Лекция #33. Работа с сетью. WebView

WebView

WebView представляет простейший элемент для рендеринга html-кода, базирующийся на движке WebKit. Благодаря этому мы можем использовать WebView как примитивный веб-браузер, просматривая через него контент из сети интернет. Использование движка WebKit гарантирует, что отображение контента будет происходить примерно также, как и в других браузерах, построенных на этом движке - Google Chrome и Safari.

Некоторые основные методы класса WebView:

- **boolean canGoBack():** возвращает true, если перед текущей веб-страницей в истории навигации WebView еще есть страницы
- **boolean canGoForward():** возвращает true, если после текущей веб-страницей в истории навигации WebView еще есть страницы
- **void clearCache(boolean includeDiskFiles):** очищает кэш WebView
- **void clearFormData():** очищает данный автозаполнения полей форм
- **void clearHistory():** очищает историю навигации WebView
- **String getUrl():** возвращает адрес текущей веб-страницы
- **void goBack():** переходит к предыдущей веб-странице в истории навигации
- **void goForward():** переходит к следующей веб-странице в истории навигации
- **void loadData(String data, String mimeType, String encoding):** загружает в веб-браузере данные в виде html-кода, используя указанный mime-тип и кодировку

- **void loadDataWithBaseURL (String baseUrl, String data, String mimeType, String encoding, String historyUrl):** также загружает в веб-браузере данные в виде html-кода, используя указанный mime-тип и кодировку, как и метод loadData(). Однако кроме того, в качестве первого параметра принимает валидный адрес, с которым ассоциируется загруженные данные.
  Зачем нужен этот метод, если есть loadData()? Содержимое, загружаемое методом loadData(), в качестве значения для window.origin будет иметь значение null, и таким образом, источник загружаемого содержимого не сможет пройти проверку на достоверность. Метод loadDataWithBaseURL() с валидными адресами (протокол может быть и HTTP, и HTTPS) позволяет установить источник содержимого.
- **void loadUrl(String url):** загружает веб-страницу по определенному адресу
- **void postUrl(String url, byte[] postData):** отправляет данные с помощью запроса типа "POST" по определенному адресу
- **void zoomBy(float zoomFactor):** изменяет масштаб на опредленный коэффициент
- **boolean zoomIn():** увеличивает масштаб
- **boolean zoomOut():** уменьшает масштаб

Работать с WebView очень просто. Определим данный элемент в разметке layout:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <WebView
        android:id="@+id/webViewMain"
        android:layout_width="409dp"
        android:layout_height="729dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Для получения доступа к интернету из приложения, необходимо указать в файле манифеста AndroidManifest.xml соответствующее разрешение:

