Kotlin Fest 2019

Kotlin Multiplatform Project入門

あらたに　あきら
荒谷　光

@_a_akira

About me

Akira Aratani

M3, Inc.

@_a_akira

AAkira

https://aakira.app

aakira.studio

https://aakira.studio

✈️📷🎾

🇺🇸🇬🇧🇫🇷🇮🇹🇩🇪🇦🇹🇨🇿🇸🇰🇵🇱🇭🇺🇫🇮🇮🇸🇦🇺🇵🇪🇲🇾🇮🇩🇵🇼🇰🇭🇧🇴🇹🇼🇳🇱🇹🇭🇲🇲🇵🇭🇸🇬

Kotlin Multiplatform Projectと私

- **Kotlin/Nativeチュートリアル Android, iOS編**
  http://aakira.app/blog/2018/10/kotlin-native
- **Kotlin Multiplatform構想 ~今やる理由編~**
  https://aakira.app/blog/2018/12/kotlin-mpp-reason
- **Kotlin Multiplatform構想 ~設計編~**
  https://aakira.app/blog/2018/12/kotlin-mpp-architecture
- **Kotlin Multiplatform環境でKotlin SerializationとAndroid ExtensionsのParcelize Annotationを使う**
  http://aakira.app/blog/2018/12/kotlin-mpp-android-parcelable
- **NapierというKotlin Multiplatform用のログライブラリを作った**
  https://aakira.app/blog/2019/02/napier

Kotlin Multiplatform Projectと私

Timber likeなKotlin mpp用ライブラリ

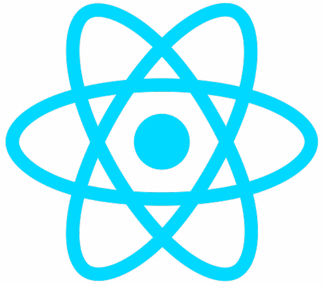
https://github.com/AAkira/Napier

Agenda

1. Kotlin Multiplatform Projectとは
   1.1. Kotlin Multipratform Projectについて
   1.2. Kotlin/Native
   1.3. Kotlin/JS
   1.4. Kotlin Multiplatform Projectの仕組み
   1.5. メリット、デメリット
2. 実践Kotlin Multiplatform Project
   2.1. ライブラリ選定
   2.2. 設計を考える
   2.3. サンプルアプリを作る
3. まとめ

Kotlin Multiplatform Projectとは

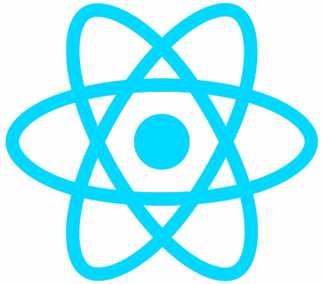
クロスプラットフォームの歴史








クロスプラットフォームの歴史



React Native

Flutter

Xamarin

Unity

Java2ObjC

Titanium Mobile

PhoneGap

Adobe Air

クロスプラットフォームツールとは

アプローチ方法が異なる

・UI含む全てのコードを共通化(独自UIを使用)

・UI含む全てのコードを共通化(ネイティブUIを使用)

・ロジックのみ共通化

UI含む全てのコードを共通化(独自UI)

- **PhoneGap(Cordova)**
  - HTML & JS based
  - Adobe, Apache

- **Flutter**
  - Dart
  - Google
  - OpenGLを使って低レイヤにUIを描画

UI含む全てのコードを共通化(ネイティブUI)

- **Xamarin Forms**
  - C# 等の .NET言語
  - Microsoft

- **React Native**
  - JSベース
  - Facebook

ロジックのみ

- **Java2ObjC**
  - Java
  - Google
  - Kotlin/Nativeに置き換えられそう
- **Xamarin(Native)**
  - C#等の.NET言語
  - Microsoft
- **Kotlin Multiplatform Project**

Kotlin Multiplatform Projectとは

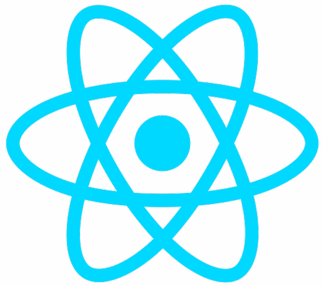
Kotlin Multiplatform Project では

UI 部分は提供せずに

ロジック部分のみ を共通化する

Kotlin Multiplatform Projectとは

全てのコードを共通で管理したい



React Native

Flutter

Xamarin Forms

Kotlin Multiplatform Projectとは

そもそも

Kotlin Multiplatform Project とは？

Kotlin Multiplatform Projectとは

Kotlin/Native ≠ Multiplatform Project

Kotlin Multiplatform Projectとは

Kotlin/Native ⊂ Multiplatform Project

Android

Native

JS

JVM

Kotlin Multiplatform Projectとは

- Kotlin/JVM (Android, Server)
- Kotlin/Native (iOS, Windows, Linux, macOS …etc)
- Kotlin/JS

の総称

Kotlin Multiplatform Projectとは

Multiplatform Projectの頭文字を取って

MPP と呼ぶ（公式）

MPPとは

既に2012年1月の時点でThe Road Aheadという構想を発表している

https://blog.jetbrains.com/kotlin/2012/01/the-road-ahead

← Kotlin Web Demo is out!

Let's Kode Together! →

The Road Ahead

Posted on January 16, 2012 by Andrey Breslav

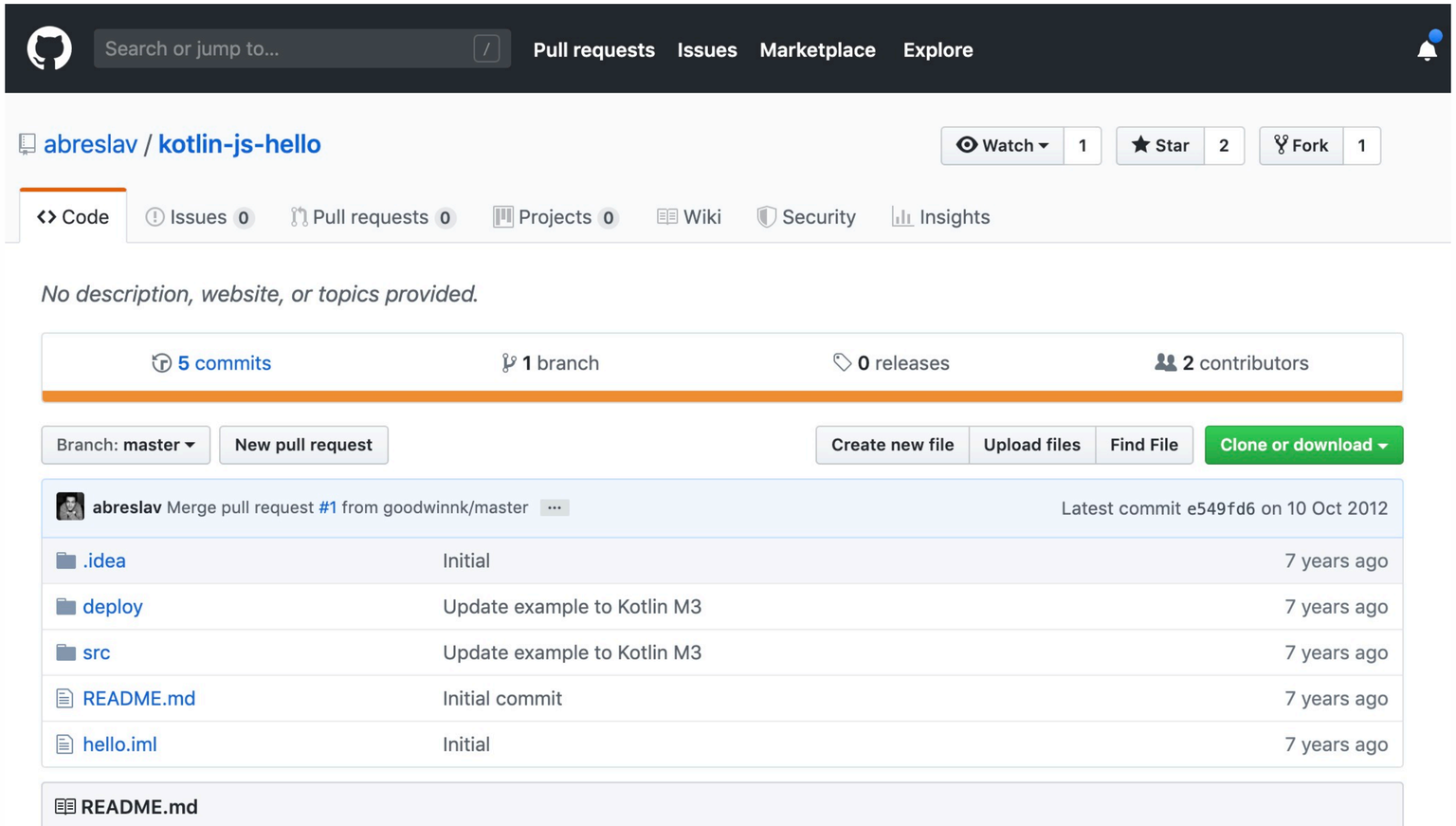
As you all know, we rolled out our first public release last week: kotlin-demo.jetbrains.com. And while you (7K+ unique visitors) are having fun with it, we keep working on the Kotlin compiler, IDE and standard library. In this post, I'll give an overview of our plans.

What you can play with today

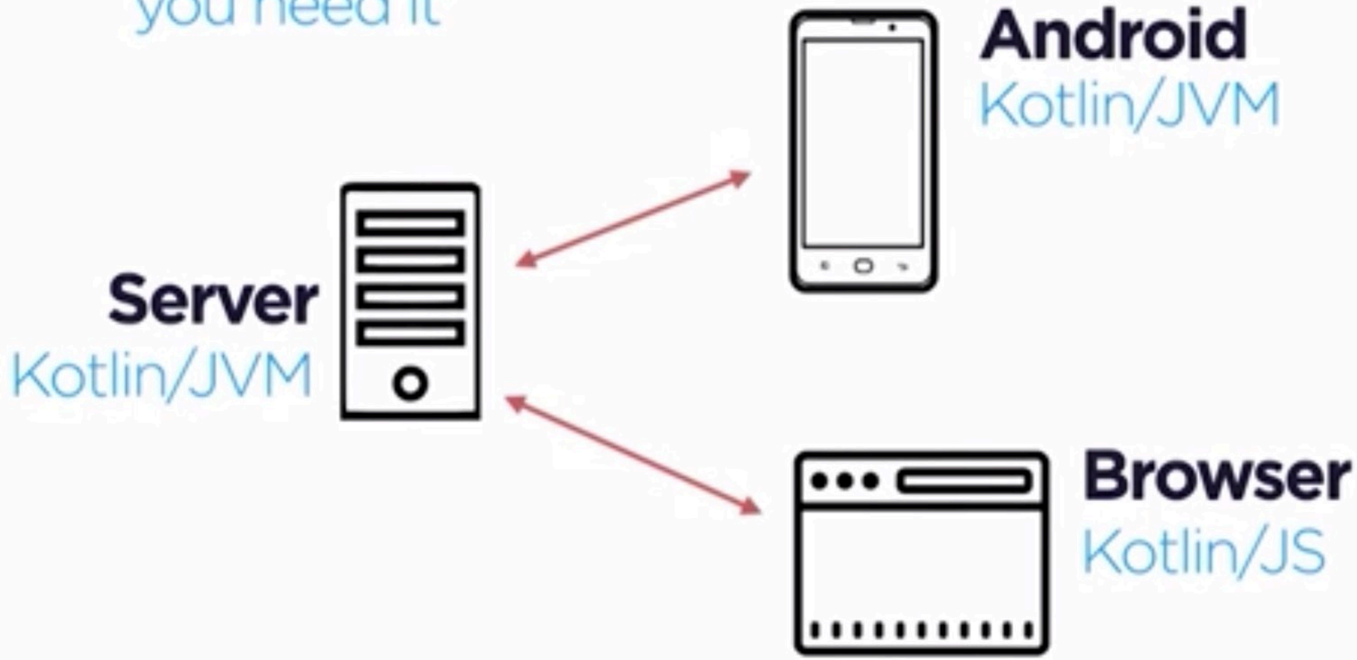
MPPとは

Kotlin/JSのサンプル

https://github.com/abreslav/kotlin-js-hello



Kotlin/Anywhere
you need it



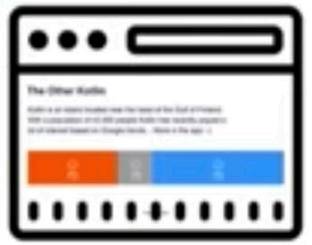
Multiplatform

Server
Kotlin/JVM

Android
Kotlin/JVM

common code

iOS
Kotlin/Native



Browser
Kotlin/JS

MPPとは

AAkira < とはいえ...

AAkira < Kotlin/Nativeもまだアレだったし
キワモノだろう...🤔


Effective multiplatform Kotlin development
by Marcin Moskała
https://www.youtube.com/watch?v=UyTBXEZ983g

Kotlin/JVM

Kotlin/JVM

- 特に説明は無い
- 普段のイメージしているKotlinのことを指す
- MPPの観点から見ると
  Server, Androidがこれにあたる

Kotlin/Native

Kotlin/Native

- LLVM Toolchainを使用して
  各プラットフォームのバイナリを作成

- VM環境が不要

Kotlin/Native

プラットフォーム	対応アーキテクチャ
iOS	arm32, arm64, simulator x86/64
MacOS	x86_64
Android NDK	arm32, arm64
Windows	mingw x86_64, x86
Linux	x86_64, arm32, arm64, MIPS, MIPS little endian, Raspberry Pi
WebAssembly	wasm32

https://kotlinlang.org/docs/reference/native-overview.html#target-platforms

Kotlin/Native

MacOS	x86_64
ndroid NDK	arm32, arm64
Windows	mingw x86_64, x86
Linux	x86_64, arm32, **arm64**, MIPS, MIPS little endian, Raspberry Pi
ebAssembly	wasm32

Kotlin/Native

Kotlin/Native ≠ iOS

Kotlin/Native - 歴史

Version	日付	主な変更	Kotlin Version
0.1	2017/04	Early Previewリリース	1.1.x
0.2	2017/05	Coroutineのサポート	1.1.x
0.3	2017/06	Android, Windowsのサポート	1.1.x
0.4	2017/10	iOS, macOSのサポート	1.2
0.5	2017/12	Objective-C, SwiftからのKotlin呼び出し	1.2
0.6	2018/02	Kotlin Multiplatform対応	1.2.20
0.7	2018/02	Objective-C, Swiftの相互運用性、multithreadまわりの強化	1.2.x
0.8	2018/07	stdlibがKotlin/JVMとKotlin/JSに, ios Arm32サポート	1.2.x
0.9	2018/09	安定版Coroutine等 主にKotlin1.3のアップデートに追従	1.3.M2
Beta	2018/09	Kotlin1.3リリースに伴いBeta版に	1.3
1.1.0	2018/12	パフォーマンス改善, Contracts対応	1.3
1.2.0(1)	2019/04	Windows32bitサポート, WindowsとMacからLinuxへのクロスコンパイル	1.3.30
1.3.0	2019/06	Linux Arm64サポート, メモリ管理の大幅改善	1.3.40

Kotlin/Native - 互換性

Kotlin	Swift	Objective-C
class	class	@interface
interface	protocol	@protocol
@Throws	throws	error:(NSError**)error
null	nil	nil
Unit return type	Void	void
String	String	NSString
String	NSMutableString	NSMutableString
List	Array	NSArray
MutableList	NSMutableArray	NSMutableArray
Set	Set	NSSet
MutableSet	NSMutableSet	NSMutableSet
Map	Dictionary	NSDictionary
MutableMap	NSMutableDictionary	NSMutableDictionary

https://kotlinlang.org/docs/reference/native/objc_interop.html

Kotlin/Native - 互換性

```kotlin
package com.github.aakira

fun hoge() {
}
```

Hoge.kt

```swift
HogeKt.hoge()
```

ViewController.swift

Kotlin/Native - 互換性

```kotlin
package com.github.aakira

fun hoge() {
}
```

Hoge.kt

```swift
HogeKt.hoge()
```

ViewController.swift

Kotlin/Native - 互換性なし

- suspend関数(Coroutine)
- inlineクラス
- Kotlinのコレクションインタフェースを実装した独自クラス
- Objective-Cのクラスを継承したKotlinのクラス

https://github.com/JetBrains/kotlin-native/blob/master/OBJC_INTEROP.md#unsupported

Kotlin/Native - Freeze

FreezeされていないObjectは
別スレッドで操作することが出来ない

Kotlin/Native - Freeze

Immutable objects

Freezeされていないは

別スレッドで操作することが出来ない

Frozen objects

Primitive型, Enum, Singleton Object...

