Compose for WebでもDomじゃないCompose UIが使える？

2022/12/10 Kotlin Fest 2022

LINE Fukuoka

吉田清亮 YOSHIDA Seisuke

自己紹介

- @seisuke
- LINE証券のバックエンドエンジニア
- Kotlin + Spring Framework
- 二年程前までAndroidエンジニア Kotlinで開発
- ドライブワイパー（ウルトラハンコ）S+50

Agenda

- Motivation
- Demo
- Compose Multiplatform Framework
- Sample
- Conclusion

Motivation

- 勉強のために個人開発でCompose For Desktopアプリを開発
  - せっかくなのでcommonとjvmを分離してコードを書く
  - Experimentalで通常のCompose UIがWebでも使えることに気づく
- 業務でも管理システムなどの開発につかえるのでは？
  - LazyリストなどのJetpack Composeのエコシステム
- 情報がほとんどないため、とりあえず情報発信

https://github.com/seisuke/ksn-draw

- Ascii Diagramを編集するツール
- Kotlin Multiplatform
  - The Elm Architecture
  - R-Tree
- Material Theme

Compose Multiplatform Framework

Desktop app

Compose for Desktop provides a declarative and reactive approach to creating desktop user interfaces with Kotlin. Combine composable functions to build your user interface, and enjoy full tooling support from your IDE and build system – no XML or templating language required.

Compose for Desktop targets the JVM, and supports high-performance, hardware-accelerated UI rendering on all major desktop platforms (macOS, Windows, and Linux) by leveraging the powerful native Skia graphics library. Native application distributions can be created with a single click.

Windows

macOS

Linux

Web app

Compose for Web allows you to build reactive user interfaces for the web in Kotlin, using the concepts and APIs of Jetpack Compose to express the state, behavior, and logic of your application.

Compose for Web provides multiple ways of declaring user interfaces in Kotlin code, allowing you to have full control over your website layout with a declarative DOM API.

Web

Compose Multiplatform Framework

Compose for Desktop targets the JVM, and supports high-performance, hardware-accelerated UI rendering on all major desktop platforms (macOS, Windows, and Linux) by leveraging the powerful native Skia

Compose for Web provides multiple ways of declaring user interfaces in Kotlin code, allowing you to have full control over your website layout with a declarative DOM API.

Compose Multiplatform Framework

- Compose Multiplatform 1.2 Is Out
  - https://blog.jetbrains.com/kotlin/2022/10/compose-multiplatform-1-2-is-out/

- 最新は1.2.1

Source

- Jetbrains/compose-jb
- Jetbrains/androidx
- Jetbrains/skiko

Jetbrains/compose-jb

1.2.1

- Compiler 1.3.2
- Runtime 1.2.1
- UI 1.2.1
- Foundation 1.2.1
- Material 1.2.1
- Material3 1.0.0-alpha14

1.3.0-beta

- 1.2.1 -> 1.3.1
- 1.0.0-alpha14 -> 1.0.1

Jetbrains/androidx

- Forked from androidx/androidx

- Compose Multiplatformの実装はcomposeディレクトリ以下に固まっている
- 大体ここのソースから何が動作するか判断できる

Jetbrains/skiko

- Skia for Kotlin

- Kotlin/JS + WebAssembly in browsers

- Multiplatformな機能も提供してくれている

設定

gradle.properties

```
org.jetbrains.compose.experimental.jscanvas.enabled=true
```

Sample

- seisuke/compose_multiplatform_sample_js_canvas

- seisuke/ksn-draw

- JetBrains/compose-jb/tree/master/experimental/examples

設定

build.gradle.kts

```
sourceSets {
  val jsMain by getting {
        dependencies {
            //implementation(compose.web.core)
            implementation(compose.ui)
            implementation(compose.foundation)
            implementation(compose.material)
            implementation(compose.runtime)
        }
    }
}
```

