Desarrollo para dispositivos móviles con Android: Aplicaciones web con Android WebView y Apache Cordova.

Contenido

Desarrollo para dispositivos móviles con Android: Aplicaciones web con Android WebView y Apache Cordova....1
1 Una aplicación web desde una aplicación Android....2
1.1 Introducción....2
1.2 El control WebView....2
1.3 Cómo configurar WebView....3
1.4 Cargar una aplicación HTML desde los assets....3
1.5 Cómo cargar un archivo desde los recursos....5
1.6 Llamando a código Android desde JavaScript....6
2 Una aplicación Android desde una aplicación web....7
2.1 Instalación....7
2.2 Creando la aplicación....7
2.3 Codificando en HTML para Android....8
2.4 Probando la aplicación....9
2.5 Compilando y ejecutando la aplicación....10
2.6 Ionic....10
3 Referencias....11

1 Una aplicación web desde una aplicación Android

1.1 Introducción

La forma más común de programar una aplicación en Android es una suerte de aplicación de escritorio, adaptada a las particularidades de un dispositivo portátil. Es posible, sin embargo, programar toda o parte de la aplicación como una aplicación web, empleando los mismos recursos que en aquellas: HTML5, CSS y JavaScript. La posibilidad de poder llevar a cabo esta aproximación tan diferente es el *widget* **WebView**, que está basado en Chrome. Si bien será necesario tener en cuenta, aún así, las particularidades del dispositivo, es perfectamente posible programar una aplicación completa de esta forma, que además puede comunicarse con la aplicación *host* creada para Android.

1.2 El control WebView

El control **WebView** se puede incorporar a una *layout* de la misma manera que se incorpora un botón o un campo de texto.

```
<WebView
    android:id=”@+id/wvView”
    android:layout_width=”fill_parent”
    android:layout_height=”fill_parent” />
```

A continuación se muestra un ejemplo completo de un *layout* que utiliza un **WebView** que ocupa prácticamente toda la pantalla. Una pequeña parte se dedica a un botón que permite abandonar la aplicación.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:orientation="vertical"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent" >
    <WebView
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:id="@+id/wvView"
            android:layout_gravity="center_horizontal"
            android:layout_weight="0.8" />
    <LinearLayout
            android:orientation="horizontal"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="0.2" android:gravity="right">
        <ImageButton
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:id="@+id/btBack"
                android:src="@android:drawable/ic_menu_close_clear_cancel" />
    </LinearLayout>
</LinearLayout>
```

Una vez el control ha sido creado, puede ser utilizado para cargar en él HTML presente en la aplicación, o puede indicársele también que cargue una URL, que podríamos denominar “externa”. En este último caso, es necesario darle a la aplicación el permiso “internet”, cuya ruta completa es “android.permissions.INTERNET”. Para poder hacer esto, es necesario modificar el archivo *AndroidManifest.xml*. En cualquier caso, este no es el uso más útil de este control (a no ser que se quiera crear un navegador, o parte de los recursos residan en la web). En definitiva, si los archivos HTML que se van a cargar en **WebView** están dentro de la aplicación como recursos (en las siguientes secciones se verá cómo crear estos recursos), este permiso no es necesario.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.example.AndroidWebViewCalculator"
          android:versionCode="1"
          android:versionName="1.0">
    <uses-sdk android:minSdkVersion="21"/>
    <uses-permission android:name="android.permission.INTERNET" />
    <application android:label="@string/app_name" android:icon="@drawable/ic_launcher">
        <activity android:name="Main"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Para cargar una página web situada en una URL determinada, se utiliza el método **WebView**.*loadUrl(url)*, que acepta una URL en formato de cadena.