Kotlin/Native - Freeze

```kotlin
class Hoge {}

fun foo(block: (Hoge) -> Unit) {
    val hoge = Hoge()
    block(hoge)
}
```

Actual.kt

```swift
ActualKt.foo { hoge in
    DispatchQueue.global(qos: .background).async {
        print(hoge)
    }
}
```

ViewController.swift

Kotlin/Native - Freeze

```kotlin
class Hoge {}

fun foo(block: (Hoge) -> Unit) {
    val hoge = Hoge()
    block(hoge)
}
```

Actual.kt

```swift
ActualKt.foo { hoge in
    DispatchQueue.global(qos: .background).async {
        print(hoge)
    }
}
```

ViewController.swift

Kotlin/Native - Freeze

```kotlin
class Hoge {}

fun foo(block: (Hoge) -> Unit) {
    val hoge = Hoge()
    block(hoge)
}
```

kotlin.native.IncorrectDereferenceException

```swift
ActualKt.foo { hoge in
    DispatchQueue.global(qos: .background).async {
        print(hoge)
    }
}
```

ViewController.swift

Kotlin/Native - Freeze

```kotlin
class Hoge {}

fun foo(block: (Hoge) -> Unit) {
    val hoge = Hoge()
    block(hoge)
}
```

Actual.kt

Immutableを保証

```swift
ActualKt.foo { hoge in
    DispatchQueue.global(qos: .background).async {
        print(hoge)
    }
}
```

ViewController.swift

Kotlin/Native - Freeze

```kotlin
class Hoge {}

fun foo(block: (Hoge) -> Unit) {
    val hoge = Hoge().freeze()
    block(hoge)
}
```

Actual.kt

Immutableを保証

```swift
ActualKt.foo { hoge in
    DispatchQueue.global(qos: .background).async {
        print(hoge)
    }
}
```

ViewController.swift

Kotlin/Native - Freeze

```kotlin
fun bar(block: (Int) -> Unit) {
    val hoge = 46
    block(hoge)
}
```

Actual.kt

```swift
ActualKt.bar { hoge in
    DispatchQueue.global(qos: .background).async {
        print(hoge)
    }
}
```

ViewController.swift

Kotlin/Native - Freeze

```kotlin
fun bar(block: (Int) -> Unit) {
    val hoge = 46
    block(hoge)
}
```

Primitive型は問題ない

Actual.kt

```swift
ActualKt.bar { hoge in
    DispatchQueue.global(qos: .background).async {
        print(hoge)
    }
}
```

ViewController.swift

Kotlin/Native - 成果物

成果物	詳細
EXECUTABLE	実行ファイル
KLIBRARY	Kotlin/Native library(*.klib)
FRAMEWORK	Objective-Cのフレームワーク(*.framework)
DYNAMIC	動的リンクライブラリ
STATIC	静的リンクライブラリ

https://kotlinlang.org/docs/reference/native-overview.html

Kotlin/JS

Kotlin/JS

- Kotlinで書いたコードをJavaScriptに変換する
- DOM(Document Object Model)の操作、WebGL, Node.jsを使ったサーバサイドJSも利用可能

Kotlin/JS

- JavaScriptは**動的**型付け言語
- Kotlinは**静的**型付け言語
- Kotlin => JavaScriptへの変換で型情報は失われる
- TypeScriptは将来的に対応予定 (優先度低)

Kotlin/JS - DOM

```
import kotlin.browser.document

fun main() {
    document.body?.textContent = "Hello world!"
}
```

/web/Main.kt

Kotlin/JS - DOM

```
import kotlin.browser.document

fun main() {
    document.body?.textContent = "Hello world!"
}
```

/web/Main.kt

Kotlin/JS - DOM

```
(function (root, factory) {
  if (typeof define === 'function' && define.amd)
    define(['exports', 'kotlin'], factory);
  else if (typeof exports === 'object')
    factory(module.exports, require('kotlin'));
  else {
    if (typeof kotlin === 'undefined') {
      throw new Error("Error loading module 'mpp-web'. Its dependency 'kotlin' was not found. Please,
          check whether 'kotlin' is loaded prior to 'mpp-web'.");
    }
    root['mpp-web'] = factory(typeof this['mpp-web'] === 'undefined' ? {} : this['mpp-web'], kotlin);
  }
}(this, function (_, Kotlin) {
  'use strict';
  function main() {
    var tmp$;
    (tmp$ = document.body) != null ? (tmp$.textContent = 'Hello world!') : null;
  }
  var package$com = _.com || (_.com = {});
  var package$aakira = package$com.aakira || (package$com.aakira = {});
  var package$mpp = package$aakira.mpp || (package$aakira.mpp = {});
  var package$web = package$mpp.web || (package$mpp.web = {});
  package$web.main = main;
  main();
  return _;
}));
```

/web/build/mpp-web.js

Kotlin/JS - DOM

```
(function (root, factory) {
  if (typeof define === 'function' && define.amd)
    define(['exports', 'kotlin'], factory);
  else if (typeof exports === 'object')
    factory(module.exports, require('kotlin'));
  else {
    if (typeof kotlin === 'undefined') {
      throw new Error("Error loading module 'mpp-web'. Its dependency 'kotlin' was not found. Please,
          check whether 'kotlin' is loaded prior to 'mpp-web'.");
    }
    root['mpp-web'] = factory(typeof this['mpp-web'] === 'undefined' ? {} : this['mpp-web'], kotlin);
  }
}(this, function (_, Kotlin) {
  'use strict';
  function main() {
    var tmp$;
    (tmp$ = document.body) != null ? (tmp$.textContent = 'Hello world!') : null;
  }
  var package$com = _.com || (_.com = {});
  var package$aakira = package$com.aakira || (package$com.aakira = {});
  var package$mpp = package$aakira.mpp || (package$aakira.mpp = {});
  var package$web = package$mpp.web || (package$mpp.web = {});
  package$web.main = main;
  main();
  return _;
}));
```

/web/build/mpp-web.js

Kotlin/JS - DOM

```
    root['mpp-web'] = factory(typeof this['mpp-web'] === 'undefined' ? {} :
this['mpp-web'], kotlin);
}(this, function (_, Kotlin) {
  'use strict';

function main() {
  var tmp$;
  (tmp$ = document.body) != null ?
  (tmp$.textContent = 'Hello world!') : null;
}

  var package$com = _.com || (_.com = {});
  var package$aakira = package$com.aakira || (package$com.aakira = {});
```

/web/build/mpp-web.js

Kotlin/JS - DOM

```
    root['mpp-web'] = factory(typeof this['mpp-web'] === 'undefined' ? {} :
t  'mpp-web'], kotlin);

}(this, function (_, Kotlin) {
  'use strict';

  function main() {
    var tmp$;
    (tmp$ = document.body) != null ?
    (tmp$.textContent = 'Hello world!') : null;
  }

  var package$com = _.com || (_.com = {});
  var package$aakira = package$com.aakira || (package$com.aakira = {});
```

/web/build/mpp-web.js

Kotlin/JS - DOM

```kotlin
fun main() {
    document.body?.textContent = "Hello world!"
}
```

```javascript
function main() {
    var tmp$;
    (tmp$ = document.body) != null ?
    (tmp$.textContent = 'Hello world!') : null;
}
```

Kotlin/JS - external

DOMだけではなく

JavaScriptのライブラリも読み込み可能

Kotlin/JS - external

```
external class Logger {
  companion object {
    fun log(log: String)
  }
}

fun main() {
  Logger.log("Hello logger world!")
}
```

/web/Main.kt

Kotlin/JS - external

```
external class Logger {
  companion object {
    fun log(log: String)
  }
}

fun main() {
  Logger.log("Hello logger world!")
}
```

JavaScriptのライブラリ

/web/Main.kt

Kotlin/JS - external

```
external class Logger {
  companion object {
    fun log(log: String)
  }
}

fun main() {
  Logger.log("Hello logger world!")
}
```

external修飾子

/web/Main.kt

Kotlin/JS - external

```
external class Logger {
  companion object {
    fun log(log: String)
  }
}

fun main() {
  Logger.log("Hello logger world!")
}
```

生成されるJavaScriptはどうなるか

/web/Main.kt

Kotlin/JS - external

```
(function (root, factory) {
  if (typeof define === 'function' && define.amd)
    define(['exports', 'kotlin'], factory);
  else if (typeof exports === 'object')
    factory(module.exports, require('kotlin'));
  else {
    if (typeof kotlin === 'undefined') {
      throw new Error("Error loading module 'mpp-web'. Its dependency 'kotlin' was not found. Please,
         check whether 'kotlin' is loaded prior to 'mpp-web'.");
    }
    root['mpp-web'] = factory(typeof this['mpp-web'] === 'undefined' ? {} : this['mpp-web'], kotlin);
  }
}(this, function (_, Kotlin) {
  'use strict';
  function main() {
    Logger.log('Hello logger world!');
  }
  var package$com = _.com || (_.com = {});
  var package$aakira = package$com.aakira || (package$com.aakira = {});
  var package$mpp = package$aakira.mpp || (package$aakira.mpp = {});
  var package$web = package$mpp.web || (package$mpp.web = {});
  package$web.main = main;
  main();
  return _;
}));
```

/web/build/mpp-web.js

Kotlin/JS - external

```
(function (root, factory) {
  if (typeof define === 'function' && define.amd)
    define(['exports', 'kotlin'], factory);
  else if (typeof exports === 'object')
    factory(module.exports, require('kotlin'));
  else {
    if (typeof kotlin === 'undefined') {
      throw new Error("Error loading module 'mpp-web'. Its dependency 'kotlin' was not found. Please,
          check whether 'kotlin' is loaded prior to 'mpp-web'.");
    }
    root['mpp-web'] = factory(typeof this['mpp-web'] === 'undefined' ? {} : this['mpp-web'], kotlin);
  }
}(this, function (_, Kotlin) {
  'use strict';
  function main() {
    Logger.log('Hello logger world!');
  }
  var package$com = _.com || (_.com = {});
  var package$aakira = package$com.aakira || (package$com.aakira = {});
  var package$mpp = package$aakira.mpp || (package$aakira.mpp = {});
  var package$web = package$mpp.web || (package$mpp.web = {});
  package$web.main = main;
  main();
  return _;
}));
```

/web/build/mpp-web.js

Kotlin/JS - external

```
    }
    root['mpp-web'] = factory(typeof this['mpp-web'] === 'undefined' ? {
  }
}(this, function (_, Kotlin) {
  'use strict';
  function main() {
    Logger.log('Hello logger world!');
  }
  var package$com = _.com || (_.com = {});
  var package$aakira = package$com.aakira || (package$com.aakira = {});
  var package$mpp = package$aakira.mpp || (package$aakira.mpp = {});
  var package$web = package$mpp.web || (package$mpp.web = {});
  package$web.main = main;
  main();
  return _;
```

/web/build/mpp-web.js

Kotlin/JS - external

```
    }
    root['mpp-web'] = factory(typeof this['mpp-web'] === 'undefined' ? {
  }
}(this, function (_, Kotlin) {
  'use strict';
  function main() {
    Logger.log('Hello logger world!');
  }
  var package$com = _.com || (_.com = {});
  var package$aakira = package$com.aakira || (package$com.aakira = {});
  var package$mpp = package$aakira.mpp || (package$aakira.mpp = {});
  var package$web = package$mpp.web || (package$mpp.web = {});
  package$web.main = main;
  main();
  return _;
```

ラップしたLoggerは生成されていない

/web/build/mpp-web.js

Kotlin/JS - dynamic

dynamic型

Kotlin/JS - dynamic

dynamic型

var hoge

val hoge

型がわからない

Kotlin/JS - dynamic

dynamic ≒ **Any?**

Kotlin/JS - dynamic

```
external class Logger {

    val hoge: dynamic

    fun log(log: String)
}

fun main() {
    val logger = Logger()
    val hoge = logger.hoge as String
    logger.log(hoge)
}
```

/web/Main.kt

Kotlin/JS - dynamic

```
external class Logger {

    val hoge: dynamic

    fun log(log: String)
}

fun main() {
    val logger = Logger()
    val hoge = logger.hoge as String
    logger.log(hoge)
}
```

/web/Main.kt

Kotlin/JS - dynamic

```kotlin
external class Logger {

    val hoge: dynamic

    fun log(log: String)
}

fun main() {
    val logger = Logger()
    val hoge = logger.hoge as String
    logger.log(hoge)
}
```

/web/Main.kt

Kotlin/JS - dynamic

```
external class Logger {

    val hoge: dynamic

    fun log(log: String)
}

fun main() {
    val logger = Logger()
    val hoge = logger.hoge as String
    logger.log(hoge)
}
```

/web/Main.kt

Kotlin/JS - dynamic

```kotlin
external class Logger {

    val hoge: dynamic

    fun log(log: String)
}

fun main() {
    val logger = Logger()
    val hoge = logger.hoge as String
    logger.log(hoge)
}
```

/web/Main.kt

```
var hoge = typeof (tmp$ = logger.hoge) === 'string' ? tmp$ : throwCCE();
```

/web/build/mpp-web.js

Kotlin/JS - dynamic

```kotlin
external class Logger {

    val hoge: dynamic

    fun log(log: String)
}

fun main() {
    val logger = Logger()
    val hoge = logger.hoge as String
    logger.log(hoge)
}
```

/web/Main.kt

```
var hoge = typeof (tmp$ = logger.hoge) === 'string' ? tmp$ : throwCCE();
```

