Kotlin/JS + React

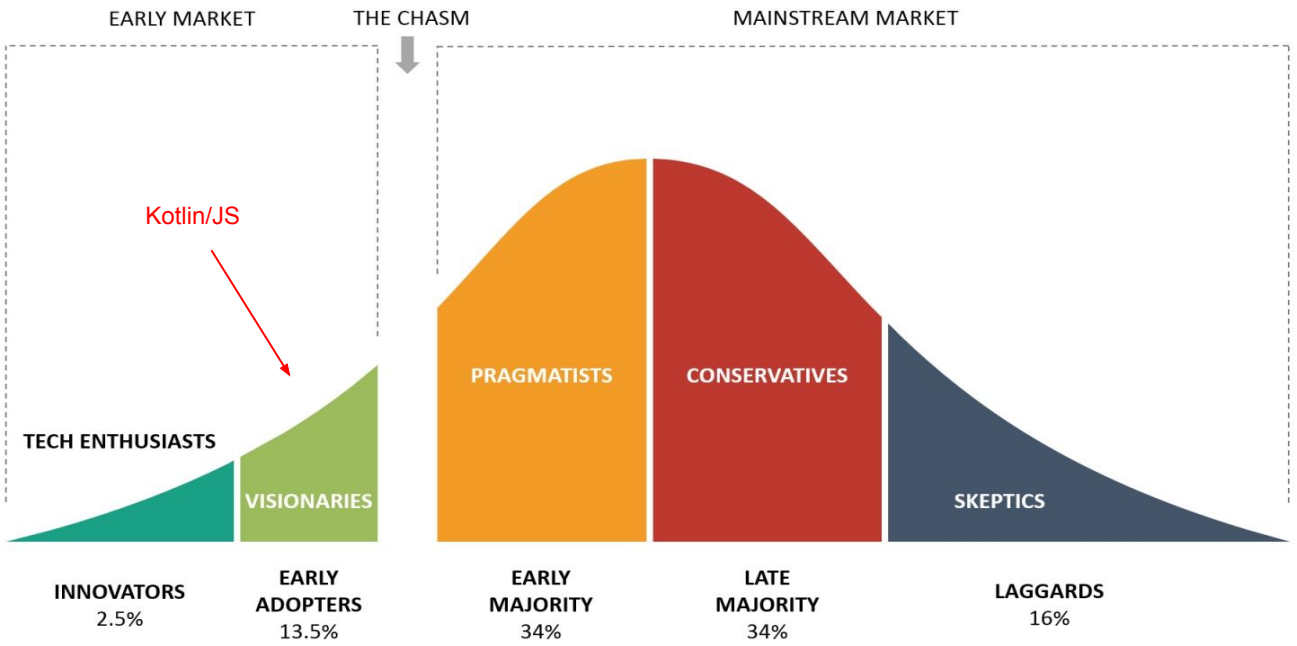
Introduction

https://gitlab.com/jcore_nl/kotlinjs-react

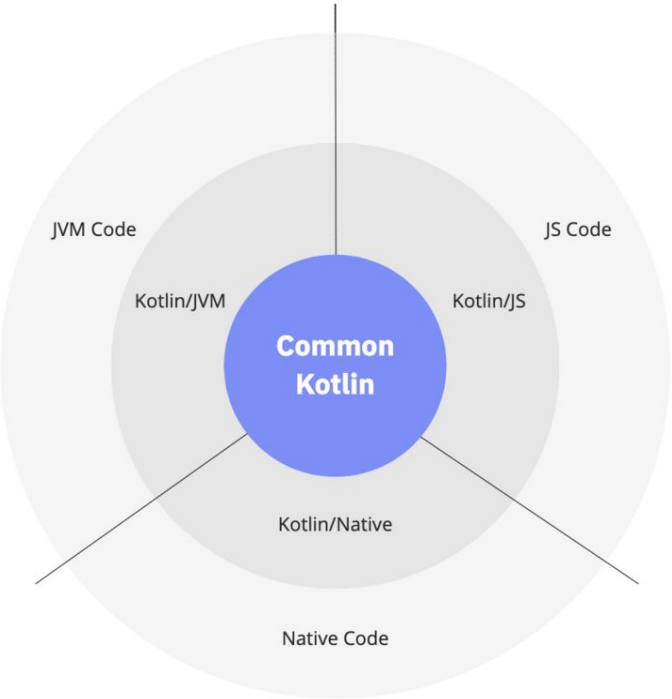
Why Kotlin/JS?

