

커스텀뷰

DDD 5기 직군 세션 (안드로이드)

정 린

목차

- CustomView의 constructor 을 deep하게 파헤치기
 - AttributeSet
 - JVMOverloads
 - defStyleAttr
 - defStyleRes
- 커스텀 Slider를 구현해보자!!
 - ThemeEnforcement.obtainStyledAttributes() vs context.obtainStyledAttributes() 의 차이??
 - invalidate() vs postInvalidate()
 - View의 LifeCycle
 - Slider ticker -> canvas rotate (하는 법)
 - 모션 감지
 - TooltipDrawable

커스텀뷰 생성자 4개의 역할

1. 인자를 1개 받는 constructor

```
constructor(context: Context) : super(context)
```

- 코드 상에서 직접 View를 구현할 경우

2. 인자를 2개 받는 constructor

```
constructor(context: Context, attrs: AttributeSet) : super(context, attrs)
```

- XML로부터 View를 Inflate할 경우
- Q) 그런데 AttributeSet이 뭐지??

AttributeSet이 무엇일까?

```
<com.ddd.android.session.DDSSlider
    android:id="@+id/slider"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    app:trackColorInactive="@color/color_black_200"
    app:trackColorActive="?colorOnSurface"
    app:tickColorInactive="@color/color_black_200"
    app:tickColorActive="?colorOnSurface"
    app:labelStyle="@style/Widget.Design.Tooltip.DDD"
    app:thumbStrokeColor="?colorOnSurface"
    app:thumbColor="?colorSurface"
    app:thumbStrokeWidth="@dimen/spacing_2xmicro"
    app:thumbRadius="@dimen/margin_normal"
    app:trackHeight="2dp"/>
```

XML에서 사용자가 지정한 커스텀뷰 속성
(예: activity_main.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <declare-styleable name="DDSSlider">
        <attr name="android:value"/>
        <attr name="android:valueFrom"/>
        <attr name="android:valueTo"/>
        <attr name="values" format="reference"/>
        <attr name="android:enabled"/>
        <attr name="android:stepSize"/>
        <attr name="thumbColor" format="color" />
        <attr name="thumbStrokeColor" format="color" />
        <attr name="thumbStrokeWidth" format="dimension" />
        <attr name="thumbRadius" format="dimension"/>
        <attr name="trackColor" format="color"/>
        <attr name="trackColorActive" format="color" />
        <attr name="trackColorInactive" format="color" />
        <attr name="tickVisible" format="boolean"/>
        <attr name="tickColor" format="color" />
        <attr name="tickColorActive" format="color"/>
        <attr name="tickColorInactive" format="color"/>
        <attr name="trackHeight" format="dimension"/>
        <attr name="labelStyle" format="reference" />
    </declare-styleable>
</resources>
```

res/values/attrs.xml에서 사용자가 지정한 속성 변수

AttributeSet이 커스텀뷰에 적용되는 과정

1. Custom Attributes를 attrs.xml의 `<declare-styleable>` 태그 내부에 선언한다.
2. 화면을 정의하는 xml 파일(예: activity main.xml)에 커스텀뷰 및 구체적인 값을 정의한다.
`app:thumbStrokeColor="?colorOnSurface"`
3. 런타임에서 AttributeSet에 선언된 구체적인 값을 가져온다.
`val typedArray: TypedArray = context.obtainStyledAttributes(attrs, R.styleable.DDDSlider)`
4. View에 이를 적용한다.

3. 인자를 3개 받는 constructor