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

А так же свойство

```
android:hardwareAccelerated="true"
```

В

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/Theme.WebViewDemonstration"
    android:hardwareAccelerated="true">
```

Это необходимо чтобы приложение отрисовывалось с ускорением графического процессора устройства и не «лагало».

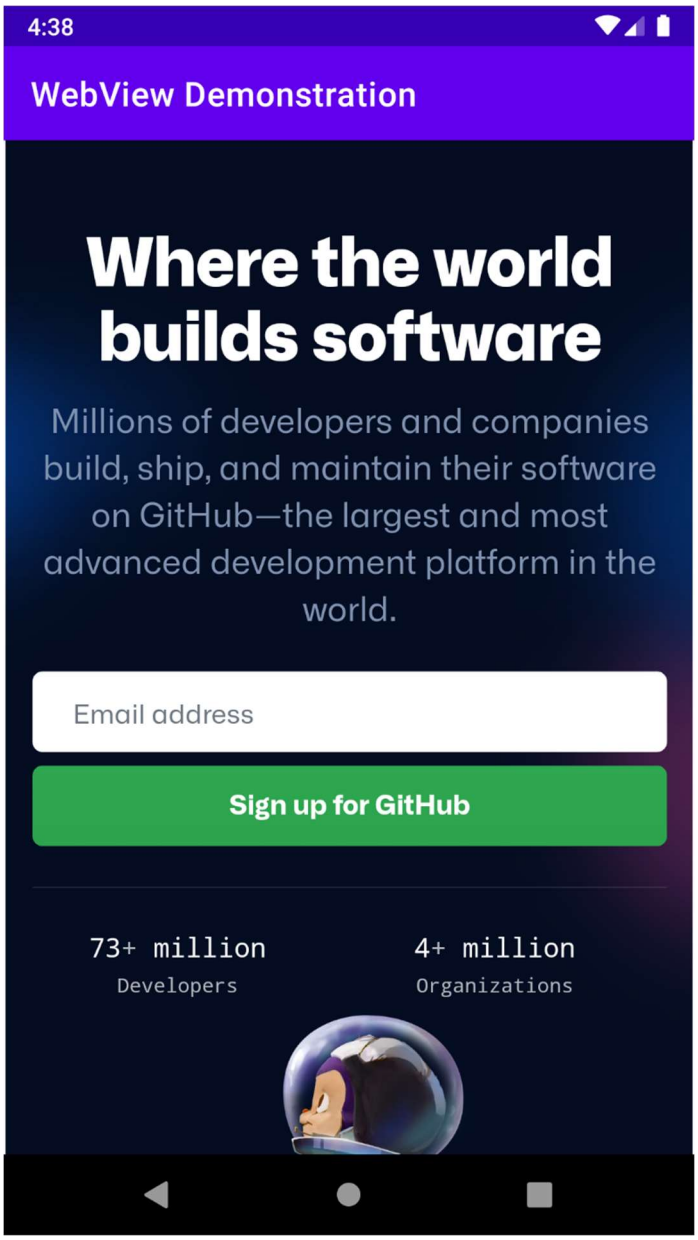
Чтобы загрузить определенную страницу в WebView, через метод loadUrl() надо установить ее адрес:

```
package com.awkitsune.webviewdemonstration

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.webkit.WebView

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val webView = findViewById<WebView>(R.id.webViewMain)
        webView.loadUrl("https://github.com/")
    }
}
```



Вместо определения элемента в layout мы можем создать WebView в коде Activity:

```
package com.awkitsune.webviewdemonstration

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.webkit.WebView

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        val webView = WebView(this)
        setContentView(webView)
        webView.loadUrl("https://github.com/")
    }
}
```

Кроме загрузки конкретной страницы из интернета с помощью метод **loadData():**

```
package com.awkitsune.webviewdemonstration

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.webkit.WebView

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        val webView = WebView(this)
        setContentView(webView)
        webView.loadData(
            "<html><body><h1>Greetings, fellow humans, human
fellas!</h1></body></html>",
            "text/html",
            "UTF-8"
        )
    }
}
```

Первым параметром метод принимает строку кода html, во втором - тип содержимого, а в третьем - кодировку.

JavaScript

По умолчанию в WebView отключен javascript, чтобы его включить надо изменить настройку **javaSctiptEnabled** на true:

```
val webView = WebView(this)
setContentView(webView)
webView.settings.javaScriptEnabled = true
webView.loadUrl("https://github.com/")
```

Загрузка данных и класс HttpURLConnection

На сегодняшний день если не все, то большинство Android-устройств имеют доступ к сети интернет. А большое количество мобильных приложений так или иначе взаимодействуют с средой интернет: загружают файлы, авторизуются и получают информацию с внешних веб-сервисов и т.д. Рассмотрим, как мы можем использовать в своем приложении доступ к сети интернет.

Среди стандартных элементов нам доступен виджет WebView, который может загружать контент с определенного url-адреса. Но этим возможности работы с сетью в Android не ограничиваются. Для получения данных с определенного интернет-ресурса мы можем использовать классы **HttpUrlConnection** (для протокола HTTP) и **HttpsUrlConnection** (для протокола HTTPS) из стандартной библиотеки Kotlin.

Итак, создадим новый проект с пустой MainActivity. Первым делом для работы с сетью нам надо установить в файле манифеста **AndroidManifest.xml** соответствующее разрешение:

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

В файле **activity_main.xml**, который представляет разметку для MainActivity, определим следующий код:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <Button
        android:id="@+id/downloadBtn"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:layout_marginLeft="16dp"
```

```
        android:layout_marginRight="16dp"
        android:text="Загрузка"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <WebView
        android:id="@+id/webView"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:layout_marginTop="8dp"
        android:layout_marginLeft="16dp"
        android:layout_marginRight="16dp"
        app:layout_constraintBottom_toTopOf="@id/scrollView"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@id/downloadBtn" />
    <ScrollView
        android:id="@+id/scrollView"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:layout_marginLeft="16dp"
        android:layout_marginRight="16dp"
        android:layout_marginTop="8dp"
        android:layout_marginBottom="8dp"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@id/webView"
        app:layout_constraintBottom_toBottomOf="parent">

        <TextView android:id="@+id/content"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" />
    </ScrollView>

</androidx.constraintlayout.widget.ConstraintLayout>
```