/web/build/mpp-web.js

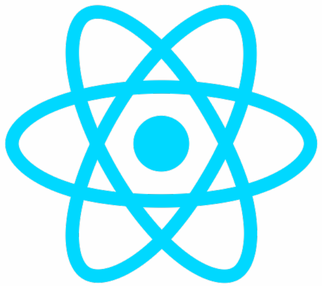
Kotlin/JS - 成果物

成果物	詳細
plain	グローバルスコープに定義される デフォルトはPlainになっている
amd	主にクライアントサイドで使われる 非同期にロードしやすい
commonjs	Node.jsなどサーバサイドで使われる事が多い
umd	AMDとCommonJSの両方をサポートしている

https://kotlinlang.org/docs/reference/js-modules.html

MPPのメリット・デメリット

MPPのメリット



React Native

Flutter

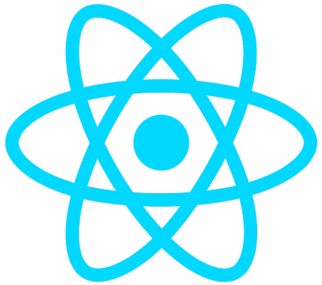
MPP

UIは共通

UIはプラットフォーム毎

MPPは敢えてUIの共通化は行っていない

MPPのメリット



JavaScript

Dart

Kotlin

ネイティブアプリと同じ言語

全てのプラットフォームを共通化出来る

Android

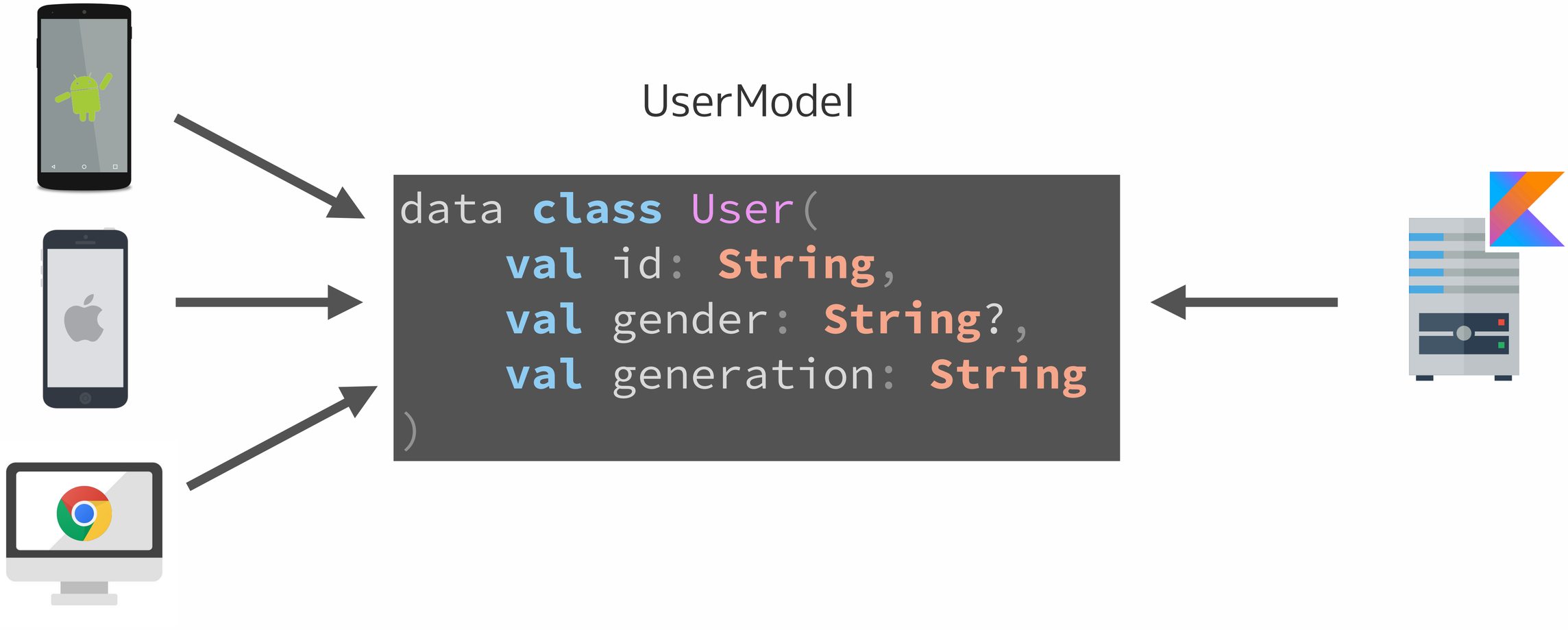
Native

JS

JVM

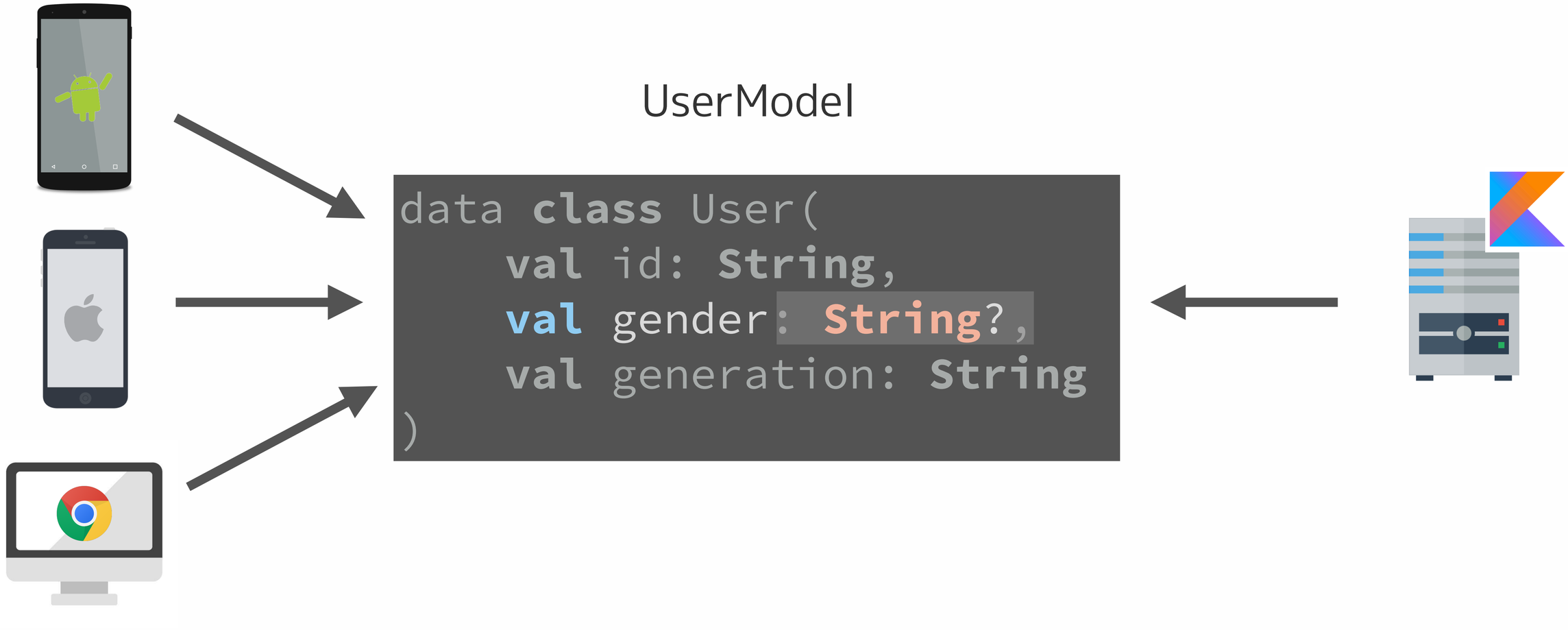
MPPのメリット

Domain Objectを共通化出来る



MPPのメリット

Domain Objectを共通化出来る



MPP(クロスプラットフォーム）のメリット

ロジックの共通化が可能

・認証系を共通化

・ログ送信基盤を共通化

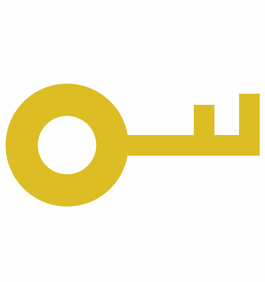
・広義の意味でのUtilityを共通化

MPP(クロスプラットフォーム）のメリット

ロジックの共通化が可能

- **認証系を共通化**
- ログ送信基盤を共通化
- 広義の意味でのUtilityを共通化

Kotlin Fest 2019



Have a nice Kotlin

69WIAVXI6buoHunGhub1J542wI3I1o3RqTqB9OgPZJM=

MPP(クロスプラットフォーム）のメリット

ロジックの共通化が可能

- 認証系を共通化
- **ログ送信基盤を共通化**
- 広義の意味でのUtilityを共通化

MPP(クロスプラットフォーム）のメリット

ロジックの共通化が可能

- 認証系を共通化
- **ログ送信基盤を共通化**
- 広義の意味でのUtilityを共通化

PM < Dimension(Key)の値が
register-userとregister**ed**-user
の２つあるのですが...?

iOSer, Webmen < register**ed**-userやろ!

Androider < あっ...ほんま...
ごめんて...

MPP(クロスプラットフォーム）のメリット

ロジックの共通化が可能

- 認証系を共通化
- ログ送信基盤を共通化
- **広義の意味でのUtilityを共通化**

MPP(クロスプラットフォーム）のメリット

ロジックの共通化が可能

- 認証系を共通化
- ログ送信基盤を共通化
- **広義の意味でのUtilityを共通化**

QA　< Androidは "残り1分"
　　iOSは "残り60秒"
　　と表示されるのですが...?

iOSer　< どっちも正しい!!!

Androider　< どっちも正しい!!!

MPPのメリット

- Android, サーバで広く使われているKotlinを使用できる
- Andorid, iOSのコードだけでなく、
  Web(JavaScript, wasm), **サーバ** のコードまでも共有することが可能
- 最初にGradle等の設定さえすれば、新しくフレームワークの記法を
  覚える必要がない
- 他のクロスプラットフォームツールではAndroidの方がバグが多いが、
  Kotlin/NativeではAndroid側が今までと変わらず開発出来る

MPPのデメリット

・Javaの資産が使えない

・iOSでCoroutineがメインスレッドしか使えない(1.3.40現在)

・.frameworkを1つしか読み込めない(1.3.40現在)

・ライブラリのKotlin versionを揃えないといけない(1.3.40現在)

・全てを共通化する場合iOS, WEBのエンジニアの理解が必要

Kotlin Multiplatform Projectの仕組み

MPPの仕組み

共通モジュールで生成された成果物を
各プラットフォームから参照する

MPPの仕組み

共通モジュールで生成された成果物を
各プラットフォームから参照する

- **Android**, **Server**

  JVM言語でGradle使っていれば、通常の外部ライブラリと同じ

- **iOS**

  .frameworkを作成してXcodeで読み込む

- **WEB**

  生成されたJavaScriptファイルを読み込む

MPPの作り方

1. 共通モジュールの作成

   ・ スライド内ではCommonモジュールと呼ぶ
   (名前は自由に変更可能)

   ・ 共通モジュール = Gradleのライブラリモジュール

2. Gradleの設定

   ・ プラグインの読み込み

   ・ アーティファクトの指定

MPPの作り方

1. 共通モジュールの作成

   ・ スライド内ではCommonモジュールと呼ぶ
   (名前は自由に変更可能)

   ・ 共通モジュール = Gradleのライブラリモジュール

2. Gradleの設定

   ・ プラグインの読み込み

   ・ アーティファクトの指定

以上！

MPPの仕組み

全ての共通化は難しい

MPPの仕組み

全ての共通化は難しい

Expect, **Actual** でプラットフォーム間の差異を解決

MPPの仕組み

Expect

抽象オブジェクト(≒Abstract)

Actual

具象オブジェクト(≒Override)

クラス、関数、プロパティ、アノテーション等全てに付けられる

ExpectとActual

```
fun hello(): String {
    return "Hello, ${platformString()}"
}

expect fun platformString(): String
```

共通モジュールに定義

/common/src/commonMain/kotlin/com/github/mpp/common/Common.kt

ExpectとActual

```
actual fun platformString() = "Hello Android!"
```

/common/src/androidMain/kotlin/com/github/mpp/common/Actual.kt

```
actual fun platformString() = "Hello iOS!"
```

/common/src/iosMain/kotlin/com/github/mpp/common/Actual.kt

```
actual fun platformString() = "Hello Web!"
```

/common/src/jsMain/kotlin/com/github/mpp/common/Actual.kt

```
actual fun platformString() = "Hello Server!"
```

/common/src/jvmMain/kotlin/com/github/mpp/common/Actual.kt

ExpectとActual

```
actual fun platformString() = "Hello Android!"
```

/common/src/androidMain/kotlin/com/github/mpp/common/Actual.kt

```
actual fun platformString() = "Hello iOS!"
```

/common/src/iosMain/kotlin/com/github/mpp/common/Actual.kt

```
actual fun platformString() = "Hello Web!"
```

/common/src/jsMain/kotlin/com/github/mpp/common/Actual.kt

```
actual fun platformString() = "Hello Server!"
```

/common/src/jvmMain/kotlin/com/github/mpp/common/Actual.kt

ExpectとActual

```
actual fun platformString() = "Hello Android!"
```

/common/src/androidMain/kotlin/com/github/mpp/common/Actual.kt

```
actual fun platformString() = "Hello iOS!"
```

/common/src/iosMain/kotlin/com/github/mpp/common/Actual.kt

```
actual fun platformString() = "Hello Web!"
```

/common/src/jsMain/kotlin/com/github/mpp/common/Actual.kt

```
actual fun platformString() = "Hello Server!"
```

/common/src/jvmMain/kotlin/com/github/mpp/common/Actual.kt

ExpectとActual

```
actual fun platformString() = "Hello Android!"
```

/common/src/androidMain/kotlin/com/github/mpp/common/Actual.kt

```
actual fun platformString() = "Hello iOS!"
```

/common/src/iosMain/kotlin/com/github/mpp/common/Actual.kt

```
actual fun platformString() = "Hello Web!"
```

/common/src/jsMain/kotlin/com/github/mpp/common/Actual.kt

```
actual fun platformString() = "Hello Server!"
```

/common/src/jvmMain/kotlin/com/github/mpp/common/Actual.kt

ExpectとActual

```
actual fun platformString() = "Hello Android!"
```

/common/src/androidMain/kotlin/com/github/mpp/common/Actual.kt

```
actual fun platformString() = "Hello iOS!"
```

/common/src/iosMain/kotlin/com/github/mpp/common/Actual.kt

```
actual fun platformString() = "Hello Web!"
```

/common/src/jsMain/kotlin/com/github/mpp/common/Actual.kt

```
actual fun platformString() = "Hello Server!"
```

/common/src/jvmMain/kotlin/com/github/mpp/common/Actual.kt

ExpectとActual

```
actual fun platformString() = "Hello Android!"
```

/common/src/androidMain/kotlin/com/github/mpp/common/Actual.kt

```
actual fun platformString() = "Hello iOS!"
```

/common/src/iosMain/kotlin/com/github/mpp/common/Actual.kt

```
actual fun platformString() = "Hello Web!"
```

/common/src/jsMain/kotlin/com/github/mpp/common/Actual.kt

```
actual fun platformString() = "Hello Server!"
```

/common/src/jvmMain/kotlin/com/github/mpp/common/Actual.kt

ExpectとActual

```
actual fun platformString() = "Hello Android!"
```

/common/src/androidMain/kotlin/com/github/mpp/common/Actual.kt

```
actual fun platformString() = "Hello iOS!"
```

/common/src/iosMain/kotlin/com/github/mpp/common/Actual.kt

```
actual fun platformString()
```

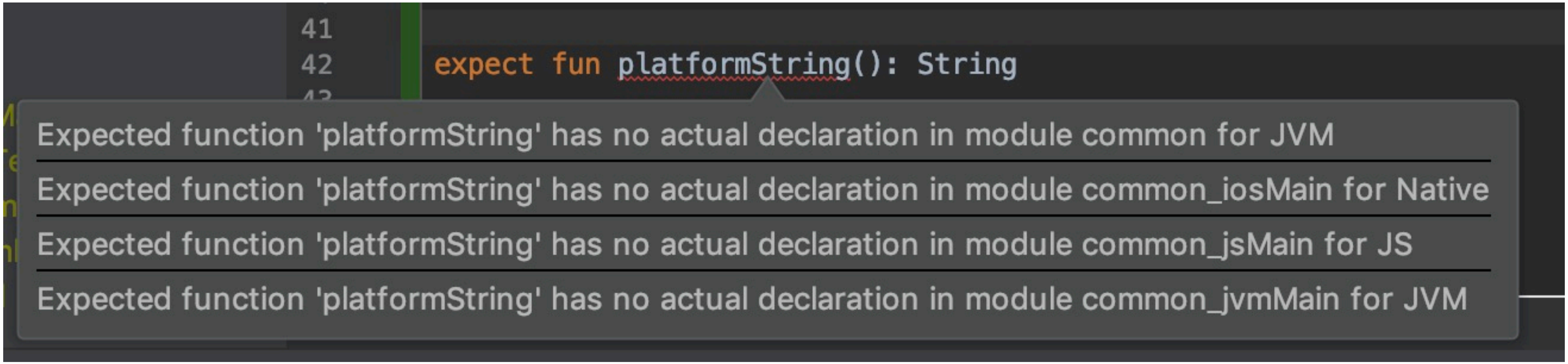
それぞれのディレクトリに定義が必要

/common/src/jsMain/kotlin/com/github/mpp/common/Actual.kt

```
actual fun platformString() = "Hello Server!"
```

/common/src/jvmMain/kotlin/com/github/mpp/common/Actual.kt

ExpectとActual



全て実装してないと、コンパイラがエラーで教えてくれる

実践Kotlin Multiplatform Project

ライブラリ選定

ジャンル	ライブラリ	URL
HTTP	ktor(client)	https://github.com/ktorio/ktor
Serializer	kotlin serialization	https://github.com/kotlin/kotlinx.serialization
RDB	SQLDelight	https://github.com/square/sqldelight
KVS	multiplatform-settings	https://github.com/russhwolf/multiplatform-settings
DI	Kodein	https://github.com/Kodein-Framework/Kodein-DI
IO	Kotlin IO	https://github.com/Kotlin/kotlinx-io
Date	Klock	https://github.com/korlibs/klock
Logger	Napier	https://github.com/AAkira/Napier