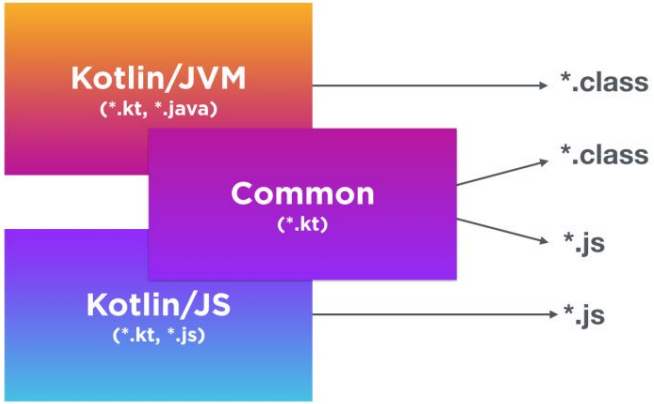
- Access to all of Kotlins features
- Partially shared infrastructure
- Easy to learn

Why not Kotlin/JS?



JetBrains' vision





Wrappers

JetBrains/**kotlin-wrappers**

Kotlin wrappers for popular JavaScript libraries

62 Contributors | 11 Issues | 975 Stars | 143 Forks

Dukat

Kotlin/**dukat**

Converter of <any kind of declarations> to Kotlin external declarations

11 Contributors

189 Used by

400 Stars

34 Forks

Roadmap

Kotlin roadmap

Edit page Last modified: 07 December 2021

Kotlin/JS

- Make the new JS IR backend Stable ↗
- JS IR BE: Add an ability to generate separate JS files for each module ↗
- ⏸ JS: support ES6 as compilation target ↗
- ⏸ Improve Dukat support ↗

Kotlin/Wasm

- Implement an experimental version of Kotlin/Wasm compiler backend ↗

Kotlin/JS

build.gradle.kts

```
plugins {
   kotlin("js") version "1.6.0"
}
```

Basics

HTML

```html
<div>
    <button class="button">
        Press me!
    </button>
</div>
```

Kotlin/JS

```kotlin
div {
    button {
        attrs.className = "button"
        +"Press me!"
    }
}
```

Basics

Multiple attributes

HTML

```html
<div>
    <button id="some-id" class="button">
        Press me!
    </button>
</div>
```

Kotlin/JS

```kotlin
div {
    button {
        attrs {
            id = "some-id"
            className = "button"
        }
        +"Press me!"
    }
}
```

Basics

Calling functions

HTML

```
<div>
    <button onclick="doSomething()">
        Press me!
    </button>
</div>
```

Kotlin/JS

```
div {
    button {
        attrs.onClick = { doSomething() }
        +"Press me!"
    }
}
```

Basics

Inline styling

build.gradle.kts

```
dependencies {
      implementation(npm("styled-components", "~5.3.3"))
      implementation("org.jetbrains.kotlin-wrappers:kotlin-styled:5.3.3-pre.286-kotlin-1.6.10"
)
}
```

Basics

Inline styling

HTML

```html
<div style="display: flex">
    <button>
        Press me!
    </button>
</div>
```

Kotlin/JS

```kotlin
styledDiv {
    css {
        display = Display.flex
    }
    button {
        +"Press me!"
    }
}
```

JS

```
document.getElementById("my-element")

console.log("Hello world!")

window.alert("Hello world!")

localStorage.getItem("theme")

js("throw Error('Hello world!')")
```

