Estructura jerárquica del proyecto

Los proyectos multiplataforma de Kotlin admiten estructuras jerárquicas de conjuntos de fuentes. Esto significa que puede organizar una jerarquía de conjuntos de fuentes intermedias para compartir el código común entre algunos, pero no todos, los objetivos compatibles. El uso de conjuntos de fuentes intermedias te ayuda a:

- Proporcione una API específica para algunos objetivos. Por ejemplo, una biblioteca puede agregar API específicas nativas en un conjunto de fuentes intermedias para los objetivos Kotlin/Native, pero no para los objetivos Kotlin/JVM.
- Consuma una API específica para algunos objetivos. Por ejemplo, puede beneficiarse de una rica API que la biblioteca multiplataforma Kotlin proporciona para algunos objetivos que forman un conjunto de fuentes intermedias.
- Utilice bibliotecas dependientes de la plataforma en su proyecto. Por ejemplo, puedes acceder a las dependencias específicas de iOS desde el conjunto de fuentes intermedias de iOS.

La cadena de herramientas Kotlin garantiza que cada conjunto de fuentes tenga acceso solo a la API que está disponible para todos los objetivos a los que se compila ese conjunto de fuentes. Esto evita casos como el uso de una API específica de Windows y luego la compilación en macOS, lo que resulta en errores de vinculación o un comportamiento indefinido en tiempo de ejecución.

La forma recomendada de configurar la jerarquía de conjuntos de origen es utilizar la plantilla de jerarquía predeterminada. La plantilla cubre los casos más populares. Si tienes un proyecto más avanzado, puedes configurarlo manualmente. Este es un enfoque de nivel más bajo: es más flexible, pero requiere más esfuerzo y conocimiento.

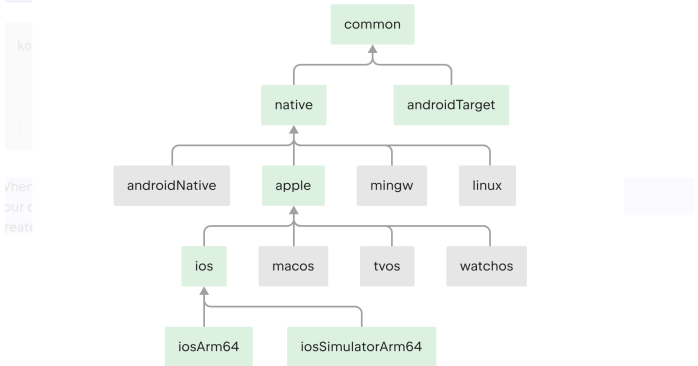
Plantilla de jerarquía predeterminada

A partir de Kotlin 1.9.20, el complemento Kotlin Gradle tiene una plantilla de jerarquía predeterminada integrada. Contiene conjuntos de fuentes intermedias predefinidas para algunos casos de uso populares. El complemento configura esos conjuntos de fuentes automáticamente en función de los objetivos especificados en su proyecto.

Considere el siguiente ejemplo:

```
kotlin {
    androidTarget()
    iosArm64()
    iosSimulatorArm64()
}
```

Cuando declaras los objetivos androidTarget, iosArm64 e iosSimulatorArm64 en tu código, el complemento Kotlin Gradle encuentra conjuntos de fuentes compartidas adecuados de la plantilla y los crea para ti. La jerarquía resultante se ve así:



En realidad, los conjuntos de fuentes verdes se crean y están presentes en el proyecto, mientras que los grises de la plantilla predeterminada se ignoran. El complemento Kotlin Gradle no ha creado el conjunto de fuentes de watchos, por ejemplo, porque no hay objetivos de watchOS en el proyecto.

Si agrega un objetivo de watchOS, como watchosArm64, se crea el conjunto de fuentes de watchos, y el código de los conjuntos de fuentes de Apple, nativos y comunes también se compila en watchosArm64.