ライブラリ選定

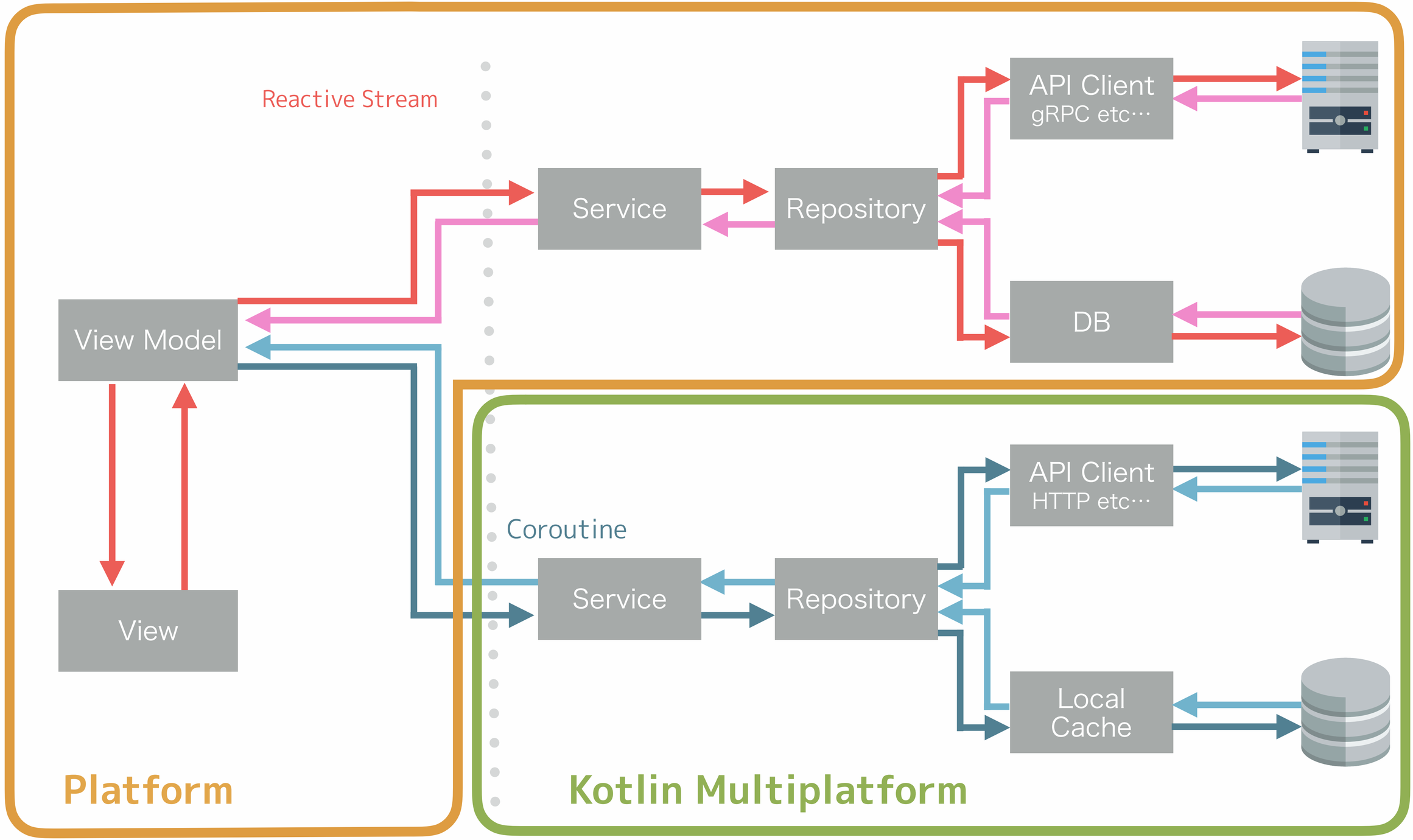
https://github.com/AAkira/Kotlin-Multiplatform-Libraries

設計を考える - Client

・公式ではMVP(Model-View-Presenter)が推奨されている
Kotlin Confのアプリもこの構成

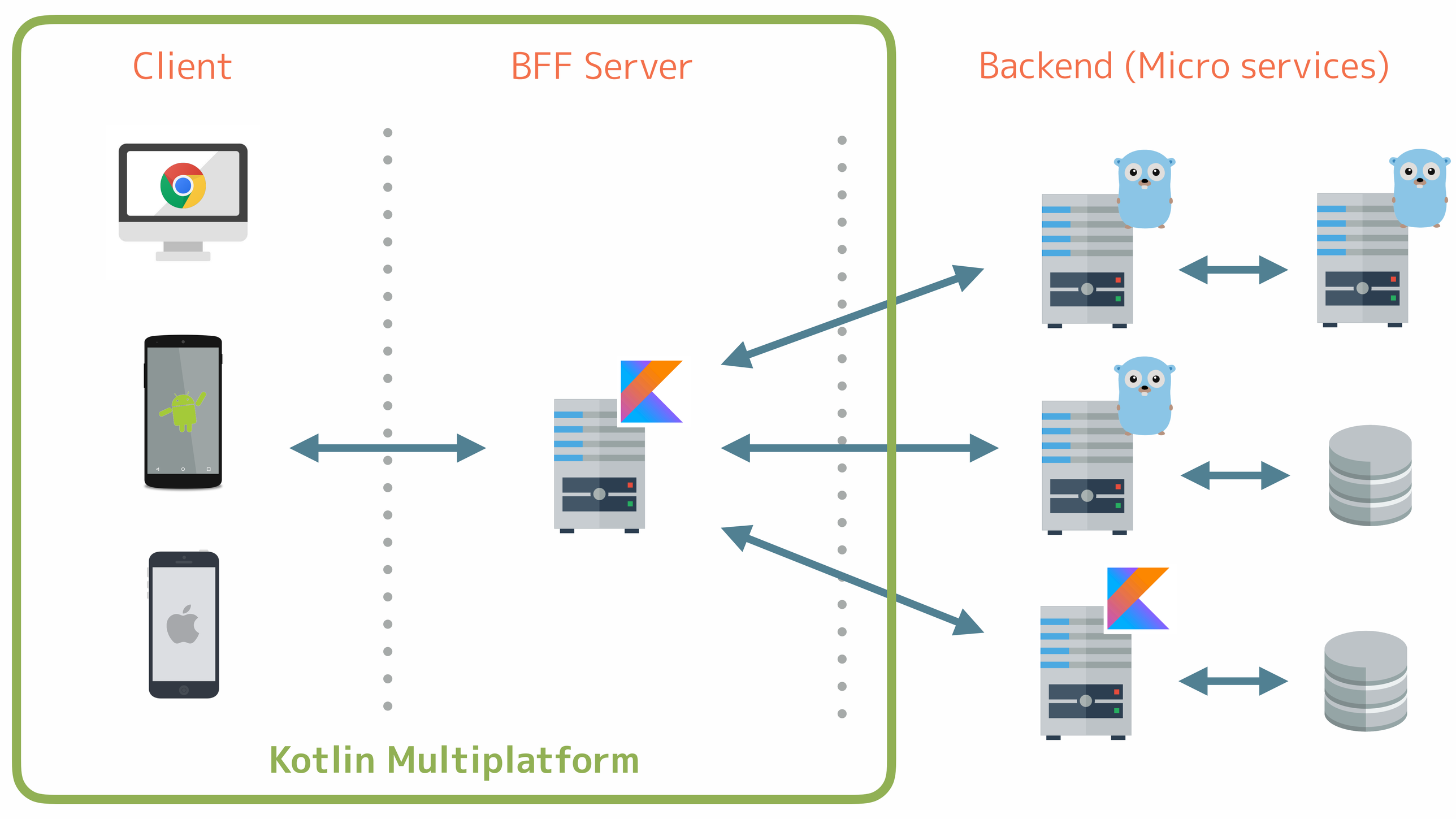
・どこまでコードを共有するかが難しい

・個人的に**現段階では**MVVM+Layered Architectureで
疎結合にしておく方が良いかもしれない

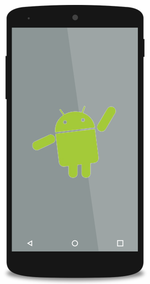

Reactive Stream
API Client
gRPC etc…
Service
Repository
View Model
DB
Coroutine
API Client
HTTP etc…
Service
Repository
View
Local
Cache
Platform
Kotlin Multiplatform

設計を考える - Server

- 現状MPPで全て作るならKtor一択
- Domain Objectだけを共有するなら
  Spring Bootとかでも良いかも
- マイクロサービス構成なら
  BFF(Backends For Frontends)サーバを作るのがオススメ


Client
BFF Server
Backend (Micro services)
Kotlin Multiplatform

MPPサンプルアプリ



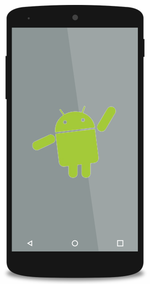
JSONを返す

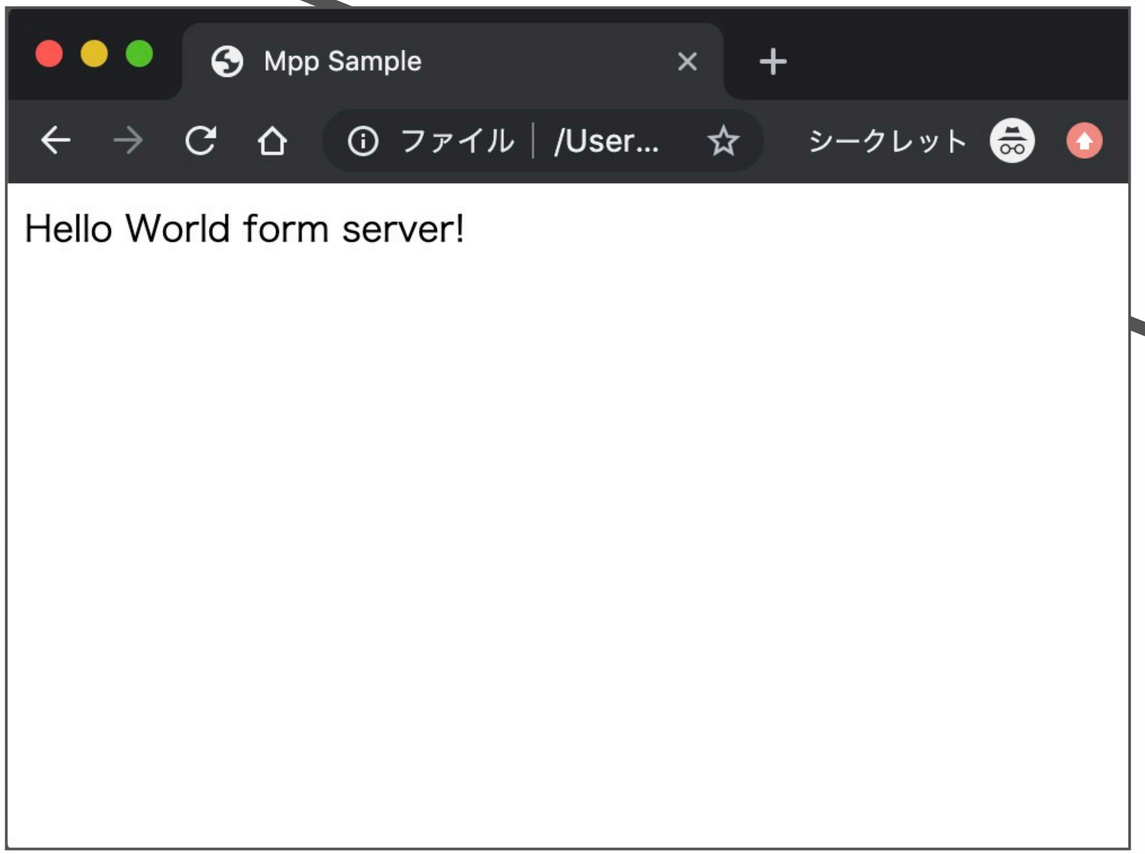
Greeting Model

```
{
    "hello":"Hello world from Server!"
}
```

./gradlew :server:run ⌥⌘1

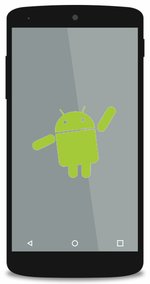
```
.Final/8462116d327bb3d1ec24258071f2e7345a73dbfc/netty-codec-4.1.36.Final.jar, /Users/aakira/.gradle/caches/modules-2/files-2.1/io.netty/netty-transport-native-unix-common/4.1.36.Final/d95d7033f400f9472db9da7834c443b96cd4bab0/netty-transport-native-unix-common-4.1.36.Final.jar, /Users/aakira/.gradle/caches/modules-2/files-2.1/io.netty/netty-transport/4.1.36.Final/8546e6be47be587acab86bbd106ca023678f07d9/netty-transport-4.1.36.Final.jar, /Users/aakira/.gradle/caches/modules-2/files-2.1/io.netty/netty-buffer/4.1.36.Final/7f2db0921dd57df4db076229830ab09bba713aeb/netty-buffer-4.1.36.Final.jar, /Users/aakira/.gradle/caches/modules-2/files-2.1/io.netty/netty-resolver/4.1.36.Final/e4d243fbf4e6837fa294f892bf97149e18129100/netty-resolver-4.1.36.Final.jar, /Users/aakira/.gradle/caches/modules-2/files-2.1/io.netty/netty-common/4.1.36.Final/f6f38fde652a70ea579897edc80e52353e487ae6/netty-common-4.1.36.Final.jar, /System/Library/Java/Extensions/MRJToolkit.jar]
15:34:48.294 [main] INFO  Application - No ktor.deployment.watch patterns match classpath entries, automatic reload is not active
15:34:48.668 [main] INFO  Application - Responding at http://0.0.0.0:8080
15:34:48.668 [main] TRACE Application - Application started: io.ktor.application.Application@61eaec38
15:35:21.122 [nioEventLoopGroup-4-1] TRACE Application - 200 OK: GET - /
<==========--> 88% EXECUTING [47s]
> :server:run
```

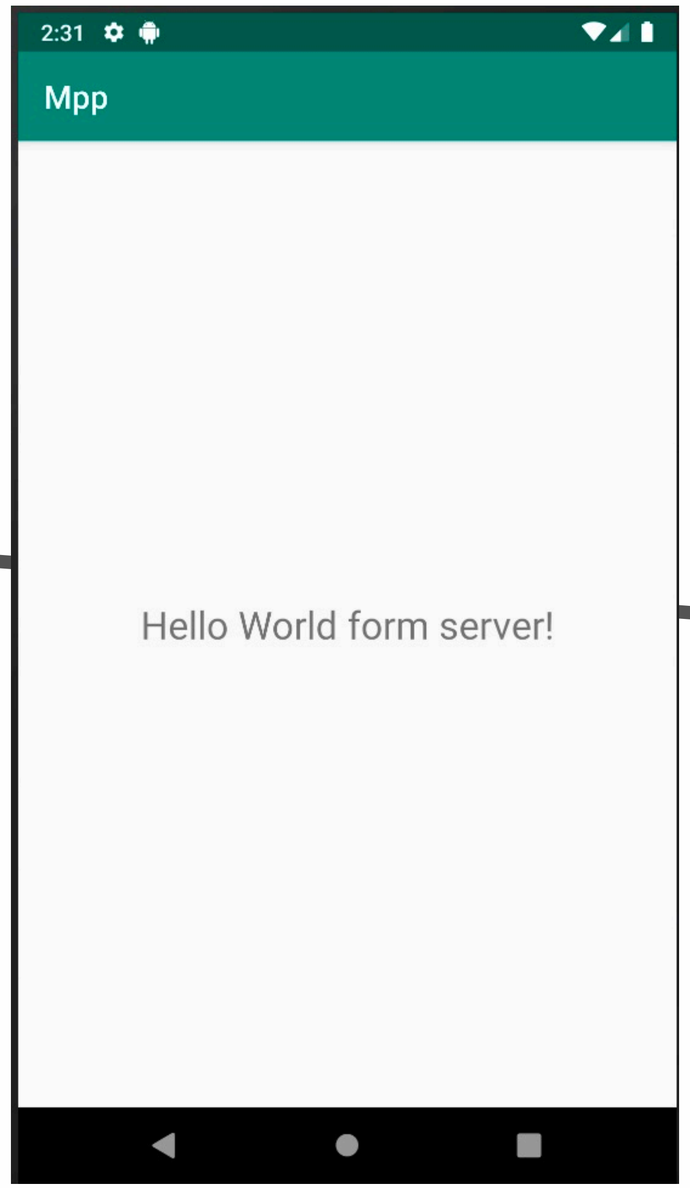




Greeting Model

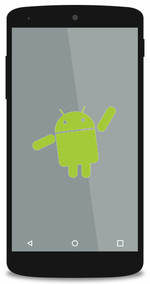
```
{
   "hello":"Hello world from Server!"
}
```





Greeting Model

```
{
    "hello":"Hello world from Server!"
}
```



Carrier 2:52 PM

Hello World form server!