Exceptions

Javascript Error is mapped to Kotlin Throwable, which means the following will work:

```
try {
    js("throw Error('Message')")
} catch (e: Throwable) {
    // Handle error
}
```

Testing

```kotlin
import kotlin.test.AfterTest
import kotlin.test.BeforeTest
import kotlin.test.Test
import kotlin.test.assertEquals

class TestClass {

   @BeforeTest
   fun setup() {

   }

   @Test
   fun test() {
       assertEquals(2 * 2, 4)
   }

   @AfterTest
   fun cleanup() {

   }
}
```

Adapters

build.gradle.kts

```
dependencies {
      implementation(npm("awesome-notifications", "3.1.2"))
}
```

Adapters

```
@file:JsModule("awesome-notifications")
@file:JsNonModule

package components.adapters

@JsName("default")
external class AwesomeNotifications {
    fun tip(msg: String)
    fun info(msg: String)
    fun success(msg: String)
    fun warning(msg: String)
    fun alert(msg: String)
}
```

Adapters

```
@file:JsModule("awesome-notifications")
@file:JsNonModule

package components.adapters

@JsName("default")
external class AwesomeNotifications {
    fun tip(msg: String)
    fun info(msg: String)
    fun success(msg: String)
    fun warning(msg: String)
    fun alert(msg: String)
}
```

Similar to Java's native keyword, the external keyword is used to access methods implemented in a language other than Kotlin

Adapters

```
@file:JsModule("awesome-notifications")
@file:JsNonModule

package components.adapters

@JsName("default")
external class AwesomeNotifications {
    fun tip(msg: String)
    fun info(msg: String)
    fun success(msg: String)
    fun warning(msg: String)
    fun alert(msg: String)
}
```

Indicates that this external declaration must be imported from a Javascript module

Adapters

```
@file:JsModule("awesome-notifications")
@file:JsNonModule

package components.adapters

@JsName("default")
external class AwesomeNotifications {
    fun tip(msg: String)
    fun info(msg: String)
    fun success(msg: String)
    fun warning(msg: String)
    fun alert(msg: String)
}
```

Some Javascript libraries are distributed as both a standalone downloadable piece of Javascript and as a module available as an npm package. To tell the Kotlin compiler to accept both, add @JsNonModule to a @JsModule annotation.

API Calls

```
plugins {
   kotlin("plugin.serialization") version "1.6.0"
}

dependencies {
   implementation(org.jetbrains.kotlinx:kotlinx-serialization-json:1.3.2" ))
}
```

API Calls

Fetch API

```
private suspend fun get(url: String): Response =
    window
        .fetch(url, RequestInit("GET"))
        .await()
```

API Calls

Serializable

```
@Serializable
data class MySerializable(
    val valueOne: String,
    val valueTwo: String,
)
```

API Calls

Deserialization

```
private fun parse(json: String): List<MySerializable>  =
    val strategy = ListSerializer(MySerializable.serializer())
    try {
        Json.decodeFromString(strategy , json)
    } catch (t: Throwable) {
        // Handle error
    }
```

Config

build.gradle.kts

```
kotlin {
   js(LEGACY) {
       binaries.executable()
       browser {
           commonWebpackConfig {
               devtool = CHEAP_SOURCE_MAP
               devServer = devServer?.copy(
                   open = true,
                   port = 8888,
                   proxy = mutableMapOf("/api" to
"http://localhost:8080/")
               )
           }
       }
   }
}
```

Config

build.gradle.kts

```kotlin
kotlin {
    js(LEGACY) {
        browser {
            commonWebpackConfig {
                configDirectory = File("./webpack")
            }
        }
        binaries.executable()
    }
}
```

Config

webpack/webpack.config.js

```
;(function (config) {
    config.devServer = config.devServer || {}
    config.devServer.port = 8888
    config.devServer.hot = false
    config.devServer.proxy = {
        '/api': {
            target: 'http://localhost:8080',
            secure: false,
        }
    }

    config.devtool = 'cheap-source-map'

})(config);
```