設定

build.gradle.kts

```
compose.experimental {
    web.application {}
}
```

制約

- Index.html
  - skiko.js
  - projectname.js
  - < canvas />
- Window -> ComposeWindow
  - // TODO: generalize me.
  - // TODO: make it public when we stabilize it after implementing it for uikit and js
  - Canvasをひとつ配置してする前提

Sampleで使っている機能

- Ksn-draw
  - Font Loading
  - Clipboard
  - Material/Snackbar
- DOMとの連携
- Lazyリスト
- Shader

Font Loading

- Skiko💓
  - FontManager

Clipboard

- Skiko💓
  - ClipBoardManager

Snackbar

- Kotlin/JSにはDialogやPopupの実装がない
- Desktopにはある
- DropdownMenuなどもつかえない

DOMとの連携

DomAndCanvasSample.kt

```
@Composable
fun DomAndCanvas() {
    var text by remember { mutableStateOf("text") }
    TextField(
        value = text,
        onValueChange = { text = it }
    )
    LaunchedEffect(Unit) {
        renderComposable(rootElementId = "root") {
            Div { org.jetbrains.compose.web.dom.Text(text) }
        }
    }
}
```

Shader

- AGSL
- Skiko💓
  - FpsCounter
  - currentNanoTime
- awaitFrameは使えない
- https://www.droidcon.com/2022/09/29/creative-coding-with-compose/
- https://github.com/CuriousNikhil/k5-compose

AGSL

```
private const val agsl = """
    uniform float2 iResolution;
    uniform float iTime;
    uniform shader content;
    vec4 main(vec2 FC) {
      vec4 o = vec4(0);
      vec2 p = vec2(0), c=p, u=FC.xy*2.-iResolution.xy;
      float a;
      for (float i=0; i<4e2; i++) {
        a = i/2e2-1.;
        p = cos(i*2.4+iTime+vec2(0,11))*sqrt(1.-a*a);
        c = u/iResolution.y+vec2(p.x,a)/(p.y+2.);
        o += (cos(i+vec4(0,2,4,0))+1.)/dot(c,c)*(1.-p.y)/3e4;
      }
      return o;
    }
    """
```

AGSL

https://android-developers.googleblog.com/2022/02/first-preview-android-13.html#:~:text=Android%20Graphics%20Shading%20Language

Android 13 adds support for programmable RuntimeShader objects, with behavior defined using the Android Graphics Shading Language (AGSL). AGSL shares much of its syntax with GLSL, but works within the Android rendering engine to customize painting within Android's canvas as well as filtering of View content. Android internally uses these shaders to implement ripple effects, blur, and stretch overscroll, and Android 13 enables you to create similar advanced effects for your app.

AGSL? SKSL?

https://chethaase.medium.com/agsl-made-in-the-shade-r-7d06d14fe02a

Android Graphics Shading Language (AGSL) is the language used to write shaders for Android. AGSL is essentially SkSL, which stands for Skia Shading Language, where Skia is the rendering engine for Android (among other client platforms, including Chrome). We renamed SkSL to AGSL to simplify the API essentially to avoid having to explain what Skia is in the middle of an Android API surface (which is exactly what I'm having to do here, but it's a bit better in an article than it would be in the middle of API reference docs).

Lazyリスト

- ふつうに動く
- Scrollbarがないことは考慮が必要

機能的に足りない部分

- 日本語入力があやしい☠️
- Text Selectableが使えない☠️
- Scrollbarが使えない
- Popupが使えない
- UnicodeBlocksが無い
- Layout Previewが動かない
- Layout Inspectorが欲しい

Conclusion

- テキスト入力まわりさえよくなれば、現状のKotlin/JSのCompose UIのみでアプリを作れないことはなさそう。-> 誰かContribute🤷
- 1.3.0-betaでMaterial 3を調査してみたい
- おもしろいので触ってみてKotlin Multiplatformのライブラリ増やそう