El complemento Kotlin Gradle crea accesores de tipo seguro para todos los conjuntos de fuentes a partir de la plantilla de jerarquía predeterminada, para que pueda hacer referencia a ellos sin obtener o creando construcciones en comparación con la configuración manual:

```
 kotlin {
    androidTarget()
    iosArm64()
    iosSimulatorArm64()

    sourceSets {
        iosMain.dependencies {
            implementation("org.jetbrains.kotlinx:kotlinx-coroutines-
core:1.7.3")
        }
    }
}
```

**En este ejemplo, los conjuntos de Apple y fuentes nativas
se compilan solo en los objetivos iosArm64 e iosSimulatorArm64. A pesar de sus nombres, tienen acceso a la API completa de iOS. Esto puede ser contrario a la intuición para conjuntos de fuentes como nativos, ya que es de esperar que solo se pueda acceder a las API disponibles en todos los objetivos nativos en este conjunto de fuentes. Este comportamiento puede cambiar en el futuro.**

Configuración adicional

Es posible que tenga que hacer ajustes en la plantilla de jerarquía predeterminada. Si previamente ha introducido fuentes intermedias manualmente con las llamadas dependsOn, se cancela el uso de la plantilla de jerarquía predeterminada y conduce a esta advertencia:

```
 La plantilla de jerarquía Kotlin
predeterminada no se aplicó a '<nombre del proyecto>':

Los bordes explícitos .dependsOn() se configuraron para los siguientes
conjuntos de fuentes:

[<... nombres de los conjuntos de fuentes con depende de los bordes
configurados manualmente... >]

Considere eliminar las llamadas de dependientes o deshabilitar la plantilla
predeterminada agregando

'kotlin.mpp.applyDefaultHierarchyTemplate=false'

A tus propiedades de gradle

Más información sobre las plantillas jerárquicas: https://kotl.in/hierarchy-
template
```

Para resolver este problema, configure su proyecto haciendo una de las siguientes acciones:

- Reemplace su configuración manual con la plantilla de jerarquía predeterminada
- Crear conjuntos de fuentes adicionales en la plantilla de jerarquía predeterminada
- Modificar los conjuntos de fuentes creados por la plantilla de jerarquía predeterminada

Reemplazar una configuración manual

Caso. Todos sus conjuntos de fuentes intermedias están actualmente cubiertos por la plantilla de jerarquía predeterminada.

Solución. Elimine todas las llamadas manuales de dependsOn() y los conjuntos de fuentes mediante la creación de construcciones. Para comprobar la lista de todos los conjuntos de fuentes predeterminados, consulte la plantilla de jerarquía completa.

Creación de conjuntos de fuentes adicionales

Caso. Desea agregar conjuntos de origen que la plantilla de jerarquía predeterminada aún no proporciona, por ejemplo, uno entre un objetivo de macOS y un objetivo JVM.

Solución:

- Vuelva a aplicar la plantilla llamando explícitamente a applyDefaultHierarchyTemplate().
- Configure conjuntos de fuentes adicionales manualmente usando dependsOn():

```
kotlin {
    jvm()
    macosArm64()
    iosArm64()
    iosSimulatorArm64()

    // Apply the default hierarchy again. It'll create, for example, the iosMain
source set:
    applyDefaultHierarchyTemplate()

    sourceSets {
        // Create an additional jvmAndMacos source set:
        val jvmAndMacos by creating {
            dependsOn(commonMain.get())
        }

        macosArm64Main.get().dependsOn(jvmAndMacos)
        jvmMain.get().dependsOn(jvmAndMacos)
    }
}
```

```
kotlin {
    jvm()
    macosArm64()
    iosArm64()
    iosSimulatorArm64()

    // Apply the default hierarchy again. It'll create, for example, the iosMain
source set:
    applyDefaultHierarchyTemplate()

    sourceSets {
        // Create an additional jvmAndMacos source set:
        jvmAndMacos {
            dependsOn(commonMain.get())
        }
        macosArm64Main {
            dependsOn(jvmAndMacos.get())
        }
        jvmMain {
            dependsOn(jvmAndMacos.get())
        }
    }
}
```

Modificación de conjuntos de fuentes

Caso. Ya tiene los conjuntos de origen con exactamente los mismos nombres que los generados por la plantilla, pero compartidos entre diferentes conjuntos de objetivos en su proyecto. Por ejemplo, un conjunto de fuentes nativas principales solo se comparte entre los objetivos específicos del escritorio: linuxX64, mingwX64 y macosX64.

Solución. Actualmente no hay forma de modificar el valor predeterminado que depende de las relaciones entre los conjuntos de fuentes de la plantilla. También es importante que la implementación y el significado de los conjuntos de fuentes, por ejemplo, nativeMain, sean los mismos en todos los proyectos.