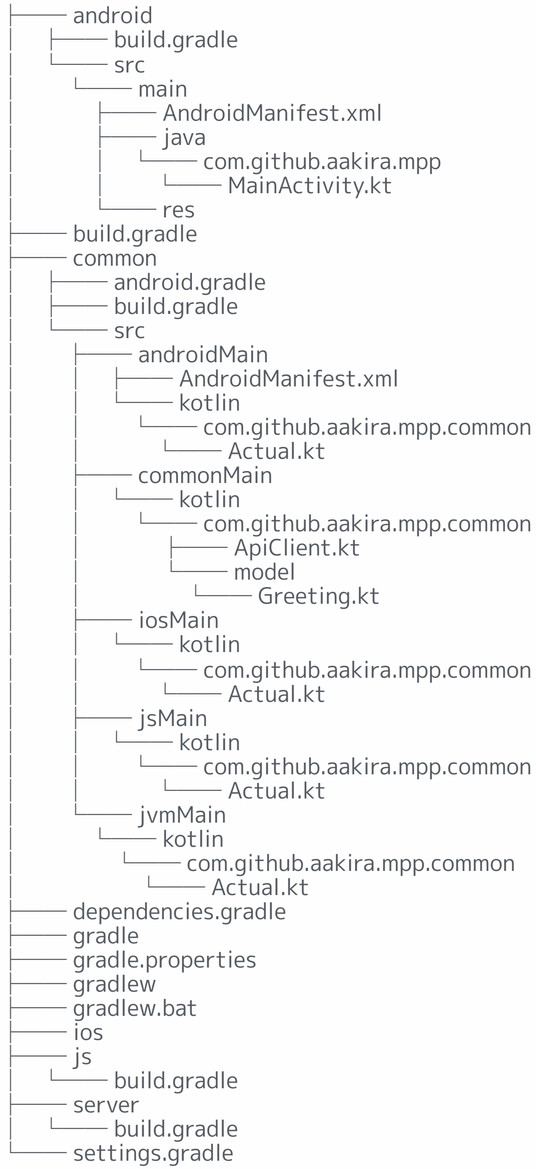
Greeting Model

```
{
   "hello":"Hello world from Server!"
}
```

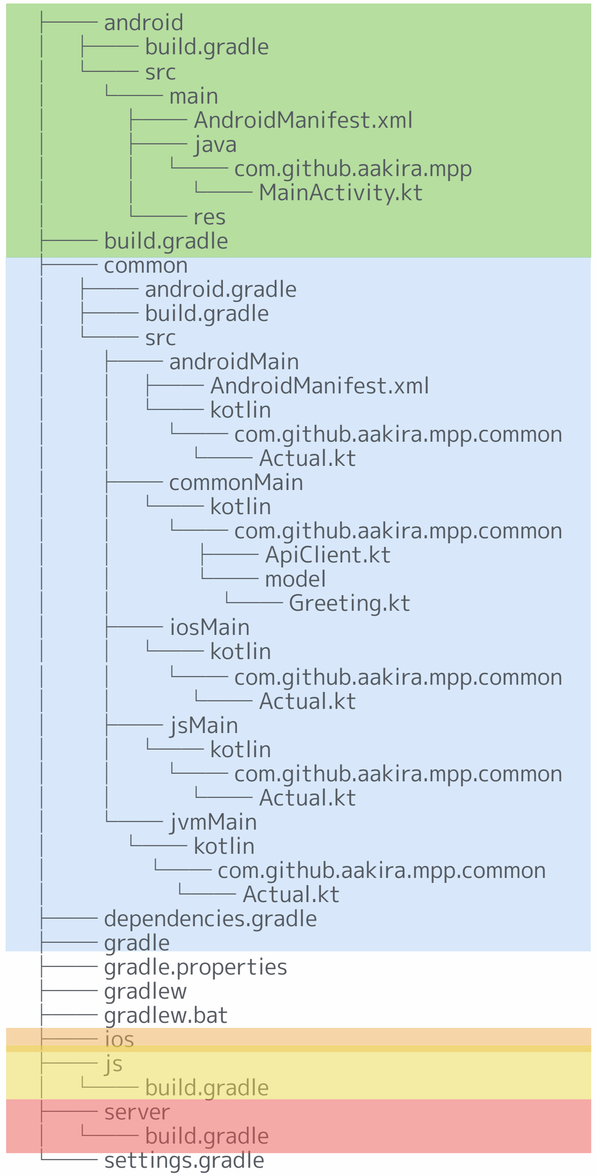
ライブラリ

ジャンル	ライブラリ	URL
Server Framework	Ktor server	https://github.com/ktorio/ktor
HTTP Client	Ktor client	https://github.com/ktorio/ktor
Serializer	kotlin serialization	https://github.com/kotlin/kotlinx.serialization
Async	Coroutine	https://github.com/Kotlin/kotlinx.coroutines

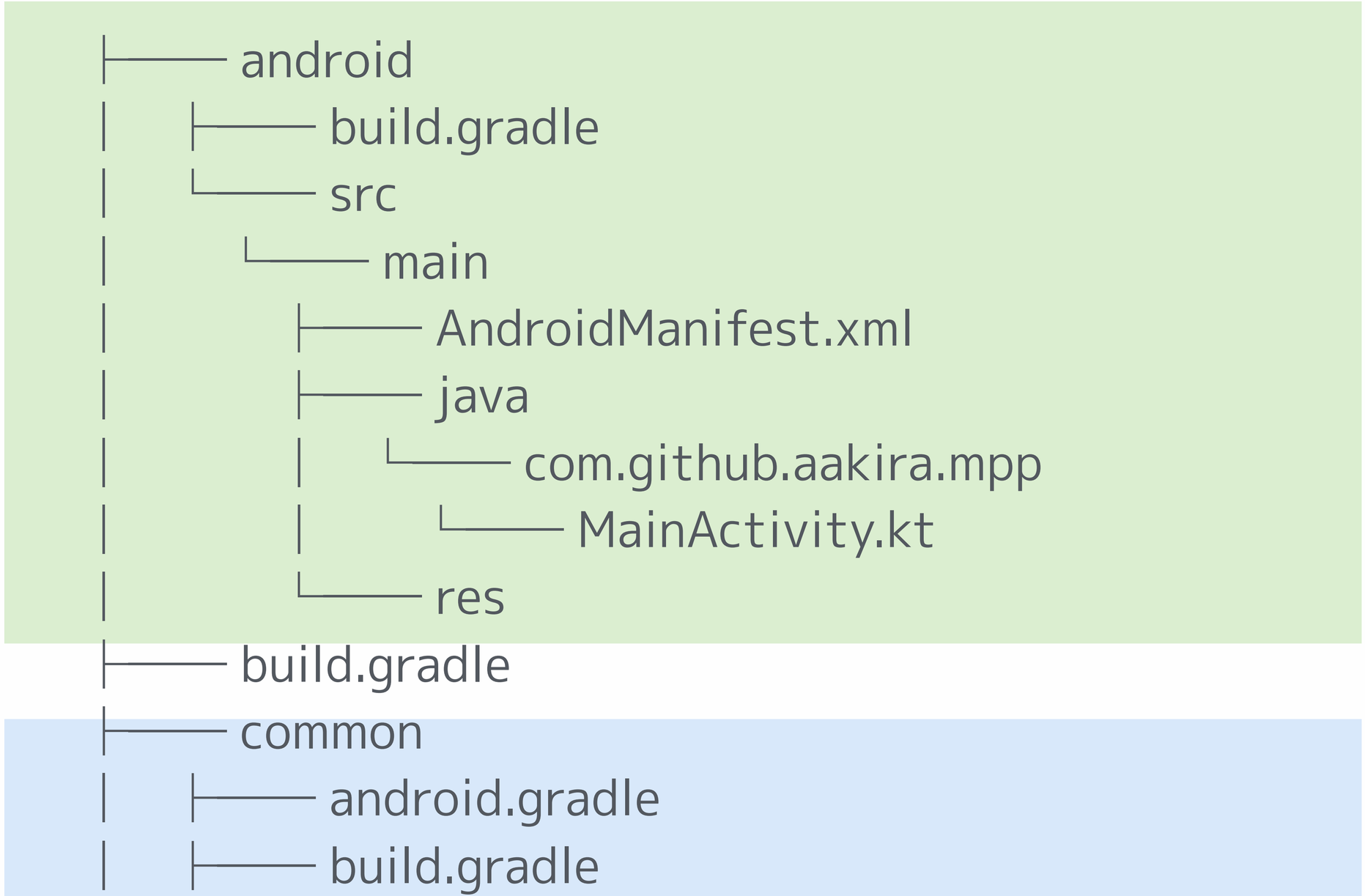
MPP - パッケージ構成


android
build.gradle
src
main
AndroidManifest.xml
java
com.github.aakira.mpp
MainActivity.kt
res
build.gradle
common
android.gradle
build.gradle
src
androidMain
AndroidManifest.xml
kotlin
com.github.aakira.mpp.common
Actual.kt
commonMain
kotlin
com.github.aakira.mpp.common
ApiClient.kt
model
Greeting.kt
iosMain
kotlin
com.github.aakira.mpp.common
Actual.kt
jsMain
kotlin
com.github.aakira.mpp.common
Actual.kt
jvmMain
kotlin
com.github.aakira.mpp.common
Actual.kt
dependencies.gradle
gradle
gradle.properties
gradlew
gradlew.bat
ios
js
build.gradle
server
build.gradle
settings.gradle

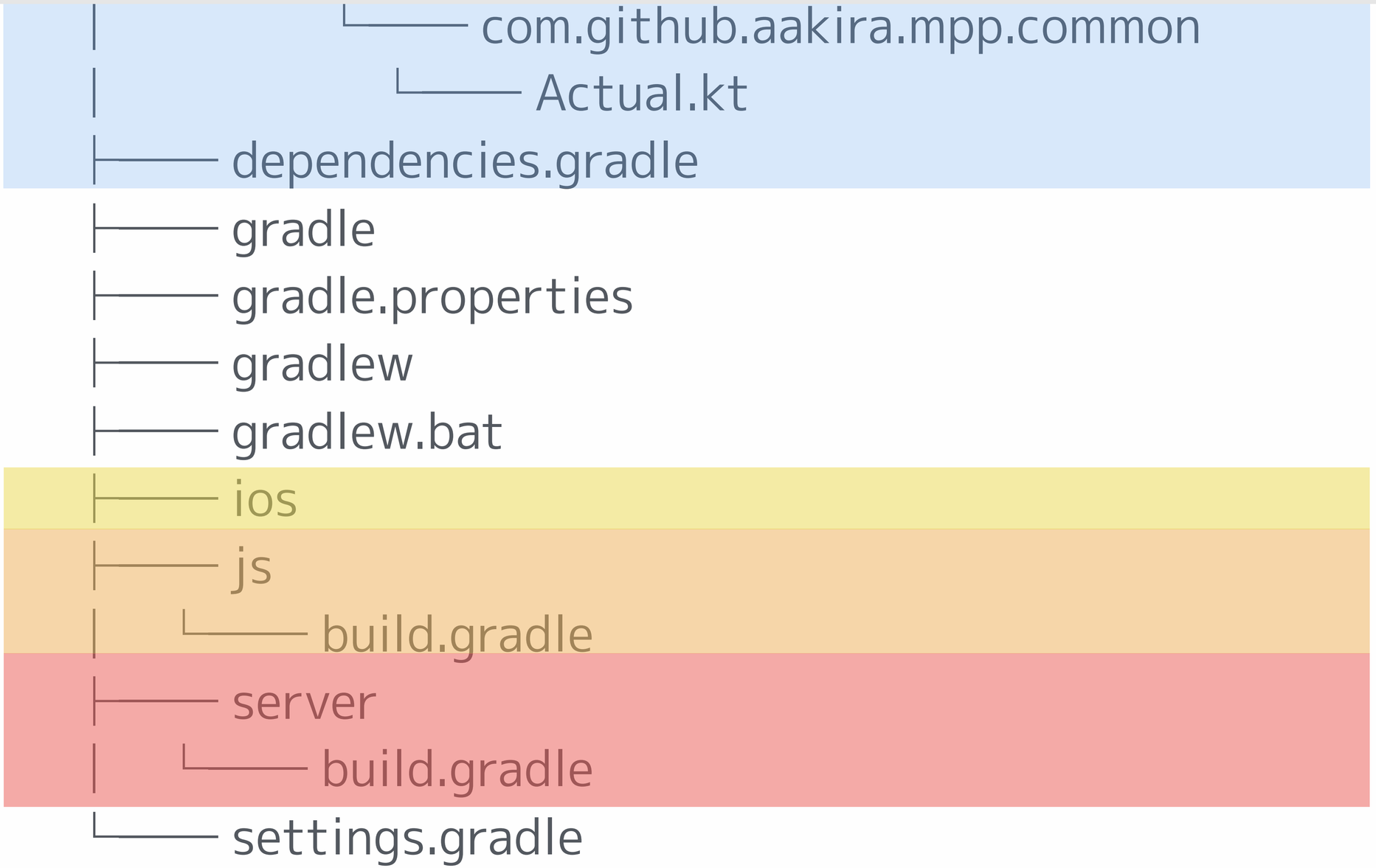
MPP - パッケージ構成


android
build.gradle
src
main
AndroidManifest.xml
java
com.github.aakira.mpp
MainActivity.kt
res
build.gradle
common
android.gradle
build.gradle
src
androidMain
AndroidManifest.xml
kotlin
com.github.aakira.mpp.common
Actual.kt
commonMain
kotlin
com.github.aakira.mpp.common
ApiClient.kt
model
Greeting.kt
iosMain
kotlin
com.github.aakira.mpp.common
Actual.kt
jsMain
kotlin
com.github.aakira.mpp.common
Actual.kt
jvmMain
kotlin
com.github.aakira.mpp.common
Actual.kt
dependencies.gradle
gradle
gradle.properties
gradlew
gradlew.bat
ios
js
build.gradle
server
build.gradle
settings.gradle