```
constructor(context: Context, attrs: AttributeSet, defStyleAttr: Int):  
super(context, attrs, defStyleAttr)|
```

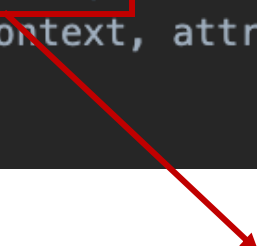
- View에 적용할 default style이 있는 경우 (없을 경우 0으로 설정한다.)
- Q) defStyleAttr이 없을 경우 2가지 인자만 받는 constructor를 호출하면 되지,
왜 굳이 3개의 인자를 받는 constructor를 호출하여 defStyleAttr에 0을 지정하는 코드를 자주 보게 되는가?
 - A) 2가지 인자만 받는 constructor를 호출해도 된다.
defStyleAttr에 0을 지정하는 것은 관습적인 표현.
- Q) 아무때나 defStyleAttr에 0을 선언하고 @JvmOverloads로 지정해주면 되나?
- Q) 그런데 defStyleAttr이 뭐지??

@JvmOverloads의 남발을 피해야 하는 이유

@JvmOverloads란?

메소드가 N개의 인자를 갖고 있을 때, 그 중 **M개**의 인자가 기본값을 갖는다고 가정하자.
이 경우 @JvmOverloads를 선언하면 **M개**의 overload 메소드가 추가로 생성된다.

```
class DDDButton @JvmOverloads constructor(  
    context: Context,  
    attrs: AttributeSet,  
    defStyleAttr: Int = 0  
) : MaterialButton(context, attrs, defStyleAttr) {  
    // ...  
}
```



이미 정의되어 있는 커스텀뷰를 상속할 경우,
기존 뷰에 정의되어 있던 style이 없어질 수 있다.

@JvmOverloads의 남발을 피해야 하는 이유

XML에서 View를 Inflating할 때 2개의 인자를 갖는 constructor가 호출된다.



2번째 constructor내에서 defStyleAttr에 0을 넘겨,
3개의 인자를 받는 constructor를 호출한다.

```
public DDDButton(Context context, @Nullable AttributeSet attrs){  
    this(context, attrs, 0);  
}
```

MaterialButton을
상속한 커스텀뷰

```
public MaterialButton(@NonNull Context context, @Nullable AttributeSet attrs) {  
    this(context, attrs, R.attr.materialButtonStyle);  
}
```

?!?!?!?

Parent view에서 지정한 style 대신
subclass가 정의한 0으로 대체되어버림!!

defStyleAttr은 무엇일까?

```
public TextView(Context context) { this(context, attrs: null); }
```

```
public TextView(Context context, @Nullable AttributeSet attrs) {  
    this(context, attrs, com.android.internal.R.attr.textViewStyle);  
}
```

Reference 참조타입 attribute을 넘겨준다.

TextView의 constructor

```
public TextView(Context context, @Nullable AttributeSet attrs, int defStyleAttr) {  
    this(context, attrs, defStyleAttr, defStyleAttrRes: 0);  
}
```

```
<declare-styleable name="Theme">  
    <!-- ...more... -->  
    <attr name="textViewStyle" format="reference" />  
    <!-- ...more... -->  
</declare-styleable>
```

**Android Platform이 사
전에 지정해 놓은
default-styleable**

defStyleAttr은 무엇일까?

```
<style name="Theme">
  <!-- ...more... -->
  <item name="textStyle">@style/Widget.TextView</item>
  <!-- ...more... -->
</style>
```

Android platform
기본 Theme

Application이나 Activity 에 선언되는 Theme

□ Theme의 정보를 가져옴

□ reference type의 attr가
@style/Widget.TextView를
참조하도록 한다.

```
final Resources.Theme theme = context.getTheme();
```

```
TypedArray a = theme.obtainStyledAttributes(attrs,  
com.android.internal.R.styleable.TextViewAppearance, defStyleAttr, defStyleRes);
```

3. 인자를 4개 받는 constructor

```
constructor(context: Context, attrs: AttributeSet, defStyleAttr: Int, defStyleRes: Int):  
super(context, attrs, defStyleAttr, defStyleRes)
```

- defStyleAttr 값이 0일 때 혹은 지정한 defStyleAttr를 찾을 수 없을 때 비로소 적용된다.
- defStyleRes에는 말 그대로 스타일 리소스를 지정하면 된다.
(예: R.style.sliderStyle)

defStyleRes는 왜 필요한가?

- 기본 style이 필요할 때 (defStyleAttr은 주로 "**defStyleRes에 지정된 styles**"를 상속한 style resource를 참조한다. 물론 무조건 상속해야 한다는 것은 아님.)

- 생성자에서는 defStyleAttr에 정의되지 않은 attribute가 AttributeSet에 선언되어 있을 경우, defStyleRes에 가서 찾게 된다.

- Q) 그런데 애초에 생성자를 3개만 구현했다면???