```
public class Main extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate( savedInstanceState );
        requestWindowFeature( Window.FEATURE_NO_TITLE );
        this.setContentView( R.layout.main );

        ImageButton btBack = (ImageButton) this.findViewById( R.id.btBack );
        WebView wvView = (WebView) this.findViewById( R.id.wvView );

        btBack.setOnClickListener( new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Main.this.finish();
            }
        } );

        wvView.loadUrl( “http://www.google.es/” );
    }
```

1.3 Cómo configurar WebView

Por defecto, un control **WebView** no es capaz de ejecutar JavaScript. Además, en cuanto se pulse un hiperenlace en él, se abrirá el navegador por defecto, en lugar de continuar utilizando el propio navegador del **WebView**. Normalmente, este no es el comportamiento deseado, y es necesario configurarlo para que se comporte como esperamos.

Lo primero, es necesario permitir JavaScript. Para ello, será necesario acceder a la configuración mediante **WebView**.*getSettings()*, que está almacenada en un objeto **WebSettings**. Dentro de este objeto existen muchos métodos interesantes para variar la configuración, pero el que nos interesa es **WebSettings**.*setJavaScriptEnabled(boolean);*. Al pasar un **true**, JavaScript queda activado. Además, es necesario que al hacer click en un enlace, no se abra el navegador por defecto. Para eso, será necesario llamar al método **WebView**.*setWebViewClient(***WebViewClient***);* si se le pasa un nuevo objeto **WebViewClient**, automáticamente se consigue el efecto deseado.

1.4 Cargar una aplicación HTML desde los assets

Es posible crear una aplicación completa HTML, y cargarla desde el directorio *assets*. El subdirectorio de recursos (*assets*), debe estar al mismo nivel que el directorio *res*, es decir, en *app/src/main/*. Desde Android Studio, es posible crear el directorio automáticamente pulsando botón derecho sobre el proyecto, y seleccionando *New >> Folder >> Assets Folder*.

En este caso, se colocará el archivo *calc.html*, cuyo contenido se muestra a continuación. Se trata de una aplicación muy simple que permite sumar dos números.

```
<html>
<head>
<meta charset="utf-8"/>
<title>Calculator</title>
<script type="text/javascript">
function calculate() {
        var edOp1 = document.getElementById( "edOp1" );
        var edOp2 = document.getElementById( "edOp2" );
        var dvResult = document.getElementById( "dvResult" );

        dvResult.innerHTML = "";
        var pResult = document.createElement( "p" );
        dvResult.appendChild( pResult );

        var op1 = parseInt( edOp1.value, 10 );
        var op2 = parseInt( edOp2.value, 10 );
        var res = op1 + op2;
        pResult.appendChild( document.createTextNode( res.toString() ) );
        Android.showToast( res.toString() );
}
</script>
</head>
<body>
<p>
        <form id="formCalc" onSubmit="javascript: calculate(); return false;">
        <input id="edOp1" type="text" size="10" placeholder="0" style="text-align: right"/>
        +
        <br /><input id="edOp2" type="text" size="10" placeholder="0" style="text-align: right"/>
        <input id="btCalc" type="submit" value="="/>
        </form>
</p>
<p>
    Result:
    <div id="dvResult">
    </div>
</p>
</body>
</html>
```

Y desde la aplicación Android, solo será necesario configurar el *web view*, y decirle que cargue el archivo desde los recursos.

```
    private void configureWebView(WebView wvView, String url, int defaultFontSize)
    {
        WebSettings webSettings = wvView.getSettings();

        webSettings.setBuiltInZoomControls( true );
        webSettings.setDefaultFontSize( defaultFontSize );
        webSettings.setJavaScriptEnabled( true );

        // Impedir que se lance el navegador por defecto al pulsar en un enlace
        wvView.setWebViewClient( new WebViewClient() );

        // Carga un archivo HTML desde el archivo de recursos
        wb.loadUrl("file:///android_asset/calc.html");
    }
```

De esta forma tan sencilla, es posible ejecutar una aplicación web HTML5 completa desde un código de soporte muy sencillo.

En una sección posterior, se tratará como interactuar desde código Android con el código JavaScript.

1.5 Cómo cargar un archivo desde los recursos

Es posible cargar cualquier archivo desde el directorio de recursos. A continuación, se verá cómo cargar un archivo HTML desde allí, y cargarlo en el *web view*.

Para poder cargar código HTML está previsto poder usar el método **WebView**.*loadData(s, m, e)*. Este método acepta una cadena con el código HTML, el *mime type* (normalmente, "text/html"), y la codificación de caracteres o *encoding*, que suele ser "utf-8".

Los archivos que se desee que estén disponibles para la aplicación deben colocarse en el subirectorio de recursos (*assets*). De nuevo, el subdirectorio de recursos debe estar al mismo nivel que el directorio *res*, es decir, en *app/src/main/*. Desde Android Studio, es posible crear el directorio automáticamente pulsando botón derecho sobre el proyecto, y seleccionando *New >> Folder >> Assets Folder*.

De nuevo, se colocará el archivo *calc.html*, cuyo contenido se muestra a continuación. Se trata de una aplicación muy simple que permite sumar dos números.

