ULIZA Player (HTML5)

WebViewインテグレーションガイド v1.0.4

はじめに

本書では、ULIZA Player (HTML5)をAndroid／iOSのWebView上で動作させるための開発方法を説明します。

用語一覧

本書で使用する表記を説明します。

ポスター画像

コンテンツの再生開始前にプレイヤー領域に表示する画像です。video要素のposter属性で指定します。

プレイヤー

本文書では、ULIZA Player (HTML5)を指します。

プレイヤー領域

プレイヤー初期化の際に指定するサイズの領域です。コンテンツのアスペクト比によっては黒領域を含む場合があります。

リニア広告

本編の再生開始前、再生中、再生を完了した後に再生する広告です。動画広告とVPAID広告を含みます。

参考資料

- ULIZA Player (HTML5)ユーザーガイド

> **補足**
>
> 本書内では、ULIZA Player (HTML5)ユーザーガイドは「ユーザーガイド」と表記します。

サポート範囲

プレイヤーをWebView上で動作させる場合のサポート範囲を説明します。

WebViewのクラス

サポートするWebViewのクラスは、以下の通りです。

Android	WebViewクラス
iOS	WKWebViewクラス ※ UIWebViewクラスは非サポートです。

OSのバージョン

サポートするOSのバージョンは、「ユーザーガイド」を参照してください。

プレイヤーのバージョン

v1.10.0以降にリリースされたプレイヤーで動作します。サポートするプレイヤーのバージョンは、直近3マイナーバージョンです。一例として、最新のバージョンがv1.15.0の場合は、v1.13.0からv1.15.0をサポート対象とします。

機能

サポートするプレイヤーの機能は、以下の制限事項を除き、ブラウザ上で動作させる場合と同等です。詳細は、「ユーザーガイド」を参照してください。

- Google Cast Sender機能には対応していません。
- 以下の機種は非サポートです。
  - 704SH

Android向けの実装

基本実装

プレイヤーの基本的な機能を使用するために必要な実装を行います。本章内の以降のアプリの実装例は、基本実装がされている前提とします。

1. **AndroidManifest.xmlでのパーミッションの設定**

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

2. **WebView関連**

   WebViewインスタンスに必要な設定をしたうえで、プレイヤーを配置するWebページを表示します。

実装例は以下の通りです。

```java
// WebViewのインスタンスを取得する。
WebView webView = findViewById(R.id.webView);

// JavaScriptを有効にする。
webView.getSettings().setJavaScriptEnabled(true);

// プレイヤーを配置するWebページを表示する。
webView.loadUrl("https://host/path/playerpage.html");
```

フルスクリーン

WebView上のプレイヤーでフルスクリーン表示する際の拡大方式に 'native' を指定している場合、フルスクリーンボタンが反応しません。反応させるには、アプリに以下の実装を行います。

1. **WebChromeClientインスタンスの設定**

   以下2. ～ 3. のコールバックメソッドをオーバーライドしたWebChromeClientインスタンスをWebViewに設定します。

2. `WebChromeClient#onShowCustomView()` **メソッド**

   フルスクリーン表示する際にコールされます。第一引数でフルスクリーン状態のViewが渡されるので、Viewを見える状態にします。

3. `WebChromeClient#onHideCustomView()` **メソッド**

   フルスクリーン表示を解除する際にコールされます。フルスクリーン状態のViewを削除します。

実装例は以下の通りです。

```java
// 本実装例の外でViewGroup rootViewを宣言している。

// WebChromeClientインスタンスをWebViewに設定する。
webView.setWebChromeClient(new WebChromeClient() {
    View fullscreenView;

    @Override
    public void onShowCustomView(View view, CustomViewCallback callback) {
        // フルスクリーン状態で行いたい処理を書く。
        // アクションバーや画面の向きの調整等

        // フルスクリーン状態のviewを被せて見える状態にする。
```

```
            rootView.addView(view);
            fullscreenView = view;
        }

        @Override
        public void onHideCustomView() {
            // 非フルスクリーン状態で行いたい処理を書く。
            // アクションバーや画面の向きの調整等

            // フルスクリーン状態のviewを削除する。
            rootView.removeView(fullscreenView);
            fullscreenView = null;
        }
    };
```

リニア広告

Webページ内のリンクを開く際にWebView内で遷移する設定の状態において、WebView上のプレイヤーでリニア広告の再生中に広告のWebページへ遷移するリンクを開くとWebView内に表示します。ブラウザで遷移先のWebページを開くには、アプリに以下の実装を行います。

1. **マルチウィンドウの設定**

   マルチウィンドウを許可します。

2. **WebChromeClientインスタンスの設定**

   以下3. のコールバックメソッドをオーバーライドしたWebChromeClientインスタンスをWebViewに設定します。

3. `WebChromeClient#onCreateWindow()` **メソッド**

   ブラウザで遷移先のWebページを開く際にコールバックされます。WebページのURLを取得するために別のWebViewインスタンスを生成し、以下4. のコールバックメソッドをオーバーライドしたWebViewClientインスタンスを設定します。また、4. のメソッドがコールバックされるために必要な設定を、生成したWebViewインスタンスに対して行います。

4. `WebViewClient#onPageStarted()` **メソッド**

   遷移先のWebページの読み込みを開始した際にコールバックされます。第二引数からWebページのURLを取得し、ブラウザを起動します。

5. **Webページ内のリンクを開く際にWebView内で遷移する設定**

   以下6. のコールバックメソッドをオーバーライドしたWebViewClientインスタンスをWebViewに設定します。

6. `WebViewClient#shouldOverrideUrlLoading()` **メソッド**

   別のWebページへ遷移する際にコールバックされます。親クラスの同メソッドをコールするとWebView内で遷移します。1. ～ 4. を実装している場合、広告のWebページへ遷移するリンクを開いても本メソッドはコールされません。