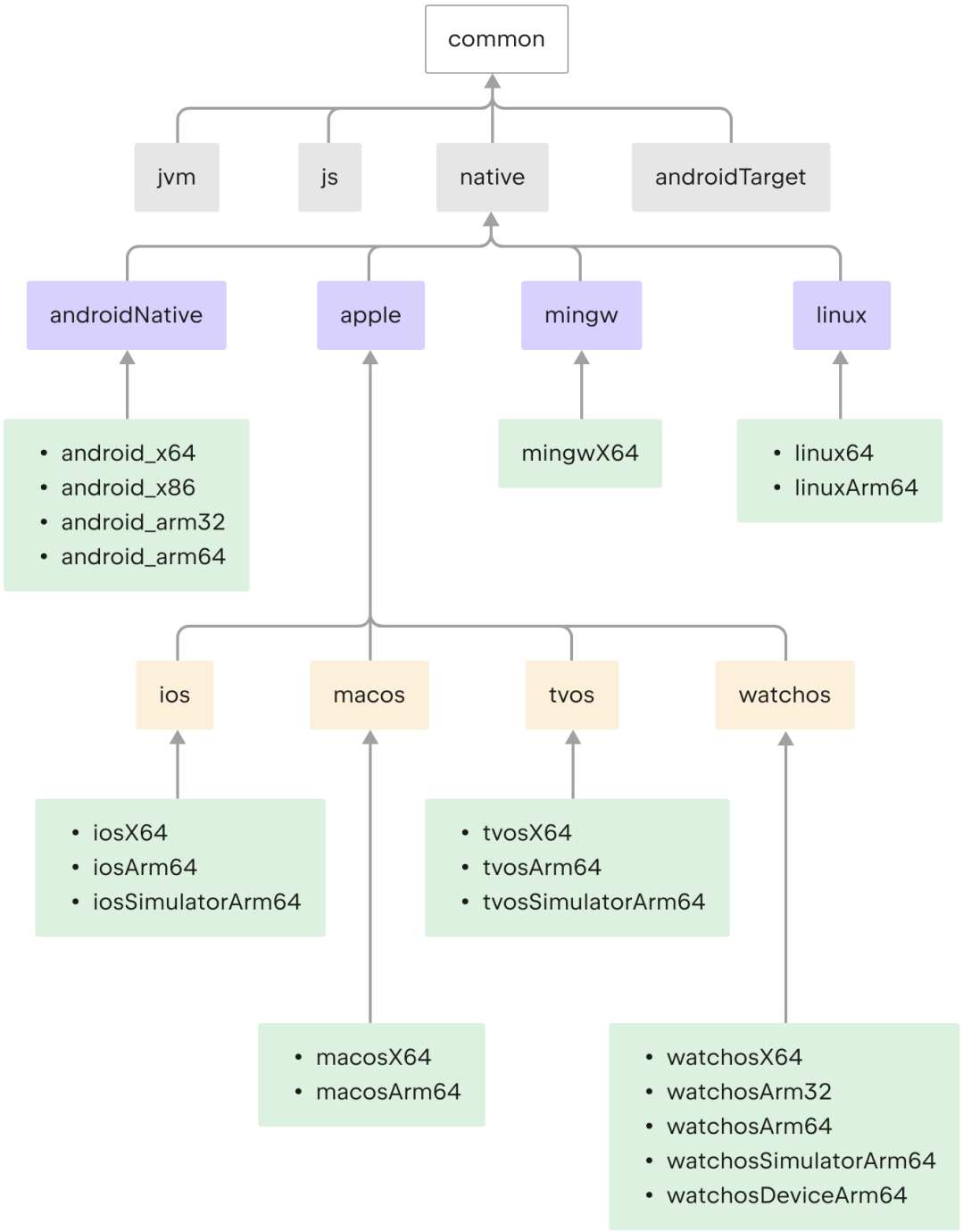
Sin embargo, todavía puedes hacer una de las siguientes acciones:

- Encuentre diferentes conjuntos de fuentes para sus propósitos, ya sea en la plantilla de jerarquía predeterminada o en los que se han creado manualmente.
- Opte por salir de la plantilla por completo agregando kotlin.mpp.applyDefaultHierarchyTemplate=false a su archivo gradle.properties y configure manualmente todos los conjuntos de fuentes.

Actualmente estamos trabajando en una API para crear sus propias plantillas de jerarquía. Esto será útil para proyectos cuyas configuraciones jerárquicas difieren significativamente de la plantilla predeterminada.