```
val a = ThemeEnforcement.obtainStyledAttributes(context, attrs, R.styleable.DDDButton,
defStyleAttr, DEF_STYLE_RES)
```

- 사실상 간단히 기본 attribute만 정의하고 싶을 때는 defStyleRes만 선언하여 인자로 넘기면 된다. 그러나 같은 커스텀 뷰라도 다양한 속성을 적용하여 표현하고 싶을 수 있다. 이럴 때 바뀔 가능성이 없는 기본값은 defStyleRes에 정의하고, 바뀔 가능성이 있는 기본값은 custom style 리소스에 정의하여, defStyleAttr가 이를 참조케 할 수 있다.

defStyleRes는 왜 필요한가?

```
<!-- themes.xml -->
<resources xmlns:tools="http://schemas.android.com/tools">
    <!-- Base application theme. --> Custom Theme
    <style name="Theme.Customview" parent="Theme.MaterialComponents.DayNight.NoActionBar">
        <!-- Customize your theme here. -->
        <item name="materialButtonStyle">@style/Widget.Button.DDD</item>
    </style>
</resources>

<!-- parent theme (Material theme) -->
<style name="Base.V14.Theme.MaterialComponents.Light.Bridge" parent="Platform.MaterialComponents.Light">
    <!-- more... -->
    <item name="materialButtonStyle">@style/Widget.MaterialComponents.Button</item>
    <!-- more... -->
</style>

<!-- styles.xml -->
<style name="Widget.Button.DDD" parent="Widget.MaterialComponents.Button">
    <item name="android:textColor">?colorOnSurface</item>
</style>
```

스타일 재정의

defStyleRes는 왜 필요한가?

activity_main.xml에 정의한 AttributeSet

```
<com.ddd.android.session.DDDButton  
    android:layout_width="0dp"  
    android:layout_height="wrap_content"  
    app:layout_constraintLeft_toLeftOf="parent"  
    app:layout_constraintRight_toRightOf="parent"  
    android:text="HELLO WORLD!"  
    android:textColor="@android:color/ho_lo_red_dark"  
    app:layout_constraintTop_toTopOf="parent"/>
```



1. custom styleable이 사전 적용된 경우

적용 순위

```
<declare-styleable name="DDDButton">  
    <attr name="android:textColor"/>  
</declare-styleable>
```



2. custom style에 attr의 value가 정의된 경우

(StyleableRes에서 value를 찾지 못한 경우)

```
<resources xmlns:tools="http://schemas.android.com/tools">
    <!-- Base application theme. -->
    <style name="Theme.Customview" parent="Theme.MaterialComponents.DayNight.NoActionBar">
        <item name="materialButtonStyle">@style/Widget.Button.DDD</item>
    </style>
</resources>
```

앱 내 기본 Theme에 custom style을 선언



```
<resources>
    <style name="Widget.Button.DDD" parent="Widget.MaterialComponents.Button">
        <!-- 적용 유무의 차이가 있다. -->
        <item name="android:textColor">@android:color/holo_blue_bright</item>
    </style>
</resources>
```

attr(예: android:textColor)을 custom style에 선언

```
<com.ddd.android.session.DDDButton
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    android:text="HELLO WORLD!"
    app:layout_constraintTop_toTopOf="parent"/>
```

AttributeSet에 선언된 attr이 없는 경우
(여기서는 android:textColor)



3. defStyleRes에 Theme가 적용된 경우

(StyleableRes | defStyleAttr에서 value를 찾지 못한 경우)

```
init {  
    val a = context.obtainStyledAttributes(attrs, R.styleable.DDDButton, defStyleAttr, DEF_STYLE_RES)  
  
    val textColor = a.getColor(R.styleable.DDDButton_android_textColor, android.R.color.darker_gray) ← Debugging...  
    a.recycle()  
}
```