実装例は以下の通りです。

```java
// マルチウィンドウを許可する。
webView.getSettings().setSupportMultipleWindows(true);

// WebChromeClientインスタンスをWebViewに設定する。
webView.setWebChromeClient(new WebChromeClient() {

    @Override
    public boolean onCreateWindow(WebView view, boolean isDialog, boolean isUserGesture, Message resultMsg) {
        // 別のWebViewインスタンスを生成する。
        WebView targetWebView = new WebView(getApplicationContext());
        targetWebView.setWebViewClient(new WebViewClient() {
            @Override
            public void onPageStarted(WebView view, String url, Bitmap favicon) {
                super.onPageStarted(view, url, favicon);
                // WebページのURLを設定した暗黙的インテントを使用してアプリ間遷移します。
```

```
                Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(url));
                startActivity(intent);
            }
        });

        // onPageStarted()がコールバックされるために必要な設定を行う。
        WebView.WebViewTransport transport = (WebView.WebViewTransport) resultMsg.obj;
        transport.setWebView(targetWebView);
        resultMsg.sendToTarget();

        return true;
    }
});

// Webページ内のリンクを開く際にWebView内で遷移させる。
webView.setWebViewClient(new WebViewClient() {
      @Override
      public boolean shouldOverrideUrlLoading(WebView view, String url) {
          return super.shouldOverrideUrlLoading(view, request);
      }
});
```

ポスター画像

WebView上のプレイヤーでポスター画像を指定しない場合、WebViewデフォルトのポスター画像が表示されます。ブラウザ上で動作させる場合と振る舞いを合わせるには、アプリに以下の実装を行います。なお、当該実装を行う場合でもポスター画像を指定していればポスター画像を表示します。

1. **WebChromeClientインスタンスの設定**

   以下2. のメソッドをオーバーライドしたWebChromeClientインスタンスをWebViewに設定します。

2. `WebChromeClient#getDefaultVideoPoster()` **メソッド**

   Webページをロードする際、ポスター画像が指定されていない場合にコールバックされます。ファーストフレームが見えるように透明な画像を返却します。

実装例は以下の通りです。

```java
// WebChromeClientインスタンスをWebViewに設定する。
webView.setWebChromeClient(new WebChromeClient() {
    @Override
    public Bitmap getDefaultVideoPoster() {
        // 透明な画像を返却する。
        return Bitmap.createBitmap(1, 1, Bitmap.Config.ARGB_8888);
    }
});
```

ビューポート

Webページに記述するビューポート設定 `<meta name="viewport" content="...">` を有効にするには、アプリに以下の実装を行います。

1. **ビューポートの設定**

   Webページに記述するビューポート設定を有効にします。

実装例は以下の通りです。

```java
// Webページに記述するビューポート設定を有効にする。
webview.getSettings().setUseWideViewPort(true);
```

iOS向けの実装

基本実装

プレイヤーの基本的な機能を使用するために必要な実装を行います。本章内の以降の実装例は、基本実装がされている前提とします。

1. **WebView関連**

   WKWebViewインスタンスに必要な設定をしたうえで、プレイヤーを配置するWebページを表示します。

実装例は以下の通りです。

```
WKWebViewConfiguration *conf = [[WKWebViewConfiguration alloc] init];

// インライン再生を許可する。
[conf setAllowsInlineMediaPlayback: YES];

// WebViewのインスタンスを生成し、配置する。
WKWebView *webView = [[WKWebView alloc] initWithFrame:self.view.bounds configuration:conf];
[self.view addSubview:webView];

// デリゲートを設定する。
webView.UIDelegate = self;

// プレイヤーを配置するWebページを表示する。
NSURL *url = [NSURL URLWithString:@"https://host/path/playerpage.html"];
NSURLRequest *request = [NSURLRequest requestWithURL:url];
[webView loadRequest:request];
```

リニア広告

WebView上のプレイヤーでリニア広告の再生中に広告のWebページへ遷移するリンクが反応しません。反応させるには、アプリに以下の実装を行います。

1. `[WKWebView createWebViewWithConfiguration:forNavigationAction:windowFeatures:]` **メソッド**

   広告のWebページへ遷移するリンクを開く際にコールされます。第三引数から遷移先のURLを取得し、SafariでWebページを表示します。

実装例は以下の通りです。

```
- (WKWebView *)webView:(WKWebView *)webView
    createWebViewWithConfiguration:(WKWebViewConfiguration *)configuration
               forNavigationAction:(WKNavigationAction *)navigationAction
                    windowFeatures:(WKWindowFeatures *)windowFeatures {
  // SafariでWebページを開く。
  [[UIApplication sharedApplication] openURL:navigationAction.request.URL
                                     options:@{}
                           completionHandler:nil];
  return nil;
}
```

改版履歴

v1.0.4 2022/1/31

- 【ポスター画像】変更しました。

v1.0.3 2021/1/25

- 「2.3. プレイヤーのバージョン」を更新

v1.0.2 2020/4/30

- 「1.3. 参考資料」を更新
- 「2.2. OSのバージョン」を更新
- 「2.4. 機能」を更新

v1.0.1 2019/9/6

- 社名変更に伴い、会社名とロゴを変更しました。

v1.0.0 2019/1/31

- 初版