Esta API aún no está lista, pero si está ansioso por probarla, busque el bloque applyHierarchyTemplate {} y la declaración de KotlinHierarchyTemplate.default como ejemplo. Tenga en cuenta que esta API todavía está en desarrollo. Es posible que no se pruebe y pueda cambiar en futuras versiones.

Cuando declara los objetivos a los que se compila su proyecto, el complemento elige los conjuntos de fuentes compartidas en función de los objetivos especificados de la plantilla y los crea en su proyecto.



Actualmente estamos trabajando en una API para crear sus propias plantillas de jerarquía. Esto será útil para proyectos cuyas configuraciones jerárquicas difieren significativamente de la plantilla predeterminada.

Esta API aún no está lista, pero si está ansioso por probarla, busque el bloque applyHierarchyTemplate {} y la declaración de KotlinHierarchyTemplate.default como ejemplo. Tenga en cuenta que esta API todavía está en desarrollo. Es posible que no se pruebe y pueda cambiar en futuras versiones.

Configuración manual

Puede introducir manualmente una fuente intermedia en la estructura del conjunto de fuentes. Contendrá el código compartido para varios objetivos.

Por ejemplo, esto es lo que debes hacer si quieres compartir código entre objetivos nativos de Linux, Windows y macOS (linuxX64, mingwX64 y macosX64):

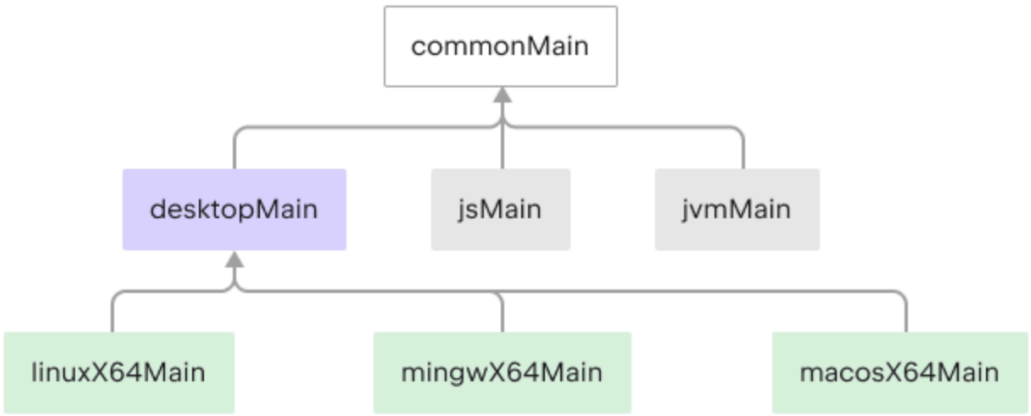
1. Agregue el conjunto de fuentes intermedias desktopMain, que contiene la lógica compartida para estos objetivos.

2. Especifique la jerarquía del conjunto de fuentes utilizando la relación dependsOn.

```
kotlin {
    linuxX64()
    mingwX64()
    macosX64()

    sourceSets {
        val desktopMain by creating {
            dependsOn(commonMain.get())
        }

        linuxX64Main.get().dependsOn(desktopMain)
        mingwX64Main.get().dependsOn(desktopMain)
        macosX64Main.get().dependsOn(desktopMain)
    }
}
```

```
kotlin {
    linuxX64()
    mingwX64()
    macosX64()

    sourceSets {
        desktopMain {
            dependsOn(commonMain.get())
        }
        linuxX64Main {
            dependsOn(desktopMain)
        }
        mingwX64Main {
            dependsOn(desktopMain)
        }
        macosX64Main {
            dependsOn(desktopMain)
        }
    }
}
```

La estructura jerárquica resultante tendrá el siguiente aspecto:



Puede tener un conjunto de fuentes compartidas para las siguientes combinaciones de objetivos:

- JVM o Android + JS + Nativo
- JVM o Android + Nativo
- JS + Nativo
- JVM o Android + JS
- Nativo

Actualmente, Kotlin no admite compartir un conjunto de fuentes para estas combinaciones:

- Varios objetivos de JVM
- Objetivos de JVM + Android
- Varios objetivos de JS

Si necesita acceder a las API específicas de la plataforma desde un conjunto de fuentes nativas compartidas, IntelliJ IDEA le ayudará a detectar declaraciones comunes que puede utilizar en el código nativo compartido. Para otros casos, utilice el mecanismo Kotlin de declaraciones esperadas y reales.