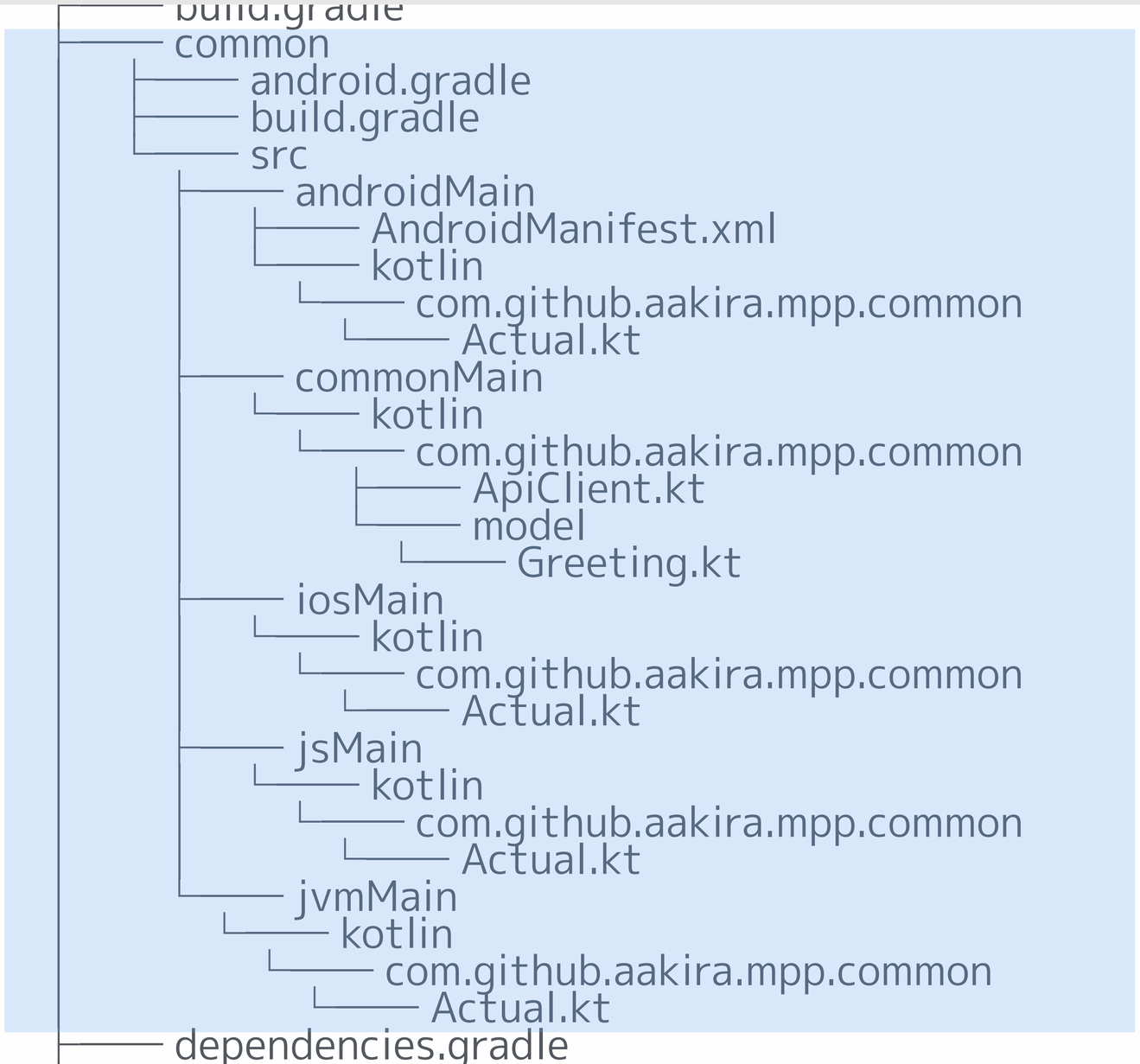
MPP - パッケージ構成


android
build.gradle
src
main
AndroidManifest.xml
java
com.github.aakira.mpp
MainActivity.kt
res
build.gradle
common
android.gradle
build.gradle

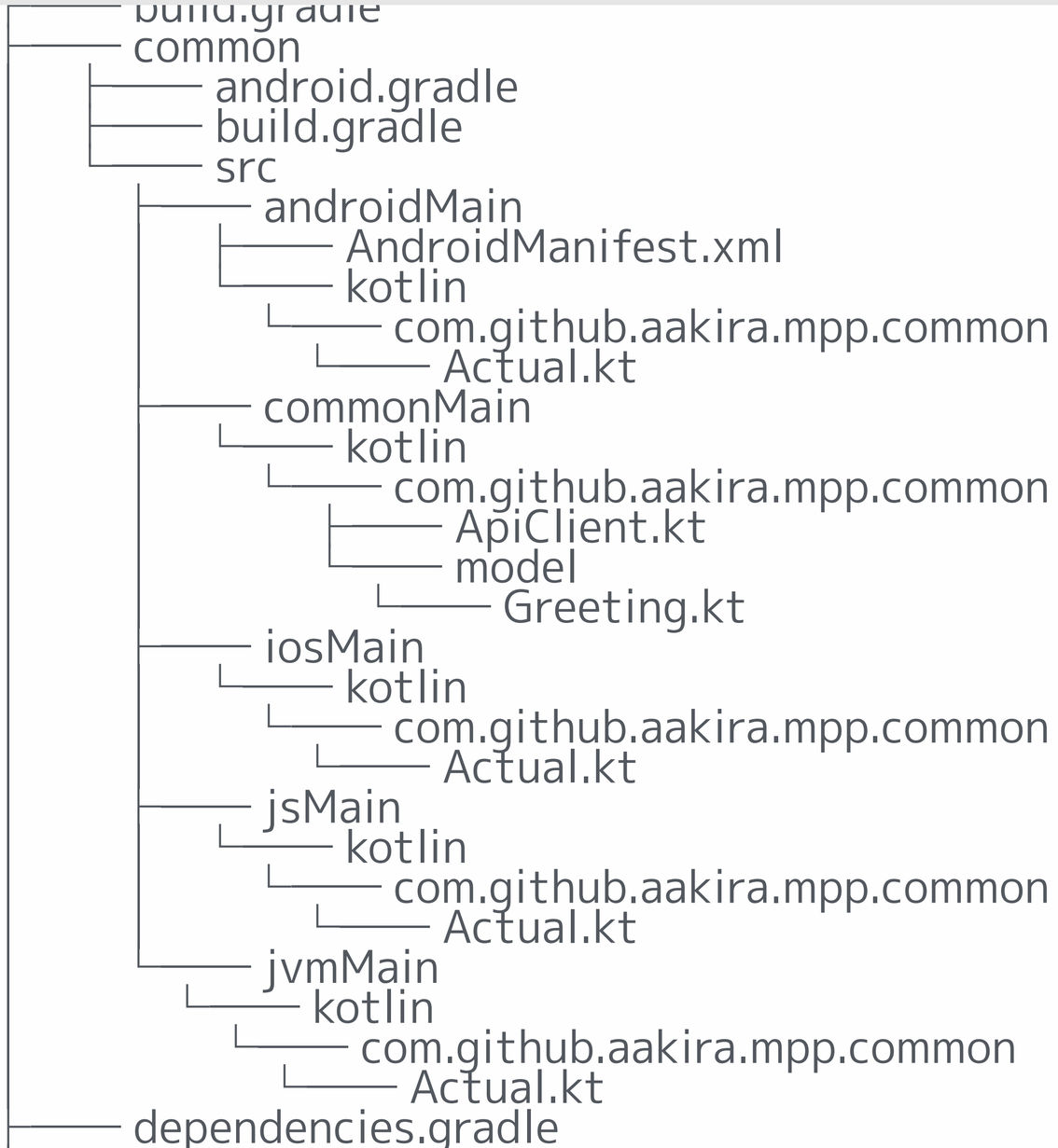
MPP - パッケージ構成


com.github.aakira.mpp.common
Actual.kt
dependencies.gradle
gradle
gradle.properties
gradlew
gradlew.bat
ios
js
build.gradle
server
build.gradle
settings.gradle

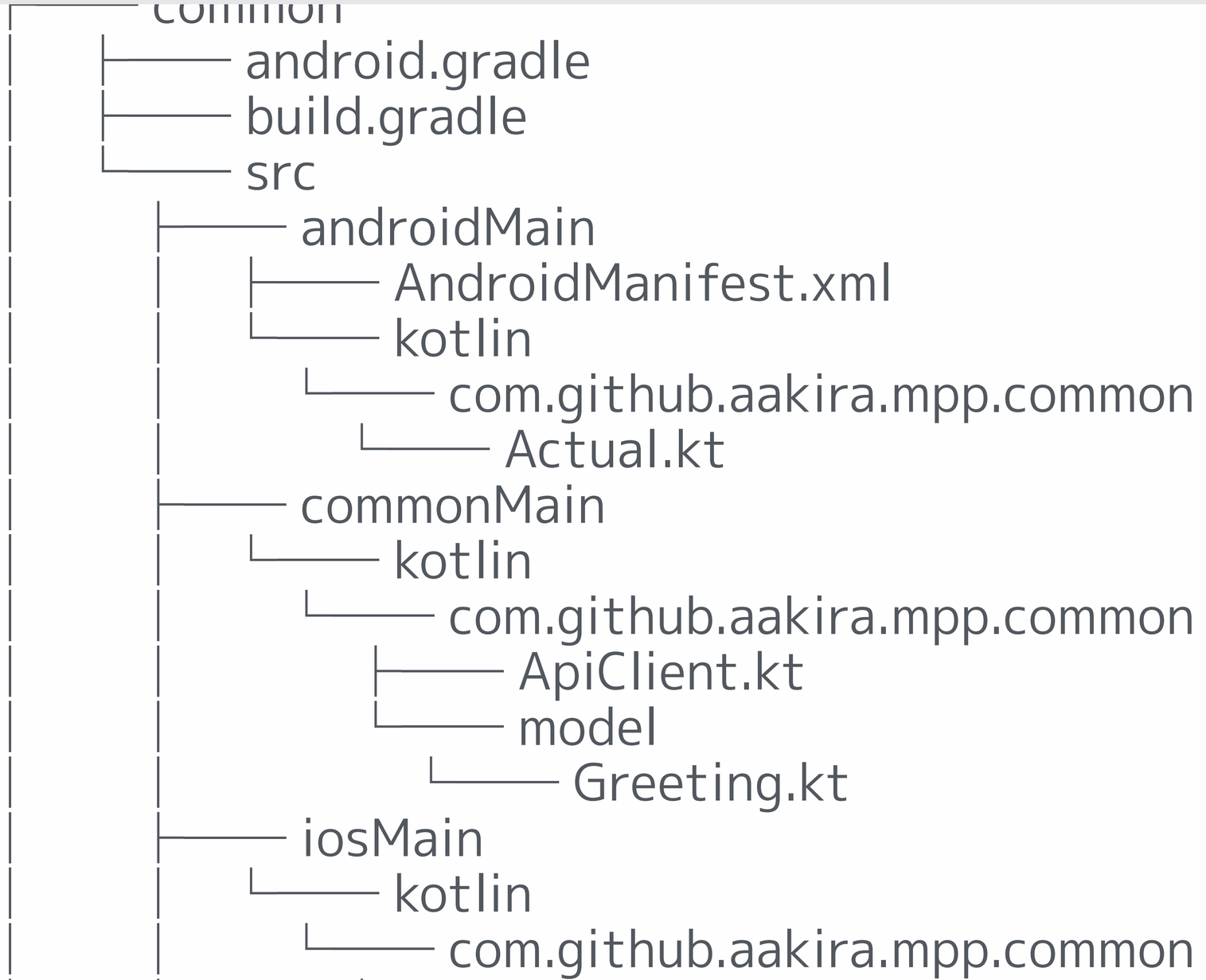
MPP - パッケージ構成


build.gradle
common
android.gradle
build.gradle
src
androidMain
AndroidManifest.xml
kotlin
com.github.aakira.mpp.common
Actual.kt
commonMain
kotlin
com.github.aakira.mpp.common
ApiClient.kt
model
Greeting.kt
iosMain
kotlin
com.github.aakira.mpp.common
Actual.kt
jsMain
kotlin
com.github.aakira.mpp.common
Actual.kt
jvmMain
kotlin
com.github.aakira.mpp.common
Actual.kt
dependencies.gradle

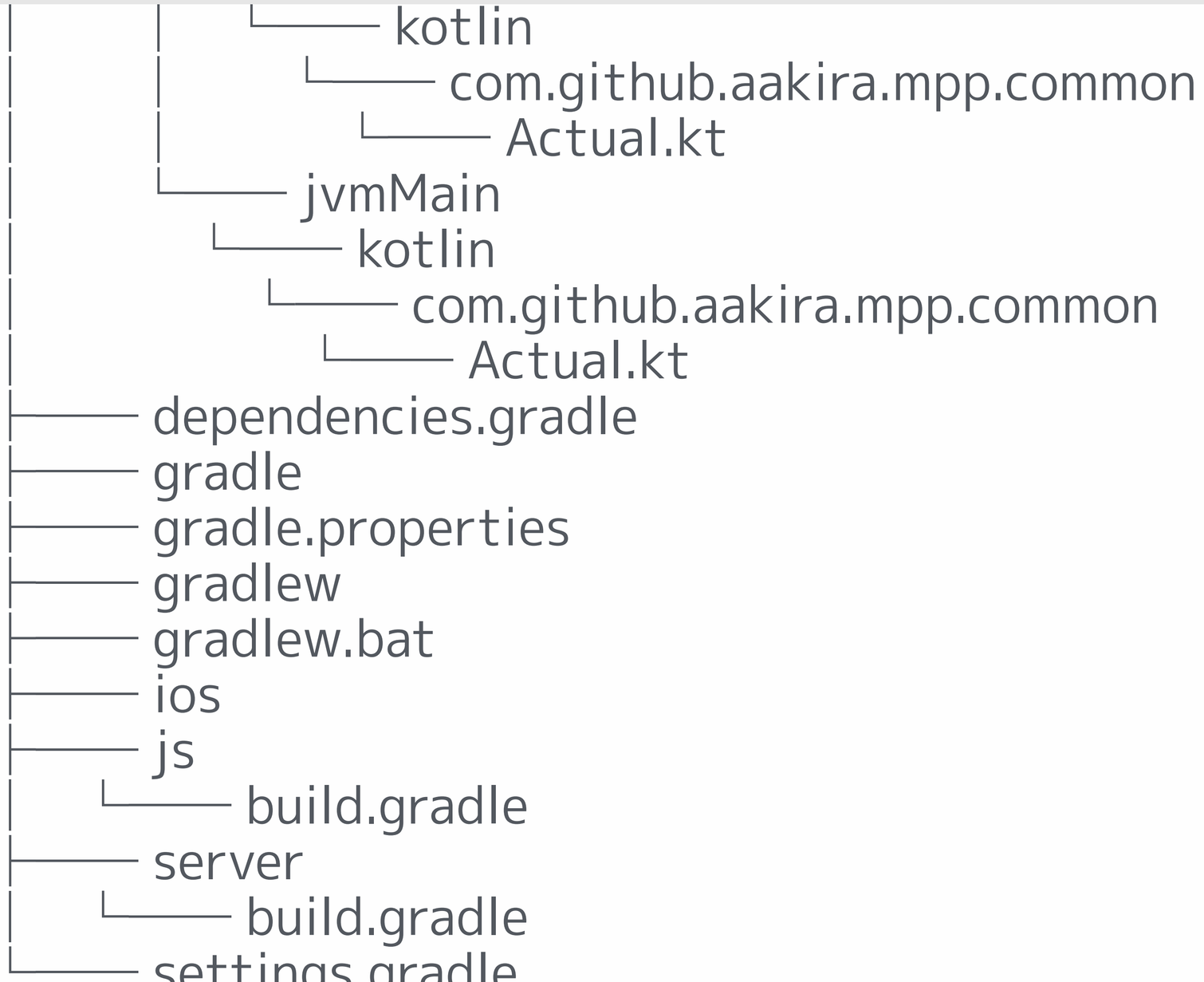
MPP - パッケージ構成


build.gradle
common
android.gradle
build.gradle
src
androidMain
AndroidManifest.xml
kotlin
com.github.aakira.mpp.common
Actual.kt
commonMain
kotlin
com.github.aakira.mpp.common
ApiClient.kt
model
Greeting.kt
iosMain
kotlin
com.github.aakira.mpp.common
Actual.kt
jsMain
kotlin
com.github.aakira.mpp.common
Actual.kt
jvmMain
kotlin
com.github.aakira.mpp.common
Actual.kt
dependencies.gradle