Здесь определена кнопка для загрузки данных, а сами данные для примера загружаются одновременно в виде строки в текстовое поле и в элемент WebView. Так как данных может быть очень много, то текстовое поле помещено в элемент ScrollView.

Поскольку загрузка данных может занять некоторое время, то обращение к интернет-ресурсу определим в отдельном потоке и для этого изменим код **MainActivity** следующим образом:

```
package com.awkitsune.httpurlconnectiondemonstration

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Button
import java.io.BufferedReader
```

```
import java.io.IOException
import java.io.InputStream
import java.io.InputStreamReader
import java.lang.StringBuilder
import java.net.URL
import javax.net.ssl.HttpsURLConnection
import android.widget.Toast
import android.webkit.WebView
import android.widget.TextView

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val contentView = findViewById<TextView>(R.id.content)

        val webView = findViewById<WebView>(R.id.webView)
        webView.settings.javaScriptEnabled = true

        findViewById<Button>(R.id.downloadBtn).setOnClickListener {
            contentView.text = "Загрузка..."

            Thread {
                try {
                    val content = getContent("https://stackoverflow.com/")
                    webView.post {
                        webView.loadDataWithBaseURL(
                            "https://stackoverflow.com/",
                            content,
                            "text/html",
                            "UTF-8",
                            "https://stackoverflow.com/"
                        )
                        Toast.makeText(
                            applicationContext,
                            "Данные загружены",
                            Toast.LENGTH_SHORT
                        ).show()
                    }
                    contentView.post { contentView.text = content }
                } catch (ex: IOException) {
                    contentView.post {
                        contentView.text = "Ошибка: " + ex.message
                        Toast.makeText(applicationContext, "Ошибка",
Toast.LENGTH_SHORT).show()
                    }
                }
            }.start()
        }
    }

    @Throws(IOException::class)
    private fun getContent(path: String): String {
        var reader: BufferedReader? = null
```

```
        var stream: InputStream? = null
        var connection: HttpsURLConnection? = null
        return try {
            val url = URL(path)
            connection = url.openConnection() as HttpsURLConnection
            connection.requestMethod = "GET"
            connection!!.readTimeout = 10000
            connection.connect()
            stream = connection.inputStream
            reader = BufferedReader(InputStreamReader(stream))
            val buf = StringBuilder()
            var line: String?
            while (reader.readLine().also { line = it } != null) {
                buf.append(line).append("\n")
            }
            buf.toString()
        } finally {
            reader?.close()
            stream?.close()
            connection?.disconnect()
        }
    }
}
```

Непосредственно для самой загрузки определен метод getContent(), который будет загружать веб-страницу с помощью класса HttpsURLConnection и возвращать код загруженной страницы в виде строки.

Вначале создается элемент **HttpsURLConnection**:

```
val url = URL(path)
connection = url.openConnection() as HttpsURLConnection
connection.requestMethod = "GET"
connection!!.readTimeout = 10000
connection.connect()
```

После подключение происходит считывание со входного потока:

```
stream = connection.inputStream
reader = BufferedReader(InputStreamReader(stream))
```

Используя входной поток, мы можем считать его в строку.