```
<html>
<head>
<meta charset="utf-8"/>
<title>Calculator</title>
<script type="text/javascript">
function calculate() {
        var edOp1 = document.getElementById( "edOp1" );
        var edOp2 = document.getElementById( "edOp2" );
        var dvResult = document.getElementById( "dvResult" );

        dvResult.innerHTML = "";
        var pResult = document.createElement( "p" );
        dvResult.appendChild( pResult );

        var op1 = parseInt( edOp1.value, 10 );
        var op2 = parseInt( edOp2.value, 10 );
        var res = op1 + op2;
        pResult.appendChild( document.createTextNode( res.toString() ) );
        Android.showToast( res.toString() );
}
</script>
</head>
<body>
<p>
        <form id="formCalc" onSubmit="javascript: calculate(); return false;">
        <input id="edOp1" type="text" size="10" placeholder="0" style="text-align: right"/>
        +
        <br /><input id="edOp2" type="text" size="10" placeholder="0" style="text-align: right"/>
        <input id="btCalc" type="submit" value="="/>
        </form>
</p>
<p>
    Result:
    <div id="dvResult">
    </div>
</p>
</body>
</html>
```

Una vez colocado allí, puede ser accedido mediante el método **Assets**.*open(archivo)*, y la instancia de **Assets** se obtiene desde el método **Activity**.*getAssets()*. Pasándole el nombre del archivo, devuelve un objeto **InputStream**, que puede ser utilizado para construir un **BufferedReader**, y poder leer líneas. Así, es posible crear una cadena de texto que pueda ser pasada al método **WebView**.*loadData(s, t, c)*, siendo *s* el contenido del archivo, *t* el tipo **MIME** (normalmente, *text/html*), y *c* la codificación de caracteres (normalmente, *utf-8*).

A continuación, se incluye una configuración completa del *webView*, incluso el cargado de la página HTML.

```
    private void configureWebView(WebView wvView, String url, int defaultFontSize)
    {
        WebSettings webSettings = wvView.getSettings();

        webSettings.setBuiltInZoomControls( true );
        webSettings.setDefaultFontSize( defaultFontSize );

        // Permitir javascript y la interfaz javascript/android
        webSettings.setJavaScriptEnabled( true );
        wvView.addJavascriptInterface( new WebAppInterface( this ), "Android" );

        // Impedir que se lance el navegador por defecto al pulsar en un enlace
        wvView.setWebViewClient( new WebViewClient() );

        // Carga un archivo HTML desde el archivo de recursos
        StringBuilder builder = new StringBuilder();
        try {
            String line;

            InputStream in = this.getAssets().open( "calc.html" );
            BufferedReader inf = new BufferedReader( new InputStreamReader( in ) );

            while( ( line = inf.readLine()) != null) {
                builder.append( line );
            }
        } catch (IOException e) {
            builder.append( "<html><body><big>ERROR internal: loading asset</big></body></html>");
        }
       finally {
           try {
               if ( in != null ) {
                   in.close();
                }
           catch(IOException exc) {
              Log.e( “calc.onCreate”, exc.getMessage() );
           }
        }

        wvView.loadData( builder.toString(), "text/html", "utf-8" );
    }
```

En el código más arriba, se deja cargada la página “calc.html” (una aplicación HTML5 muy sencilla, que involucra un formulario y código JavaScript para sumar dos números) para que funcione como código principal de la aplicación.

1.6 Llamando a código Android desde JavaScript

Para poder utilizar código Android desde JavaScript, es necesario crear una clase que actúe como puente entre los dos mundos. Esta clase debe guardar el contexto (normalmente, la actividad donde se ejecuta el **WebView**) desde donde se utiliza, y debe tener varios métodos con el decorador *@javascriptinterface*. Esos métodos son los que estarán disponibles desde HTML, a través de un objeto creado automáticamente por Android.