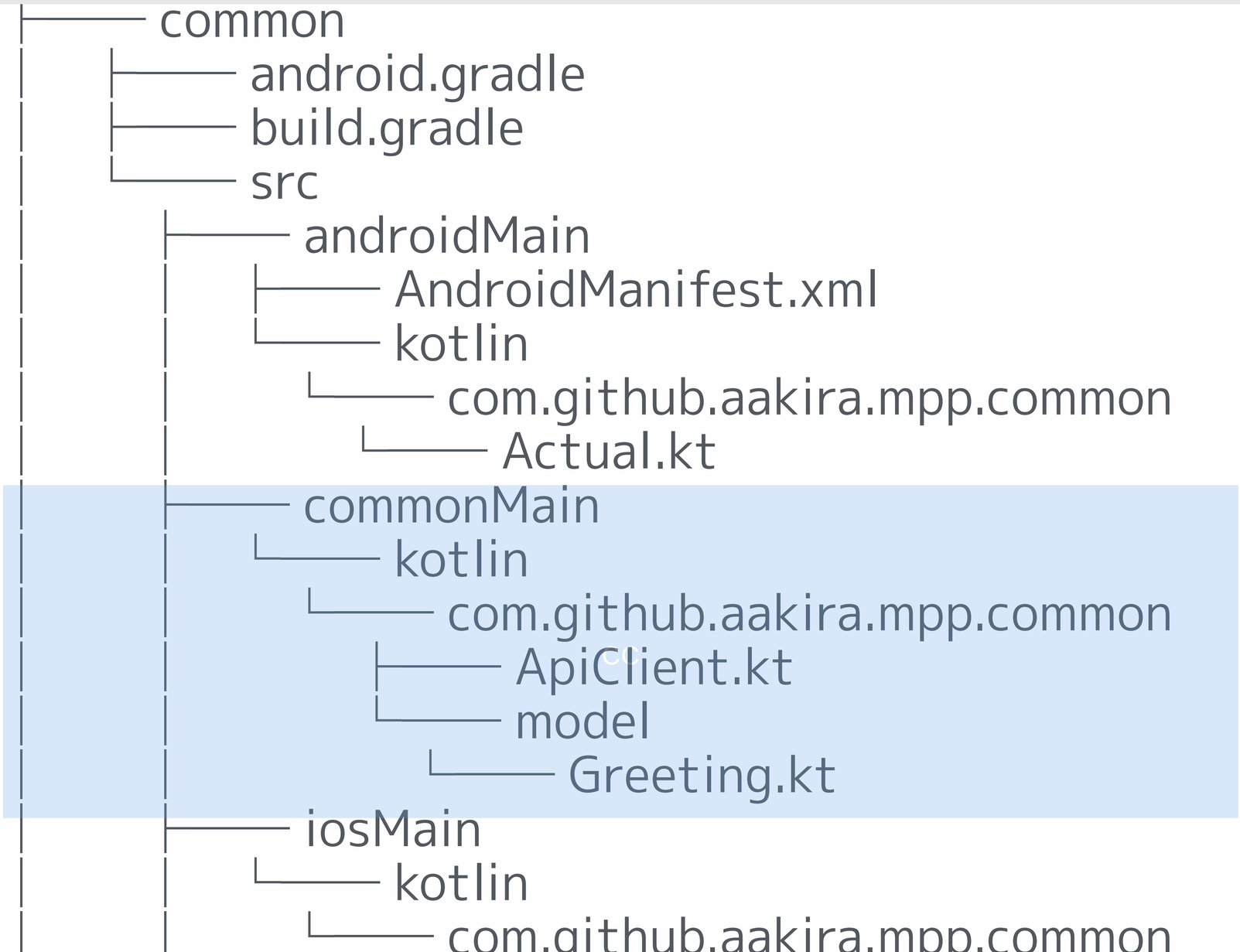
MPP - パッケージ構成


common
android.gradle
build.gradle
src
androidMain
AndroidManifest.xml
kotlin
com.github.aakira.mpp.common
Actual.kt
commonMain
kotlin
com.github.aakira.mpp.common
ApiClient.kt
model
Greeting.kt
iosMain
kotlin
com.github.aakira.mpp.common

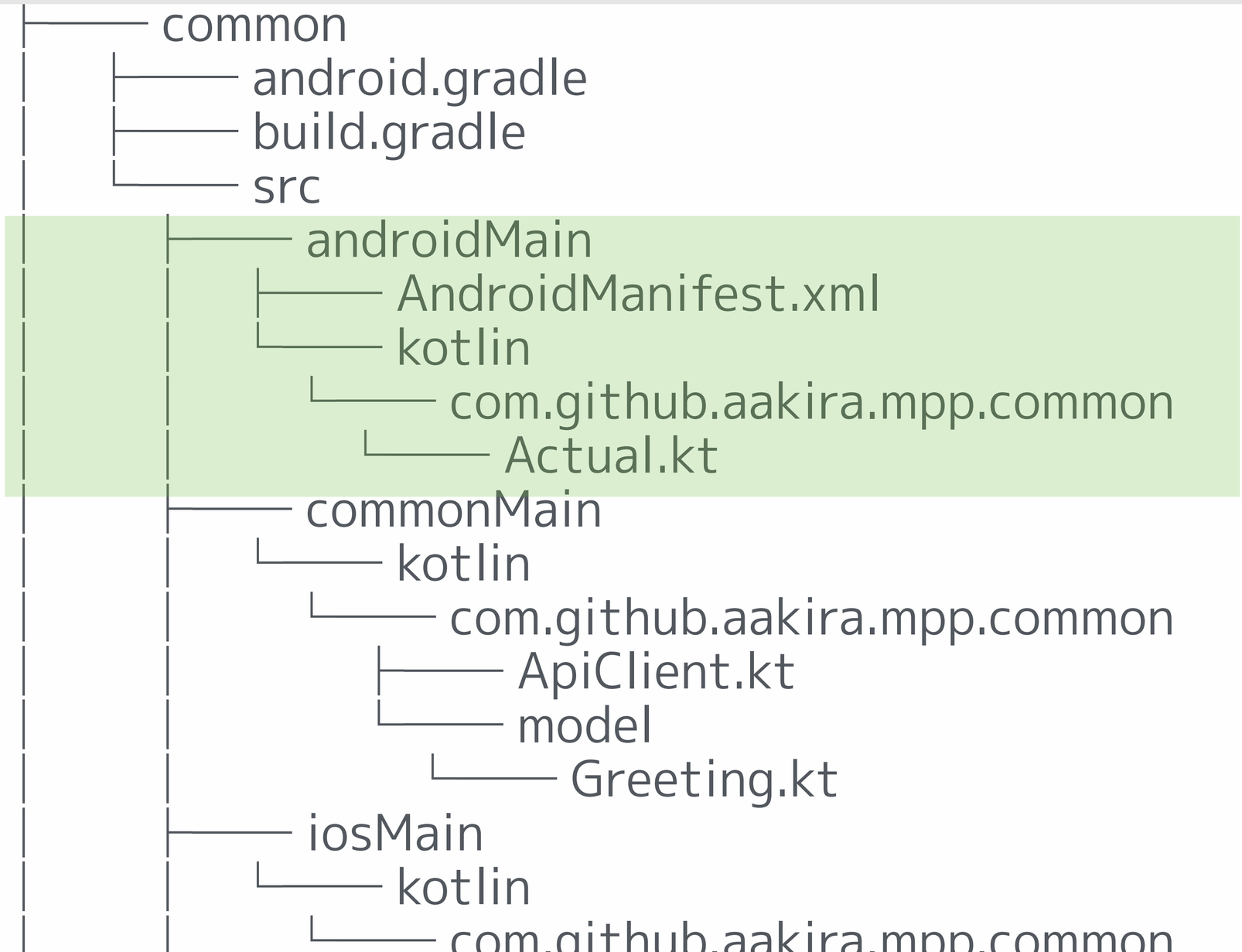
MPP - パッケージ構成


kotlin
com.github.aakira.mpp.common
Actual.kt
jvmMain
kotlin
com.github.aakira.mpp.common
Actual.kt
dependencies.gradle
gradle
gradle.properties
gradlew
gradlew.bat
ios
js
build.gradle
server
build.gradle
settings.gradle

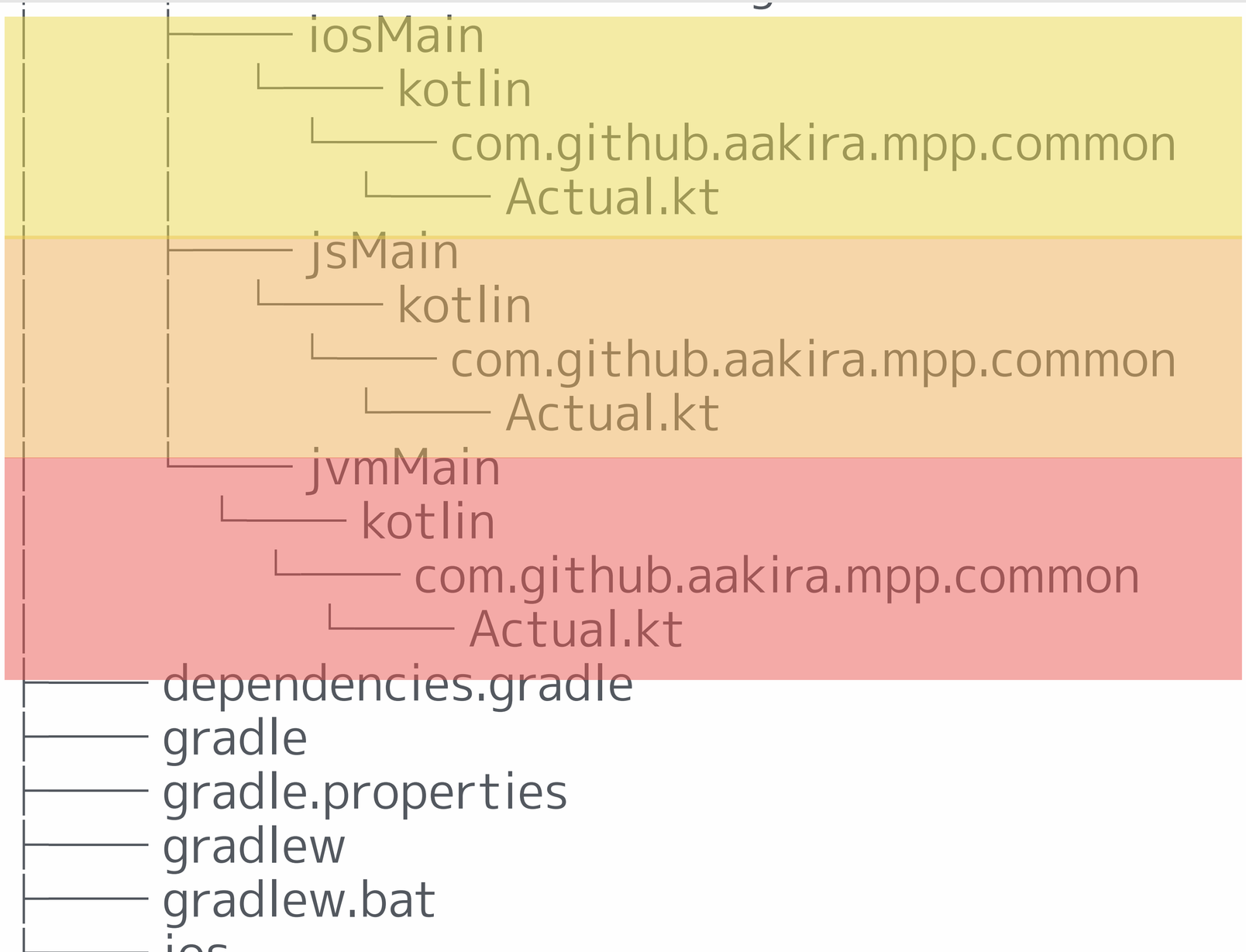
MPP - パッケージ構成


common
android.gradle
build.gradle
src
androidMain
AndroidManifest.xml
kotlin
com.github.aakira.mpp.common
Actual.kt
commonMain
kotlin
com.github.aakira.mpp.common
ApiClient.kt
model
Greeting.kt
iosMain
kotlin
com.github.aakira.mpp.common

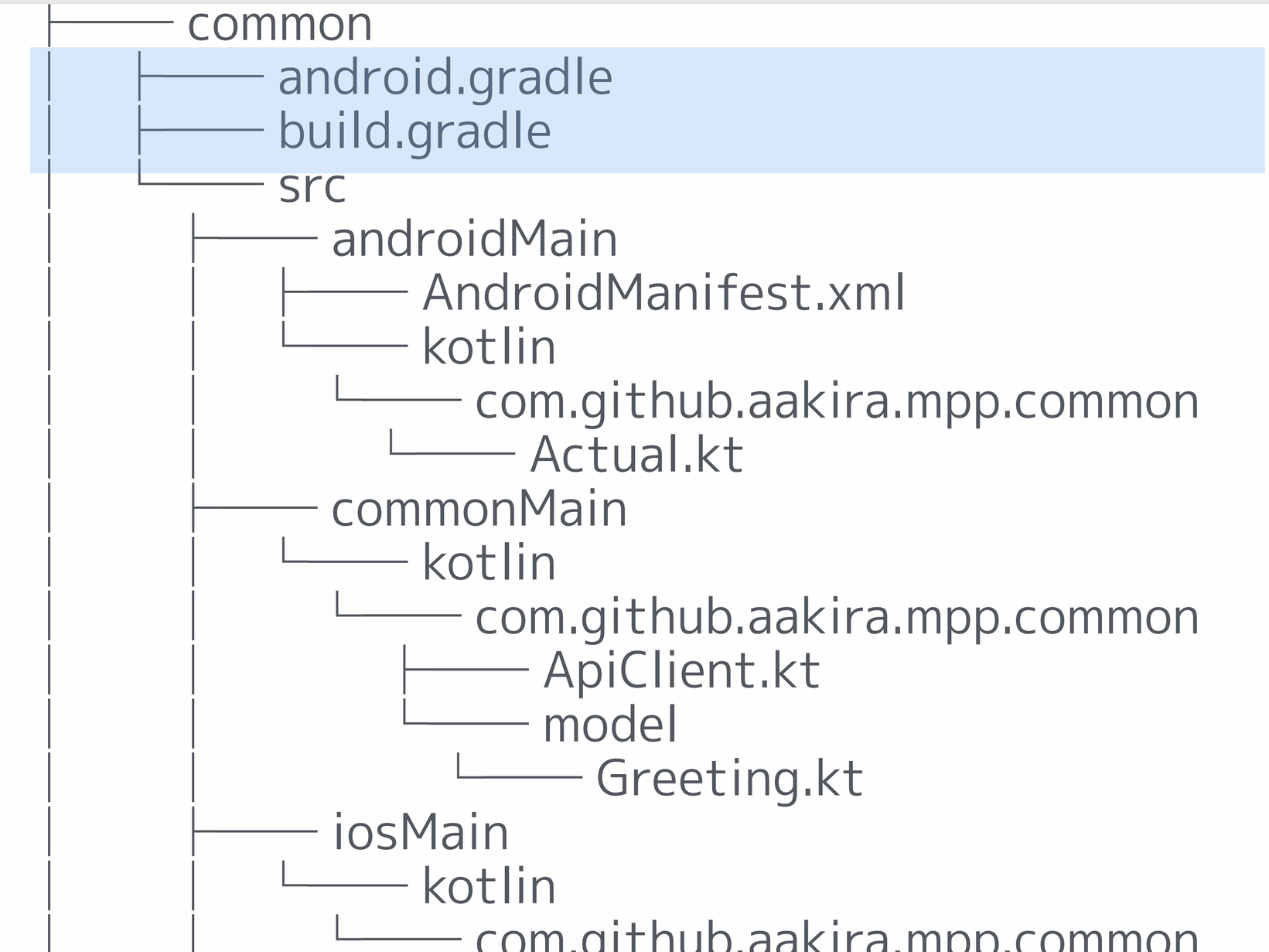
MPP - パッケージ構成


common
android.gradle
build.gradle
src
androidMain
AndroidManifest.xml
kotlin
com.github.aakira.mpp.common
Actual.kt
commonMain
kotlin
com.github.aakira.mpp.common
ApiClient.kt
model
Greeting.kt
iosMain
kotlin

