#3498db"));

mWebView.setBackgroundColor(getResources().getColor(R.color.my\_color\_name));

// и т.п.

Касания экрана

Так как поддерживаются касания экрана, то старайтесь использовать на веб-странице визуальные эффекты нажатия кнопок и других элементов при помощи псевдокласса **:active**, например, так:

.btn {

display: inline-block;

position: relative;

background-color: #f39c12;

padding: 14px;

border-radius: 5px;

border-bottom-style: solid;

border-width: 4px;

border-color: #DA8300;

}

.btn:active {

background-color: #E68F05;

border-color: #CD7600;

border-width: 2px;

top: 2px;

}

Настройки

В API 24 появилась возможность открыть окно настроек и выбрать движок для **WebView**:

Intent intent = new Intent(Settings.ACTION\_WEBVIEW\_SETTINGS);

if (intent.resolveActivity(getPackageManager()) != null) {

startActivity(intent);

}

# WebView - создай свой браузер

Android позволяет создать собственное окно для просмотра веб-страниц или даже создать свой клон браузера при помощи элемента **WebView**. Сам элемент использует движок WebKit и имеет множество свойств и методов. Мы ограничимся базовым примером создания приложения, с помощью которого сможем просматривать страницы в интернете. В последних версиях используется движок от Chromium, но большой разницы в этом нет для простых задач.

Создадим новый проект **MyBrowser** и сразу заменим код в файле разметки **res/layout/activity\_main.xml**:

<?xml version="1.0" encoding="utf-8"?>

<WebView

android:id="@+id/webView"

xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"/>

Теперь откроем файл активности **MainActivity.java** и объявим компонент **WebView**, а также инициализируем его - включим поддержку JavaScript и укажем страницу для загрузки.

private WebView mWebView;

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

mWebView = (WebView) findViewById(R.id.webView);

// включаем поддержку JavaScript

mWebView.getSettings().setJavaScriptEnabled(true);

// указываем страницу загрузки

mWebView.loadUrl("http://developer.alexanderklimov.ru/android");

}

Так как приложение будет использовать интернет, необходимо установить разрешение на доступ к интернету в файле-манифесте.

<uses-permission android:name="android.permission.INTERNET" />

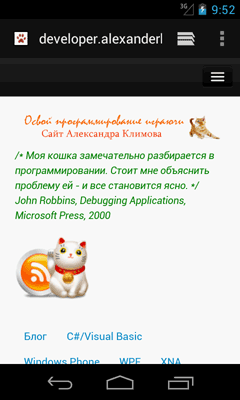
Там же в манифесте модифицируем строчку для экрана, удалив заголовок из нашего приложения (выделено жирным):

<activity android:name=".HelloWebViewActivity"

android:label="@string/app\_name"

**android:theme="@style/Theme.AppCompat.NoActionBar"**>

Запустим приложение. В нашем распоряжении появился простейший вьювер веб-страниц, но с одним недостатком. Если вы щёлкнете на любой ссылке, то у вас автоматически запустится браузер по умолчанию и новая страница отобразится уже там.



Upd. Сейчас я обнаружил, что теперь даже при запуске приложения сразу открывается браузер. Раньше такого не было.

Чтобы решить данную проблему и открывать ссылки в своей программе, нужно переопределить класс **WebViewClient** и позволить нашему приложению обрабатывать ссылки. Добавим в коде вложенный класс:

private class MyWebViewClient extends WebViewClient

{

@Override

public boolean shouldOverrideUrlLoading(WebView view, String url)

{

view.loadUrl(url);

return true;

}

}

Затем в методе **onCreate()** определим экземпляр **MyWebViewClient**. Он может находиться в любом месте после инициализации объекта **WebView**:

mWebView.setWebViewClient(new MyWebViewClient());

Теперь в нашем приложении создан **WebViewClient**, который позволяет загружать любой указанный URL, выбранный в **WebView**, в сам контейнер **WebView**, а не запускать браузер. За данную функциональность отвечает метод **shouldOverrideUrlLoading(WebView, String)**, в котором мы указываем текущий **WebView** и нужный URL. Возвращаемое значение *true* говорит о том, что мы не нуждаемся в запуске стороннего браузера, а самостоятельно загрузим контент по ссылке.

Повторно запустите программу и убедитесь, что ссылки загружаются теперь в самом приложении. Но теперь возникла ещё одна проблема. Мы не можем вернуться к предыдущей странице. Если мы нажмём на кнопку BACK (Назад) на своем устройстве, то просто закроем свое приложение. Для решения новой проблемы нам необходимо обрабатывать нажатие кнопки BACK. Добавляем новый метод:

@Override

public void onBackPressed() {

if(mWebView.canGoBack()) {

mWebView.goBack();

} else {

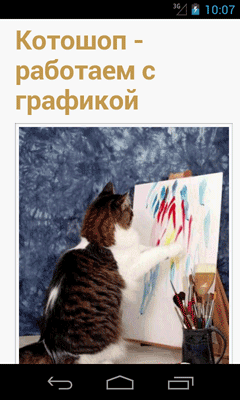
super.onBackPressed();

}

}

Мы должны проверить, что **WebView** поддерживает навигацию на предыдущую страницу. Если условие верно, тогда вызывается метод **goBack()**, который возвращает нас на предыдущую страницу на один шаг назад. Если таких страниц набралось несколько, то мы можем последовательно вернуться к самой первой странице. При этом метод всегда будет возвращать значение *true*. Когда мы вернёмся на самую первую страницу, с которой начали путешествие по интернету, то вернётся значение *false* и обработкой нажатия кнопки BACK займётся уже сама система, которая закроет экран приложения.