```
public class WebAppInterface {
    private Context mContext;

    /** Instantiate the interface and set the context */
    WebAppInterface(Context c)
    {
        mContext = c;
    }

    /** Show a toast from the web page */
    @JavascriptInterface
    public void showToast(String toast)
    {
        Toast.makeText( mContext, toast, Toast.LENGTH_LONG ).show();
    }
}
```

Para hacer que este objeto sea accesible desde el WebView, será necesario llamar al método **WebView**.*setJavaScriptInterface()* ofreciendo el nombre del objeto*: wvView.addJavascriptInterface(**new** WebAppInterface(this), "Android");*

2 Una aplicación Android desde una aplicación web

La segunda posibilidad es la utilización de *Cordova*, un *framework* que funciona desde *node.js*, y cuyo objetivo es poder generar varios tipos de aplicaciones (Android, iOS, web, Windows…) desde el mismo código fuente HTML5.

2.1 Instalación

El primer paso es la instalación de *node.js*. Se trata de un intérprete de JavaScript que funciona fuera del navegador, es decir, no es necesario un navegador para poder ejecutar los programas creados. Tanto *node.js* como *cordova* se manejan desde la consola o línea de comandos.

Una vez instalado, es posible ejecutar *node.js* con la orden "node", con lo que el intérprete nos muestra el indicativo para órdenes '>'.

```
$ node
> var msg = "hola, "
undefined
> var persona = "baltasar"
undefined
> console.log(msg + persona)
hola, baltasar
> .exit
```

Sobre este intérprete funciona el *framework Cordova*, que es la versión libre (mantenida por la fundación **Apache**) del *framework PhoneGap* de **Adobe**. Con *node.js*, se incluye un gestor de paquetes: *npm*. Con este gestor de paquetes instalaremos Cordova, así como cualquier otro paquete que podamos necesitar.

```
$ npm install -g cordova
...
Cordova 7.1.0
```

2.2 Creando la aplicación

Una vez instalado, podemos comenzar a crear aplicaciones. Para ello, utilizamos la órden create, a la que debemos acompañar de un nombre para la aplicación, y un identificador único para ella.

```
$ cordova create calc com.devbaltasarq.calc
Creating a new cordova project.
$ cd calc
$ ls
config.xml  hooks  package.json  platforms  plugins  res  www
```

Bajo el directorio desde el que se creó la aplicación, se crea un directorio con el mismo nombre que el de la *app*, que tiene los archivos que se muestran sobre estas líneas. Es importante entrar en dicho directorio con *cd*, o el resto de órdenes no funcionarán. El archivo *config.xml* permite incluir información sobre la futura aplicación, como se ve a continuación.

```
$ cat config.xml
<?xml version='1.0' encoding='utf-8'?>
<widget         id="com.devbaltasarq.calc"
                version="1.0.0"
                xmlns="http://www.w3.org/ns/widgets"
                xmlns:cdv="http://cordova.apache.org/ns/1.0" >
    <name>HelloCordova</name>
    <description>
        A sample Apache Cordova application that responds to the deviceready event.
    </description>
    <author email="dev@cordova.apache.org" href="http://cordova.io">
        Apache Cordova Team
    </author>
    <content src="index.html" />
    <plugin name="cordova-plugin-whitelist" spec="1" />
```