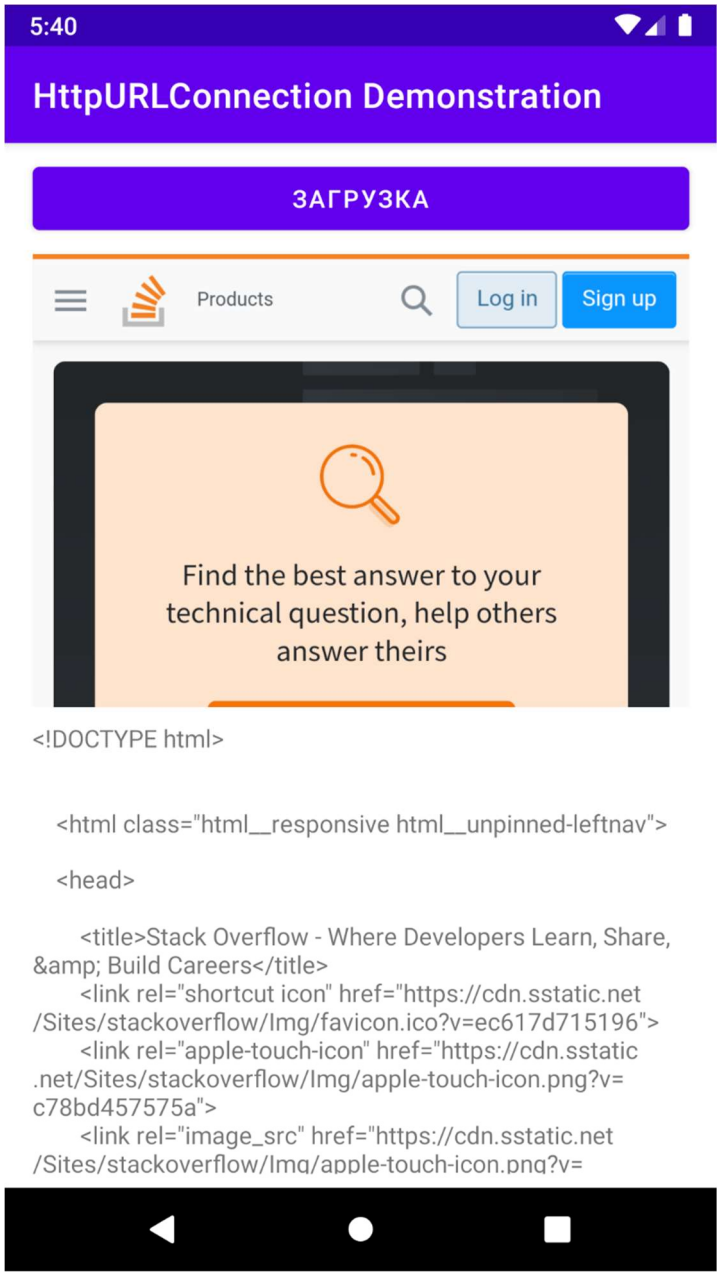
Этот метод getContent() затем будет вызываться в обработчике нажатия кнопки:

```
findViewById<Button>(R.id.downloadBtn).setOnClickListener {
    contentView.text = "Загрузка..."

    Thread {
        try {
            val content = getContent("https://stackoverflow.com/")
```

Поскольку загрузка может занять долгое время, то метод getContent() в отдельном потоке с помощью объектов Thread и Runnable. Для примера в данном случае обращение идет к ресурсу *"https://stackoverflow.com/"*.

Запустим приложение и нажмем на кнопку. И при наличии интернета приложение загрузит гравную страницу с *"https://stackoverflow.com/"* и отобразит ее в WebView и TextView:



Конечно, данный способ вряд ли подходит для просмотра интернет-страниц, однако таким образом, мы можем получать какие-либо данные (не интернет-страницы) от различных веб-сервисов, например, в формате xml или json (например, различные курсы валют, показатели погоды), используя специальные api, и затем после обработки показывать их пользователю.