Kotlin Multiplatform Library Development

Russell Wolf
Feb 2, 2020

@Russhwolf ( or. )

Multiplatform Developer
at touchlab.co

Multiplatform Kotlin

Multiplatform Kotlin

- Compile common code to multiple targets
  - JVM, JS, Android, Desktop, iOS, Embedded, WASM
- Use platform-specific code to access platform APIs

Common Code

```
class CommonFoo {
  fun bar(list: List<String>) {
    list.forEach { println(it) }
  }
}
```

Platform-Specific Code

```
expect val platform: String

actual val platform = "Android"

actual val platform = "iOS"
```

Platform-Specific Code

```
interface Foo { ... }

class AndroidFoo : Foo { ... }

class IosFoo : Foo { ... }

class SwiftFoo : NsObject, Foo { ... }

class MockFoo : Foo { ... }
```

In the Beginning...

Kotlin/Native v0.6 is Here!

Posted on February 14, 2018 by Roman Belov

We are pleased to announce Kotlin/Native v0.6 (Valentine's Day release) of our toolchain. This is a major update, including the following features:

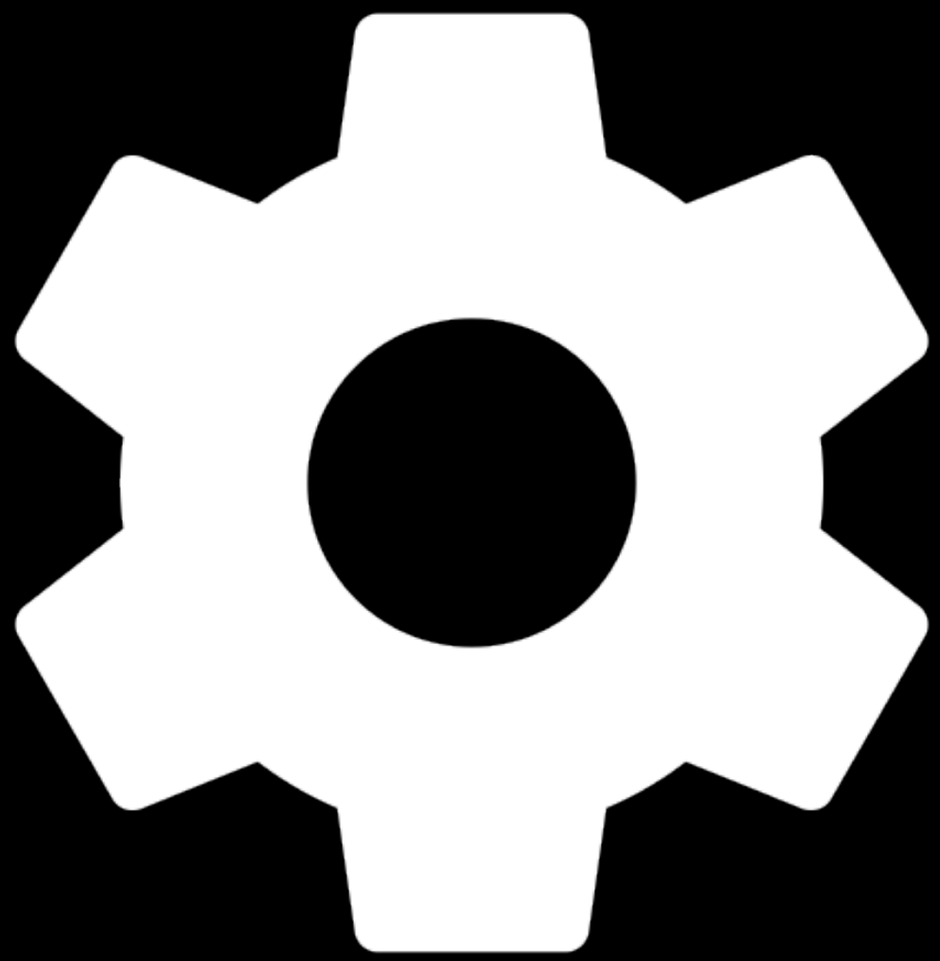
- Support for multiplatform projects in compiler and Gradle plugin
- Transparent Objective-C/Kotlin container classes interoperability

Kotlin/Native v0.7 released: smoother interop, frozen objects, optimisations and more.