```
    <access origin="*" />
    <allow-intent href="http://*/*" />
    <allow-intent href="https://*/*" />
    <allow-intent href="tel:*" />
    <allow-intent href="sms:*" />
    <allow-intent href="mailto:*" />
    <allow-intent href="geo:*" />
    <platform name="android">
        <allow-intent href="market:*" />
    </platform>
    <platform name="ios">
        <allow-intent href="itms:*" />
        <allow-intent href="itms-apps:*" />
    </platform>
</widget>
```

En este fichero debemos cambiar la información para que se corresponda con nuestra aplicación, en especial el contenido de las etiquetas en negrita: *name* (nombre de la app), *description* (descripción), author (autor, incluyendo las etiquetas *email* (e.mail) y *href* (web)).

El siguiente paso consiste en indicar para qué plataformas queremos crear la aplicación. *Cordova* es capaz de crear aplicaciones para muchas plataformas por defecto, y este aspecto se puede expandir con otros *plugins*. Podemos ver las plataformas soportadas, y las instaladas para nuestra aplicación con la orden *platforms*. Se puede añadir un plataforma con *platform add*, y eliminar con *platform remove*. En cualquier caso, nuestro objetivo es claramente **Android**.

```
$ cordova platform add android
Using cordova-fetch for cordova-android@~6.3.0
Adding android project...
Creating Cordova project for the Android platform:
        Path: platforms/android
        Package: com.devbaltasarq.calc
        Name: calc
        Activity: MainActivity
        Android target: android-26
Subproject Path: CordovaLib
Android project created with cordova-android@6.3.0
Discovered plugin "cordova-plugin-whitelist" in config.xml. Adding it to the project
…
Saving android@~6.3.0 into config.xml file ...
$ cordova platforms
Installed platforms:
  android 6.3.0
Available platforms:
  blackberry10 ~3.8.0 (deprecated)
  browser ~5.0.0
  ios ~4.5.1
  osx ~4.0.1
  ubuntu ~4.3.4 (deprecated)
  webos ~3.7.0
  windows ~5.0.0
  www ^3.12.0
```

Sobre estas líneas, se puede comprobar cómo nuestra aplicación está ya configurada para crear ejecutables para **Android** (archivos .apk).

2.3 Codificando en HTML para Android

En este momento, es necesario crear código para la aplicación. Todo el código se incluye en el directorio *www*, que este momento está ocupado con una pequeña aplicación por defecto que simplemente demuestra que funciona.

```
$ cd www
$ rm -rf *
$ ls
$ cp /path/to/calc.html ./index.html
```

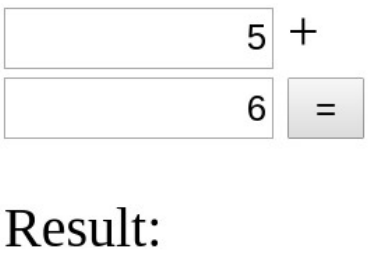
```
$ cat index.html
<html>
<head>
<meta charset="utf-8"/>
<title>Calculator</title>
<script type="text/javascript">
function calculate() {
        var edOp1 = document.getElementById( "edOp1" );
        var edOp2 = document.getElementById( "edOp2" );
        var dvResult = document.getElementById( "dvResult" );

        dvResult.innerHTML = "";
        var pResult = document.createElement( "p" );
        dvResult.appendChild( pResult );

        var op1 = parseInt( edOp1.value, 10 );
        var op2 = parseInt( edOp2.value, 10 );
        var res = op1 + op2;
        pResult.appendChild( document.createTextNode( res.toString() ) );
}
</script>
</head>
<body>
<p>
        <form id="formCalc" onSubmit="javascript: calculate(); return false;">
        <input id="edOp1" type="text" size="10" placeholder="0" style="text-align: right"/>
        +
        <br /><input id="edOp2" type="text" size="10" placeholder="0" style="text-align: right"/>
        <input id="btCalc" type="submit" value="="/>
        </form>
</p>
<p>
    Result:
    <div id="dvResult">
    </div>
</p>
</body>
</html>
```

Como se puede ver sobre estas líneas, se crea un archivo HTML que permite crear una pequeña aplicación para sumar dos números. Una de las ventajas de programar con *Cordova* es que es posible crear los archivos y subdirectorios que deseemos dentro de *www*, por lo que se puede dividir el código más arriba entre lógica de negocio (el código JavaScript dentro de la etiqueta *script*), y la vista (el resto del archivo HTML), así como añadir facilmente otras posibilidades como por ejemplo archivos de estilo css.