React

build.gradle.kts

```
dependencies {
    implementation(npm("react", "17.0.2"))
    implementation(npm("react-is", "17.0.2"))
    implementation(npm("react-dom", "17.0.2"))
    implementation(npm("react-router-dom", "6.2.1"))
    implementation("org.jetbrains.kotlin-wrappers:kotlin-react:17.0.2-pre.286-kotlin-1.6.10")
    implementation("org.jetbrains.kotlin-wrappers:kotlin-react-dom:17.0.2-pre.286-kotlin-1.6.10")
    implementation("org.jetbrains.kotlin-wrappers:kotlin-react-router-dom:6.2.1-pre.286-kotlin-1.6.10")
}
```

Reusable components

Class component

```
class ComponentA: RComponent<Props, State>() {
    override fun RBuilder.render() {
        div {
            button {
                +"Press me!"
            }
        }
    }
}

class ComponentB: RComponent<Props, State>() {
    override fun RBuilder.render() {
        child(ComponentA::class) {}
    }
}
```

Functional component

```
val ComponentA = fc<Props> {
    div {
        button {
            +"Press me!"
        }
    }
}

val ComponentB = fc<Props> {
    child(ComponentA)
}
```

Properties

```
external interface ComponentAProps: Props {
    var value: String
}

val ComponentA = fc<ComponentAProps> { props ->
    div {
        button {
            +props.value
        }
    }
}

val ComponentB = fc<Props> {
    MyFirstComponent {
        attrs.value = "Press me!"
    }
}
```

React rendering

Rendering is the process of React asking your components to describe what they want their section of the UI to look like now, based on the current combination of props and state.

React rendering

During the rendering process, React will start at the root of the component tree and loop downwards to find all components that have been flagged as needing updates.

React rendering

```
Components are flagged for update when:

-  The parent component is
-  The state of a component has changed
```

React rendering

Renders must be pure!

Side effects must be performed in hooks.

useState

The useState hook lets store state
in a functional component.

```
val MyComponent = fc<Props> {
   var counter by useState(0)

   div {
       button {
           attrs.onClick = { counter++ }
           +"Press me to increase $counter!"
       }
   }
}
```

useState

The useState hook lets store state
in a functional component.

```
val MyComponent = fc<Props> {
   var counter = 0

   div {
       button {
           attrs.onClick = { counter++ }
           +"Press me to increase $counter!"
       }
   }
}
```

useEffect

The useEffect hook lets you perform side effects in a functional component.

```
var date: Date by useState(Date())

// … User input mutating date

useEffect(date) {
   MainScope().launch {
       someVariable = httpClient.get(date)
   }
}
```

useEffect

```
// Fires upon initial render
useEffectOnce {
    …
}

// Fires each render
useEffect {
    …
}

// Fires when state of the dependency has changed
useEffect(date) {
    …
}
```

useContext

The useContext hook lets you pass context to a functional component, without passing it through properties.

AppConfig.kt

```
val Toaster = createContext(AwesomeNotifications())
```

MyComponent.kt

```
val MyComponent = fc<Props> {
    val toaster = useContext(Toaster)
    toaster.info("Hi!")
}
```

Error boundary

In contrast to the first slide on Exceptions, this does not work:

```
// Parent component
try {
   child(Component)
} catch (e: Throwable) {
   // Handle error
}


val Component = fc<Props> {
   button {
       attrs.onClick = { throw NumberFormatException("Woopsie!")
}
       +"Click me!"
   }
}
```

Error boundary

```
external interface ErrorBoundaryState : State {
   var error: Throwable?
}

class ErrorBoundary : RComponent<PropsWithChildren, ErrorBoundaryState>() {

   override fun componentDidCatch(error: Throwable, info: ErrorInfo) {
       setState {
           this.error = error
       }
   }

   override fun RBuilder.render() {
       state.error?.also {

           div {
              +"An unexpected fatal error has occurred:  ${state.error}."
           }
       }

       props.children()
   }
}
```

Error boundary

```
child(ErrorBoundary::class) { child(MyComponent) }
```