/ Моя кошка замечательно разбирается в программировании. Стоит мне объяснить проблему ей - и все становится ясно. */*
John Robbins, Debugging Applications, Microsoft Press, 2000

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.

Надеюсь, вы уже познакомились с базовым примером по созданию собственного браузера. Рассмотрим дополнительные возможности элемента **WebView**.

и картинки

Если вы хотите загружать в **WebView** страницы не из интернета, а со своего приложения, то разместите нужные файлы в папке assets, например, **assets/mypage.html**. Доступ к файлу вы можете получить через конструкцию **file://android_asset**:

```
webView = findViewById(R.id.mybrowser);

webView.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 = findViewById(R.id.webView);
String imageName = "cutecat.png";

String catUrl = "file://"
        +
Environment.getExternalStorageDirectory().getAbsolutePath()
                .toString() + "/" + imageName;
webView.loadUrl(catUrl);
```

Недавно наткнулся на фрагмент кода, где нужно добавить несколько новых настроек для работы с файлами. Пример для Kotlin.

```
val webView = findViewById(R.id.webView)
// Enable the WebView to access content through file: URLs
webView.settings.apply {
    allowFileAccess = true
    allowFileAccessFromFileURLs = true
    allowUniversalAccessFromFileURLs = true
}
```

loadData() и loadDataWithBaseURL()

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

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

```
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 просмотра картинок с котиками, к тому же вы можете включить встроенный механизм масштабирования:

> При работе с протоколом http используйте совет Cleartext HTTP traffic not permitted (https)

```
mWebView = 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) и его можно обновлять на у

- WebRTC
- WebAudio
- WebGL

Можно ознакомиться с некоторыми примерами - 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_
// 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:

```
  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);
}
```

Ночной режим

Появилась поддержка тёмной темы в последних версиях WebView.

```
implementation "androidx.webkit:webkit:1.2.0-alpha01"
```

За ночной режим отвечает класс **WebViewFeature**, который имеет в своём составе коллекцию различных возможностей. Проверить поддержку той или иной возможности можно через **isFeatureSupported()**.

```
// Поддерживается ли ночной режим
if
(WebViewFeature.isFeatureSupported(WebViewFeature.FORCE_DARK))
{ ... }
```

Всего три варианта для тёмной темы.

- FORCE_DARK_OFF

Включаем ночной режим.

```
WebSettingsCompat.setForceDark(webView.settings,
WebSettingsCompat.FORCE_DARK_ON)
```

Получаем текущий режим.

```
val forceDarkMode =
WebSettingsCompat.getForceDark(webView.settings)
```

Дополнительное чтение

WebView - создай свой браузер

Продвинутый WebView

Загрузка изображения на сервер в WebView

Android WebView — Downloading Images

Реклама