```
ct x TypedArray.java x attrs.xml x themes.xml x styles.xml x ic_launcher_background.xml x  
@ColorInt  
public int getColor(@StyleableRes int index, @ColorInt int defValue) { index: 0 defValue: 17170432  
    if (mRecycled) { mRecycled: false  
        throw new RuntimeException("Cannot make calls to a recycled instance!");  
    }  
  
    final int attrIndex = index; attrIndex: 0  
    index *= STYLE_NUM_ENTRIES;  
  
    final int[] data = mData; data: {3, 134, 6, 2131034252, 1073746944, 0, 2131690094, 5, 1, 6, + 865 more}  
    final int type = data[index + STYLE_TYPE]; type: 3  
    if (type == TypedValue.TYPE_NULL) {  
        return defValue; defValue: 17170432  
    } else if (type >= TypedValue.TYPE_FIRST_INT  
        && type <= TypedValue.TYPE_LAST_INT) {  
        return data[index + STYLE_DATA]; data: {3, 134, 6, 2131034252, 1073746944, 0, 2131690094, 5, 1, 6, + 865 more}  
    } else if (type == TypedValue.TYPE_STRING) { type: 3  
        final TypedValue value = mValue; value: "TypedValue{t=0x3/d=0x86 \"res/color/mtrl_btn_text_color_selector.xml\" a=6 r=0x7f05008c}"  
        if (getValueAt(index, value)) { index: 0  
            final ColorStateList csl = mResources.loadColorStateList( csl: "ColorStateList{mThemeAttrs=nullmChangingConfigurations=512mStateSpecs=[[16842910], []]mColors=[-:  
                value, value.resourceId, mTheme); value: "TypedValue{t=0x3/d=0x86 \"res/color/mtrl_btn_text_color_selector.xml\" a=6 r=0x7f05008c}"  
            return csl.getDefaultColor(); csl: "ColorStateList{mThemeAttrs=nullmChangingConfigurations=512mStateSpecs=[[16842910], []]mColors=[-1, 1627389952]mDefaultColor=  
        }  
    }  
    return defValue;
```

3. defStyleRes에 Theme가 적용된 경우

(StyleableRes | defStyleAttr에서 value를 찾지 못한 경우)

```
▶ value = {TypedValue@9498} "TypedValue{t=0x3/d=0x86 "res/color/mtrl_btn_text_color_selector.xml" a=6 r=0x7f05008c}"
▼ csl = {ColorStateList@9606} "ColorStateList{mThemeAttrs=nullmChangingConfigurations=512mStateSpecs=[[16842910], []]mColors=[-1, 1627389952]mDefaultColor=1627389952}"
    f mChangingConfigurations = 512
    ▼ f mColors = {int[2]@9644}
        01 0 = -1
        01 1 = 1627389952
    f mDefaultColor = 1627389952
    ▶ f mFactory = {ColorStateList$ColorStateListFactory@9645}
    f mIsOpaque = false
    ▶ f mStateSpecs = {int[2][]@9646}
    f mThemeAttrs = null
    f ComplexColor.mChangingConfigurations = 512
```

[0] state:enable
[1] state:disable

res/color/mtrl_btn_text_color_selector.xml

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:color="?attr/colorOnPrimary" android:state_enabled="true"/>
  <item android:alpha="0.38" android:color="?attr/colorOnSurface"/>
</selector>
```

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

        // defStyleAttr에 android:textColor가 정의되어 있을 때
        val defStyleResAttrTextColor = String.format("#%06X", 0xFFFFFFFF and -16720385)
        // defStyleAttr에서 android:textColor를 못 찾았을 때
        val defStyleResTextColor = String.format("#%06X", 0xFFFFFFFF and -1)

        println("defStyleResAttrTextColor: $defStyleResAttrTextColor")
        println("defStyleResTextColor: $defStyleResTextColor")
    }
}
```

I/System.out: defStyleResAttrTextColor: #00DDFF

defStyleResTextColor: #FFFFFF