Запустите приложение ещё раз. У вас появился свой собственный веб-браузер, позволяющий ходить по ссылкам и возвращаться на предыдущую страницу. Изучив документацию, вы можете оснастить приложение и другим вкусными плюшками для своего браузера.



Если вам нужно часть ссылок, ведущих на ваш сайт открывать в браузере, а локальные ссылки открывать в приложении, то применяйте условие с разными возвращаемыми значениями.

public class MyWebViewClient extends WebViewClient {

@Override

public boolean shouldOverrideUrlLoading(WebView view, String url) {

if(Uri.parse(url).getHost().endsWith("developer.alexanderklimov.ru")) {

return false;

}

Intent intent = new Intent(Intent.ACTION\_VIEW, Uri.parse(url));

view.getContext().startActivity(intent);

return true;

}

}

Универсальный метод, который все локальные ссылки откроет в приложении, остальные в браузере (меняем одну строчку):

public class MyAppWebViewClient extends WebViewClient {

@Override

public boolean shouldOverrideUrlLoading(WebView view, String url) {

**if(Uri.parse(url).getHost().length() == 0)** {

return false;

}

Intent intent = new Intent(Intent.ACTION\_VIEW, Uri.parse(url));

view.getContext().startActivity(intent);

return true;

}

}

А сейчас немного усложним пример, чтобы у пользователя появилась альтернатива стандартным браузерам.

Чтобы было понятнее, переделаем пример следующим образом. Создайте две активности. На первой активности разместите кнопку для перехода на вторую активность, а на второй активности разместите компонент **WebView**.

В манифесте прописываем у второй активности фильтр.

<activity

android:name=".SecondActivity"

android:label="@string/title\_activity\_second" >

**<intent-filter>**

**<action android:name="android.intent.action.VIEW" />**

**<action android:name="ru.alexanderklimov.Browser" />**

**<category android:name="android.intent.category.DEFAULT" />**

**<data android:scheme="http" />**

**</intent-filter>**

</activity>

Код для кнопки для перехода на вторую активность.

public void onClick(View view) {

Intent intent = new

Intent("ru.alexanderklimov.Browser");

intent.setData(Uri.parse("http://developer.alexanderklimov.ru/android/"));

startActivity(intent);

}

Мы создали собственное намерение с указанием фильтра и предоставили данные - адрес сайта.

Вторая активность должна принять данные:

package ru.alexanderklimov.testapplication;

import android.net.Uri;

import android.support.v7.app.AppCompatActivity;

import android.os.Bundle;

import android.webkit.WebView;

import android.webkit.WebViewClient;

public class SecondActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_second);

**Uri url = getIntent().getData();**

WebView webView = (WebView) findViewById(R.id.webView);

webView.setWebViewClient(new Callback());

webView.loadUrl(url.toString());

}

private class Callback extends WebViewClient {

@Override

public boolean shouldOverrideUrlLoading

(WebView view, String url) {

return(false);

}

}

}

В фильтре для второй активность мы указали два действия.

<action android:name="android.intent.action.VIEW" />

<action android:name="ru.alexanderklimov.Browser" />

Это означает, что любые активности (читай, приложения) могут вызвать вашу активность с мини-браузером по такому же принципу. Запустите в студии в отдельном окне любой старый проект или создайте новый и добавьте в него кнопку и пропишите тот же код, который мы использовали для щелчка кнопки.

Запустите второе приложение (первое приложение можно закрыть) и нажмите на кнопку. У вас запустится не первое приложение с начальным экраном, а сразу вторая активность с мини-браузером. Таким образом, любое приложение может запустить браузер, не зная имени класса вашей активности, а используя только строку **"ru.alexanderklimov.Browser"**, передаваемую в **Intent**. При этом ваша активность с браузером должна иметь категорию по умолчанию и данные. Напомню:

<category android:name="android.intent.category.DEFAULT" />

<data android:scheme="http" />

Вы можете представить свою строку в виде строковой константы и сообщить всем потенциальным пользователям вашего браузера, как они могут запустить его у себя. Но в Android уже есть такая готовая константа **ACTION\_VIEW**, которая по справке документации представляет собой следующее:

public static final java.lang.String ACTION\_VIEW = "android.intent.action.VIEW";

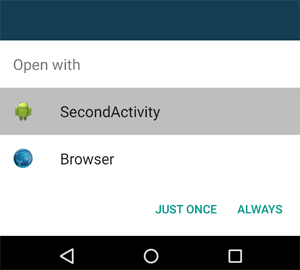
Перепишем код для кнопки у второго приложения

Intent(android.content.Intent.ACTION\_VIEW,

Uri.parse("http://developer.alexanderklimov.ru/android/"));

startActivity(intent);

Что произойдёт на этот раз? Мы помним, что у нас прописано два действия, включая и **android.intent.action.VIEW**. А значит наше первое приложение с браузером тоже должно распознавать эту команду, когда какое-то приложение у пользователя использует этот код. На эмуляторе как минимум есть одна такая программа "Browser", и теперь к ней добавилась наша вторая активность из первого приложения. На экране появится выбор из двух приложений.



А если удалить все альтернативные браузеры и оставить только вашу программу, то и выбора не будет. Ваш браузер станет основным. И если какое-то приложение захочет запустить веб-страницу указанным способом, то откроется ваша программа.

Небольшое замечание. Если заменить последнюю строчку на такую:

startActivity(Intent.createChooser(intent, "Мяу..."));

То в окне выбора программы вместо верхней строки "Open with" или её локального перевода появится ваша строка. Но не это главное. Если по каким-то причинам на устройстве не окажется ни одного браузера, то данный вариант кода не вызовет краха приложения, в отличие от первоначального варианта. Поэтому используйте предложенный вариант ради надёжности.