Posted on April 27, 2018 by Nikolay Igotti

- Use Gradle native dependency model, allowing to use `.klib` as Maven artifacts

Multiplatform Settings



- Key-value storage based on platform APIs
- Operators and Property Delegates
- https://github.com/russhwolf/multiplatform-settings

Multiplatform Settings

```
interface Settings {
  fun putInt(key: String, value: Int)
}

class AndroidSettings(
  val delegate: SharedPreferences
) : Settings {
  override fun putInt(…) = delegate.putInt(…)
}

class AppleSettings(
  val delegate: NSUserDefaults
) : Settings {
  override fun putInt(…) = delegate.setInteger(…)
}
```

Multiplatform Settings

```
interface Settings {
  fun putInt(key: String, value: Int)
}

class JsSettings(
  val delegate: Storage = localStorage
) : Settings {
  override fun putInt(…) = delegate.set(…)
}

class JvmPreferencesSettings(
  val delegate: Preferences
) : Settings {
  override fun putInt(…) = delegate.putInt(…)
}
```

Multiplatform Settings

```
interface Settings {
  fun putInt(key: String, value: Int)
}

class MockSettings(
  val delegate: MutableMap<String, Any>
) : Settings {
  override fun putInt(…) = delegate.set(…)
}
```

Multiplatform Settings

```
operator fun Settings.set(
  key: String,
  value: Int
): Unit = putInt(key, value)
```

```
settings["a"] = 3


fun Settings.int(
  key: String? = null,
  defaultValue: Int = 0
): ReadWriteProperty<Any?, Int> = ...

var a by settings.int("a")
```

Stories & Lessons

Expect/Actual vs Interface

```
expect class Settings {
  fun putInt(key: String, value: Int)
}

actual class Settings(
  val delegate: SharedPreferences
) {
  actual fun putInt(…) = delegate.putInt(…)
}

actual class Settings(
  val delegate: NSUserDefaults
) {
  actual fun putInt(…) = delegate.setInteger(…)
}
```

Expect/Actual vs Interface

```
interface Settings {
  fun putInt(key: String, value: Int)
}

expect class PlatformSettings: Settings {
  override fun putInt(key: String, value: Int)
}

actual class PlatformSettings(
  val delegate: SharedPreferences
) : Settings {
  actual fun putInt(…) = delegate.putInt(…)
}

actual class PlatformSettings(…) { … }
```

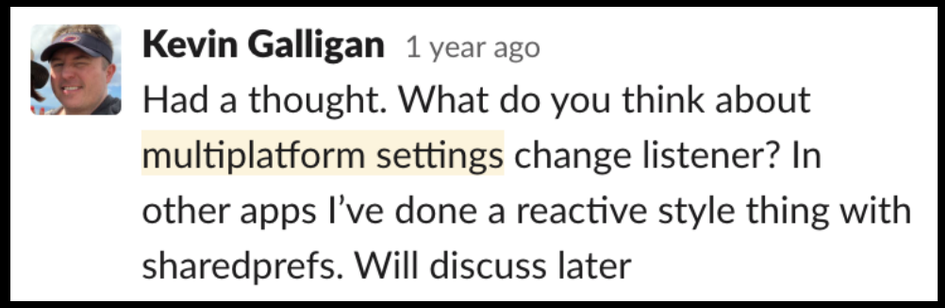
Expect/Actual vs Interface

```
interface Settings {
  fun putInt(key: String, value: Int)
}

class AndroidSettings(
  val delegate: SharedPreferences
) : Settings {
  override fun putInt(…) = delegate.putInt(…)
}

class AppleSettings(
  val delegate: NSUserDefaults
) : Settings {
  override fun putInt(…) = delegate.setInteger(…)
}
```

Listener APIs



Listener APIs

- `SharedPreferences.OnSharedPreferenceChangeListener`
  - Passes key to callback
  - Might get called for repeated values
- `NSNotificationCenter NSUserDefaultsDidChangeNotification`
  - Can't tell what changed

Listener APIs

```
val current = delegate.all[key]
if (prev != current) {
    callback.invoke()
    prev = current
}
```

```
val current = delegate.objectForKey(key)
if (prev != current) {
    callback.invoke()
    prev = current
}
```

Listener APIs

```
val current = delegate.all[key]
if (prev != current) {
    callback.invoke()
    prev = current
}
```

```
val current = delegate.objectForKey(key)
if (prev != current) {
    callback.invoke()
    prev = current
}

???
```

Listener APIs

```
val current = delegate.all[key]
if (prev != current) {
    callback.invoke()
    prev = current
}
```

```
val current = delegate.objectForKey(key)
if (prev != current) {
    callback.invoke()
    prev = current
}
```

???

Listener APIs

```
interface ObservableSettings :
  Settings { … }

class AndroidSettings(…) :
  ObservableSettings { … }
```

```
class JsSettings(…) :
  Settings { … }
```

JVM Implementations

r4zzz4k commented on Mar 1, 2019

What do you think about plain JVM? I believe `Properties` should suite well as a backing storage.