```
$ ls
index.html maths.js maths.css
```

2.4 Probando la aplicación

Antes de generar la aplicación para nuestro dispositivo, es conveniente comprobar que funciona. Para ello es interesante la órden *serve*, que crea un pequeño servidor local permitiendo comprobar que la aplicación es correcta.




```
$ cd..
$ pwd
../calc
$ cordova serve
Static file server running on: http://localhost:8000
(CTRL + C to shut down)
```

Al abrir la URL dada, *localhost:8000*, en cualquier navegador, nos encontraremos con un servidor que nos permitirá probar las plataformas añadidas (en nuestro caso, debemos seleccionar **Android**), de manera que podremos ver nuestro programa en acción.

2.5 Compilando y ejecutando la aplicación

Con *Cordova* es posible compilar y ejecutar cada una de las plataformas añadidas por separado. Para ello se utiliza la órden *build <pltf>*, donde *pltf* es la plataforma escogida. De una manera similar, se puede probar la aplicación con la órden *run <pltf>*.

```
$ cordova build android
ANDROID_HOME=/home/baltasarq/bin/android-sdk
JAVA_HOME=/usr/lib/jvm/java-8-openjdk
Starting a Gradle Daemon (subsequent builds will be faster)
...
:packageDebug
:assembleDebug
:cdvBuildDebug

BUILD SUCCESSFUL

Total time: 18.061 secs
Built the following apk(s):
        /home/baltasarq/calc/platforms/android/build/outputs/apk/android-debug.apk
```

Como se puede observar sobre estas líneas, se genera un archivo *apk* que se puede transferir a un teléfono para probarlo. En el caso de añadir el parámetro *--release*, entonces se genera un archivo *apk* que solo necesita ser firmado digitalmente para poder subirlo a la tienda (como se verá en una siguiente sección).

La ejecución se realiza con la órden *run*:

```
$ cordova run android
ANDROID_HOME=/home/baltasarq/bin/android-sdk
JAVA_HOME=/usr/lib/jvm/java-8-openjdk
Subproject Path: CordovaLib
...
Built the following apk(s):
        /home/baltasarq/calc/platforms/android/build/outputs/apk/android-debug.apk
ANDROID_HOME=/home/baltasarq/bin/android-sdk
JAVA_HOME=/usr/lib/jvm/java-8-openjdk
No target specified and no devices found, deploying to emulator
```

En caso de estar el teléfono conectado al ordenador, la aplicación se lanzará en él; en otro caso, se abrirá el emulador para ejecutarla. En algunas ocasiones, es necesario lanzar el emulador manualmente, pues no es capaz de lanzarlo por defecto.

2.6 Ionic

Ionic es un framework especialmente diseñado para crear aplicaciones para plataformas **Android** e **iOS**. Si bien funciona sobre *Cordova*, añade *widgets* y otras ayudas especialmente diseñadas para dispositivos móviles.

En la sección de referencias se encuentran las URL's de acceso tanto a *Cordova* como a *Ionic*, ya que la profundización en dichos frameworks está fuera del alcance de este texto.

En cualquier caso, *Ionic* se instala con *npm install -g ionic*, con *ionic start <app>* se crea el directorio de partida con los fuentes iniciales para la aplicación *app* (de modo similar a con *Cordova*, y se puede probar con *ionic serve*.

3 Referencias

- Recursos Android (accedido en nov. 2017):
  - Documentación y recursos de Android para desarrolladores.
    http://developer.android.com/
  - Uso de web view
    http://developer.android.com/intl/es/guide/webapps/webview.html
  - Referencia de WebView
    http://developer.android.com/intl/es/reference/android/webkit/WebView.html
  - Ejemplo (muy simple) completo:
    https://github.com/baltasarq/AndroidWebViewCalculator
- Recursos Cordova (accedido en nov. 2017):
  - Node.js
    https://nodejs.org/es/
  - Cordova:
    https://cordova.apache.org
  - Ionic:
    https://ionicframework.com/
  - Documentación básica Cordova:
    https://cordova.apache.org/docs/es/7.x/index.html
  - Documentación básica de Ionic (inglés):
    https://ionicframework.com/docs/intro/installation/