Add onModify callback to JVM implementation #29

Closed **gergelydaniel** wants to merge 1 commit into `russhwolf:master` from `gergelydaniel:master`

eskatos commented on Sep 20, 2019

On the JVM platform Preferences might be a better fit than `Properties`. `Preferences` are stored in a standard location and can notify changes to listeners.

JVM Implementations

```
class JvmPropertiesSettings(
  val delegate: Properties
) : Settings {
  override fun putInt(…) = delegate.setProperty(…)
}
```

```
class JvmPreferencesSettings(
  val delegate: Preferences
) : ObservableSettings {
  override fun putInt(…) = delegate.putInt(…)
}
```

Continuous Integration

- Using Azure Pipelines to access Mac, Linux, Windows hosts
- Build common code for all platforms

```
presets.forEach {
  if (it.name == "jvmWithJava") return@forEach
  if (targets.findByName(it.name) == null) {
    targetFromPreset(it)
  }
}
```

New Apple Targets

- Original targets: iosArm64, iosX64
- Added later: macosX64, iosArm32
- New in Kotlin 1.3.60: watchosArm32, watchosArm64, watchosX86, tvosArm64, tvosX64

New Apple Targets

```
fun putLong(key: String, value: Long): Unit =
  delegate.setInteger(value.convert(), key)
```

New Apple Targets

```
fun putLong(key: String, value: Long): Unit =
  delegate.setInteger(value.convert(), key)
```

New Apple Targets

```
fun putLong(key: String, value: Long): Unit =
  delegate.setLong(value, key)

expect fun NSUserDefaults.setLong(value: Long, forKey: String)
```

```
// 64-bit
actual fun NSUserDefaults.setLong(value: Long, forKey: String) =
  setInteger(value, forKey)

// 32-bit
actual fun NSUserDefaults.setLong(value: Long, forKey: String) =
  setObject(value.toString(), forKey)
```

New Apple Targets

```
fun putLong(key: String, value: Long): Unit =
  delegate.setLong(value, key)

expect fun NSUserDefaults.setLong(value: Long, forKey: String)
```

```
// 64-bit
actual fun NSUserDefaults.setLong(value: Long, forKey: String) =
  setInteger(value, forKey)

// 32-bit
actual fun NSUserDefaults.setLong(value: Long, forKey: String) =
  setObject(value.toString(), forKey)
```

New Apple Targets

```
fun putLong(key: String, value: Long): Unit =
  delegate.setLong(value, key)

expect fun NSUserDefaults.setLong(value: Long, forKey: String)
```

```
// 64-bit
actual fun NSUserDefaults.setLong(value: Long, forKey: String) =
  setInteger(value, forKey)

// 32-bit
actual fun NSUserDefaults.setLong(value: Long, forKey: String) =
  setObject(value.toString(), forKey)
```

New Apple Targets

```
fun putLong(key: String, value: Long): Unit =
  delegate.setLong(value, key)

expect fun NSUserDefaults.setLong(value: Long, forKey: String)
```

```
// 64-bit
actual fun NSUserDefaults.setLong(value: Long, forKey: String) =
  setInteger(value, forKey)

// 32-bit
actual fun NSUserDefaults.setLong(value: Long, forKey: String) =
  setObject(value.toString(), forKey)
```

Other Notes

- Kotlin/Native has no version compatibility! Keep up-to-date to support clients
- Gradle is complicated
  - https://kotlinlang.org/docs/reference/building-mpp-with-gradle.html
- Easier than last year

Coming Soon

- Maven Central
- Flow extensions
- Serialization extensions
- On-device unit tests
- Other Platforms/Implementations
  - Windows registry
  - Linux?

What about other stuff?

- Jetbrains
  - Coroutines
  - Serialization
  - Ktor Client
  - IO
  - AtomicFU
  - ...
- Community
  - SqlDelight
  - Stately
  - CoroutineWorker
  - Reaktive
  - Island Time
  - ...

Yours?

Strategies

- Wrap platform APIs
  - Implementation already exists!
  - Takes effort to equalize behavior/API

Strategies

- Pure-Kotlin
    - Effort to create new implementation
    - All common = all platforms

The time is now!

Thanks!

- Questions?
  - @RussHWolf ( or )
- Multiplatform Settings https://github.com/russhwolf/multiplatform-settings
- Building MPP with Gradle documentation https://kotlinlang.org/docs/reference/building-mpp-with-gradle.html
- Other community libraries https://github.com/AAkira/Kotlin-Multiplatform-Libraries

Joining the Kotlin Multiplatform Team!

touchlab.co

@touchlabhq