MPP - パッケージ構成


iosMain
kotlin
com.github.aakira.mpp.common
Actual.kt
jsMain
kotlin
com.github.aakira.mpp.common
Actual.kt
jvmMain
kotlin
com.github.aakira.mpp.common
Actual.kt
dependencies.gradle
gradle
gradle.properties
gradlew
gradlew.bat

MPP - パッケージ構成


common
android.gradle
build.gradle
src
androidMain
AndroidManifest.xml
kotlin
com.github.aakira.mpp.common
Actual.kt
commonMain
kotlin
com.github.aakira.mpp.common
ApiClient.kt
model
Greeting.kt
iosMain
kotlin
com.github.aakira.mpp.common

MPP - Gradle設定

```
apply plugin: 'kotlin-multiplatform'
apply from: 'android.gradle'

kotlin {
  android()
  iosArm64('ios') {
    binaries {
      framework()
    }
  }
  jvm()
  js()
  sourceSets {
    commonMain {
      dependencies {
        implementation rootProject.ext.kotlinCommon
        implementation rootProject.ext.coroutineCommon
        implementation rootProject.ext.serializationCommon
        implementation rootProject.ext.ktorClient
        implementation rootProject.ext.ktorClientJson
      }
    }
    androidMain.dependencies {
    }
    iosMain {
      dependencies {
        implementation rootProject.ext.coroutineNative
        implementation rootProject.ext.serializationNative
        implementation rootProject.ext.ktorClientIos
        implementation rootProject.ext.ktorClientJsonIos
      }
    }
    jsMain {
      dependencies {
        implementation rootProject.ext.kotlinJs
        implementation rootProject.ext.coroutineJs
        implementation rootProject.ext.serializationJs
        implementation rootProject.ext.ktorClientJs
        implementation rootProject.ext.ktorClientJsonJs
      }
    }
    jvmMain {
      dependencies {
        implementation rootProject.ext.kotlinJvm
        implementation rootProject.ext.coroutine
        implementation rootProject.ext.serialization
        implementation rootProject.ext.ktorClientJvm
        implementation rootProject.ext.ktorClientJsonJvm
      }
    }
  }
}
```

/common/build.gradle

MPP - Gradle設定

```
apply plugin: 'kotlin-multiplatform'
apply from: 'android.gradle'

kotlin {
  android()
  iosArm64('ios') {
    binaries {
      framework()
    }
  }
  jvm()
  js()
  sourceSets {
    commonMain {
      dependencies {
        implementation rootProject.ext.kotlinCommon
        implementation rootProject.ext.coroutineCommon
        implementation rootProject.ext.serializationCommon
        implementation rootProject.ext.ktorClient
```

/common/build.gradle

MPP - Gradle設定

```
apply plugin: 'kotlin-multiplatform'
apply from: 'android.gradle'

kotlin {
  android()
  iosArm64('ios') {
    binaries {
      framework()
    }
  }
  jvm()
  js()
  sourceSets {
    commonMain {
      dependencies {
        implementation rootProject.ext.kotlinCommon
        implementation rootProject.ext.coroutineCommon
        implementation rootProject.ext.serializationCommon
        implementation rootProject.ext.ktorClient
```

MPP用プラグインをApply

/common/build.gradle

MPP - Gradle設定

```
apply plugin: 'kotlin-multiplatform'
apply from: 'android.gradle'

kotlin {
  android()
  iosArm64('ios') {
    binaries {
      framework()
    }
  }
  jvm()
  js()
  sourceSets {
    commonMain {
      dependencies {
        implementation rootProject.ext.kotlinCommon
        implementation rootProject.ext.coroutineCommon
        implementation rootProject.ext.serializationCommon
        implementation rootProject.ext.ktorClient
```

Kotlin1.3.0より前

- kotlin-platform-common
- kotlin-platform-android
- org.jetbrains.kotlin.platform.native
- kotlin-platform-js

/common/build.gradle

MPP - Gradle設定

```
apply plugin: 'kotlin-multiplatform'
apply from: 'android.gradle'

kotlin {
  android()
  iosArm64('ios') {
    binaries {
      framework()
    }
  }
  jvm()
  js()
  sourceSets {
    commonMain {
      dependencies {
        implementation rootProject.ext.kotlinCommon
        implementation rootProject.ext.coroutineCommon
        implementation rootProject.ext.serializationCommon
        implementation rootProject.ext.ktorClient
```

Tips: Android用のgradleは別で定義

/common/build.gradle

MPP - Gradle設定

```
apply plugin: 'kotlin-multiplatform'

apply plugin: 'com.android.library'
android {
    compileSdkVersion 28
    buildToolsVersion "28.0.3"
    defaultConfig {
        minSdkVersion 21
        targetSdkVersion 28
        versionCode 1
        versionName "1.0.0"
    }
    buildTypes {
        release {
            minifyEnabled true
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
}

kotlin {
  android()
```

Android用の定義を書かなければならない

/common/build.gradle

MPP - Gradle設定

```
apply plugin: 'kotlin-multiplatform'

apply plugin: 'com.android.library'
android {
    compileSdkVersion 28
    buildToolsVersion "28.0.3"
    defaultConfig {
        minSdkVersion 21
        targetSdkVersion 28
        versionCode 1
        versionName "1.0.0"
    }
    buildTypes {
        release {
            minifyEnabled true
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
}

kotlin {
  android()
```

Android用の定義を書かなければならない

/common/build.gradle

MPP - Gradle設定

```
apply plugin: 'kotlin-multiplatform'
apply from: 'android.gradle'

kotlin {
  android()
  iosArm64('ios') {
    binaries {
      framework()
    }
  }
  jvm()
  js()
  sourceSets {
    commonMain {
      dependencies {
        implementation rootProject.ext.kotlinCommon
        implementation rootProject.ext.coroutineCommon
        implementation rootProject.ext.serializationCommon
        implementation rootProject.ext.ktorClient
        implementation rootProject.ext.ktorClientJson
      }
    }
    androidMain.dependencies {
    }
    iosMain {
      dependencies {
        implementation rootProject.ext.coroutineNative
        implementation rootProject.ext.serializationNative
        implementation rootProject.ext.ktorClientIos
        implementation rootProject.ext.ktorClientJsonIos
      }
    }
    jsMain {
      dependencies {
        implementation rootProject.ext.kotlinJs
        implementation rootProject.ext.coroutineJs
        implementation rootProject.ext.serializationJs
        implementation rootProject.ext.ktorClientJs
        implementation rootProject.ext.ktorClientJsonJs
      }
    }
    jvmMain {
      dependencies {
        implementation rootProject.ext.kotlinJvm
        implementation rootProject.ext.coroutine
        implementation rootProject.ext.serialization
        implementation rootProject.ext.ktorClientJvm
        implementation rootProject.ext.ktorClientJsonJvm
      }
    }
  }
}
```

/common/build.gradle

```
apply plugin: 'com.android.library'
apply plugin: 'kotlin-android-extensions'

android {
    compileSdkVersion 28
    buildToolsVersion "28.0.3"
    defaultConfig {
        minSdkVersion 21
        targetSdkVersion 28
        versionCode 1
        versionName "1.0.0"
    }
    buildTypes {
        release {
            minifyEnabled true
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'),
               'proguard-rules.pro'
        }
    }
}

dependencies {
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk8:$kotlin_version"
}
```

/common/android.gradle

MPP - Gradle設定

```
apply plugin: 'kotlin-multiplatform'

apply plugin: 'com.android.library'
android {
    compileSdkVersion 28
    buildToolsVersion "28.0.3"
    defaultConfig {
        minSdkVersion 21
        targetSdkVersion 28
        versionCode 1
        versionName "1.0.0"
    }
    buildTypes {
        release {
            minifyEnabled true
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
}

kotlin {
  android()
```

/common/build.gradle

MPP - Gradle設定

```
apply plugin: 'kotlin-multiplatform'
apply from: 'android.gradle'

kotlin {
  android()
  iosArm64('ios') {
    binaries {
      framework()
    }
  }
  jvm()
  js()
  sourceSets {
    commonMain {
      dependencies {
        implementation rootProject.ext.kotlinCommon
        implementation rootProject.ext.coroutineCommon
        implementation rootProject.ext.serializationCommon
        implementation rootProject.ext.ktorClient
```

/common/build.gradle

MPP - Gradle設定

```
apply plugin: 'kotlin-multiplatform'
apply from: 'android.gradle'

kotlin {
  android()
  iosArm64('ios') {
    binaries {
      framework()
    }
  }
  jvm()
  js()
  sourceSets {
    commonMain {
      dependencies {
        implementation rootProject.ext.kotlinCommon
        implementation rootProject.ext.coroutineCommon
        implementation rootProject.ext.serializationCommon
        implementation rootProject.ext.ktorClient
```

MPP用Platform Pluginを読み込み

/common/build.gradle

MPP - Gradle設定

```
apply plugin: 'kotlin-multiplatform'
apply from: 'android.gradle'

kotlin {
  android()
  iosArm64('ios') {
    binaries {
      framework()
    }
  }
  jvm()
  js()
  sourceSets {
    commonMain {
      dependencies {
        implementation rootProject.ext.kotlinCommon
        implementation rootProject.ext.coroutineCommon
        implementation rootProject.ext.serializationCommon
        implementation rootProject.ext.ktorClient
```

/common/build.gradle

MPP - Gradle設定

```
apply plugin: 'kotlin-multiplatform'
apply from: 'android.gradle'

kotlin {
  android()
  if (project.findProperty("device")?.toBoolean() ?: false) {
    iosArm64('ios') {
      binaries {
        framework()
      }
    }
  } else {
    iosX64('ios') {
      binaries {
        framework()
      }
    }
  }
  jvm()
```

/common/build.gradle

MPP - Gradle設定

```
apply plugin: 'kotlin-multiplatform'
apply from: 'android.gradle'

kotlin {
  android()
  if (project.findProperty("device")?.toBoolean() ?: false) {
    iosArm64('ios') {
      binaries {
        framework()
      }
    }
  } else {
    iosX64('ios') {
      binaries {
        framework()
      }
    }
  }
  jvm()
```

/common/build.gradle

MPP - Gradle設定

```
apply plugin: 'kotlin-multiplatform'
apply from: 'android.gradle'

kotlin {
  android()
  if (project.findProperty("device")?.toBoolean() ?: false) {
    iosArm64('ios') {
      binaries {
        framework()
      }
    }
  } else {
    iosX64('ios') {
      binaries {
        framework()
      }
    }
  }
  jvm()
```

/common/build.gradle

MPP - Gradle設定

```
apply plugin: 'kotlin-multiplatform'
apply from: 'android.gradle'

kotlin {
  android()
  iosArm64('ios') {
    binaries {
      framework()
    }
  }
  jvm()
  js()
  sourceSets {
    commonMain {
      dependencies {
        implementation rootProject.ext.kotlinCommon
        implementation rootProject.ext.coroutineCommon
        implementation rootProject.ext.serializationCommon
        implementation rootProject.ext.ktorClient
```

/common/build.gradle

MPP - Gradle設定

```
apply plugin: 'kotlin-multiplatform'
apply from: 'android.gradle'

kotlin {
  android()
  iosArm64('ios') {
    binaries {
      framework()
    }
  }
  jvm()
  js()
  sourceSets {
    commonMain {
      dependencies {
        implementation rootProject.ext.kotlinCommon
        implementation rootProject.ext.coroutineCommon
        implementation rootProject.ext.serializationCommon
        implementation rootProject.ext.ktorClient
```

/common/build.gradle

MPP - Gradle設定

```
apply plugin: 'kotlin-multiplatform'
apply from: 'android.gradle'

kotlin {
  android()
  iosArm64('ios') {
    binaries {
      framework()
    }
  }
  jvm()
  js() {
    browser()
  }
  sourceSets {
    commonMain {
      dependencies {
        implementation rootProject.ext.kotlinCommon
        implementation rootProject.ext.coroutineCommon
```

/common/build.gradle

MPP - Gradle設定

```
apply plugin: 'kotlin-multiplatform'
apply from: 'android.gradle'

kotlin {
  android()
  iosArm64('ios') {
    binaries {
      framework()
    }
  }
  jvm()
  js()
  sourceSets {
    commonMain {
      dependencies {
        implementation rootProject.ext.kotlinCommon
        implementation rootProject.ext.coroutineCommon
        implementation rootProject.ext.serializationCommon
        implementation rootProject.ext.ktorClient
```

/common/build.gradle

MPP - Gradle設定

```
sourceSets {
  commonMain {
    dependencies {
      implementation rootProject.ext.kotlinCommon
      implementation rootProject.ext.coroutineCommon
      implementation rootProject.ext.serializationCommon
      implementation rootProject.ext.ktorClient
      implementation rootProject.ext.ktorClientJson
    }
  }
  androidMain.dependencies {
  }
  iosMain {
    dependencies {
      implementation rootProject.ext.coroutineNative
      implementation rootProject.ext.serializationNative
      implementation rootProject.ext.ktorClientIos
      implementation rootProject.ext.ktorClientJsonIos
    }
```

依存関係の定義

/common/build.gradle

MPP - Gradle設定

```
sourceSets {
  commonMain {
    dependencies {
      implementation "org.jetbrains.kotlin:kotlin-stdlib-common:$kotlin_version"
      implementation "org.jetbrains.kotlinx:kotlinx-coroutines-core-common:1.2.2"
      implementation "org.jetbrains.kotlinx:kotlinx-serialization-runtime:0.11.1"
      implementation "io.ktor:ktor-client-core:1.2.2"
      implementation "io.ktor:ktor-client-gson:1.2.2"
    }
  }
  androidMain.dependencies {
  }
  iosMain {
    dependencies {
      implementation "org.jetbrains.kotlinx:kotlinx-coroutines-core-native:1.2.2"
      implementation "org.jetbrains.kotlinx:kotlinx-serialization-runtime-native:0.11.1"
      implementation "io.ktor:ktor-client-ios:1.2.2"
      implementation "io.ktor:ktor-client-json-native:1.2.2"
    }
```

ルートにまとめて定義

/common/build.gradle

MPP - Gradle設定

```
sourceSets {
  commonMain {
    dependencies {
      implementation rootProject.ext.kotlinCommon
      implementation rootProject.ext.coroutineCommon
      implementation rootProject.ext.serializationCommon
      implementation rootProject.ext.ktorClient
      implementation rootProject.ext.ktorClientJson
    }
  }
  androidMain.dependencies {
  }
  iosMain {
    dependencies {
      implementation rootProject.ext.coroutineNative
      implementation rootProject.ext.serializationNative
      implementation rootProject.ext.ktorClientIos
      implementation rootProject.ext.ktorClientJsonIos
    }
  }
```

共通モジュールの依存定義

/common/build.gradle

MPP - Gradle設定

```
        implementation rootProject.ext.serializationCommon
        implementation rootProject.ext.ktorClient
        implementation rootProject.ext.ktorClientJson
    }
}
androidMain.dependencies {
}
iosMain {
    dependencies {
        implementation rootProject.ext.coroutineNative
        implementation rootProject.ext.serializationNative
        implementation rootProject.ext.ktorClientIos
        implementation rootProject.ext.ktorClientJsonIos
    }
}
jsMain {
    dependencies {
        implementation rootProject.ext.kotlinJs
        implementation rootProject.ext.coroutineJs
```

/common/build.gradle

MPP - Gradle設定

```
        implementation rootProject.ext.serializationCommon
        implementation rootProject.ext.ktorClient
        implementation rootProject.ext.ktorClientJson
    }
}
androidMain.dependencies {
}
iosMain {
    dependencies {
        implementation rootProject.ext.coroutineNative
        implementation rootProject.ext.serializationNative
        implementation rootProject.ext.ktorClientIos
        implementation rootProject.ext.ktorClientJsonIos
    }
}
jsMain {
    dependencies {
        implementation rootProject.ext.kotlinJs
        implementation rootProject.ext.coroutineJs
```

/common/build.gradle

MPP - Gradle設定

```
apply plugin: 'kotlin-multiplatform'
apply from: 'android.gradle'

kotlin {
  android()
  iosArm64('ios') {
    binaries {
      framework()
    }
  }
  jvm()
  js()
  sourceSets {
    commonMain {
      dependencies {
        implementation rootProject.ext.kotlinCommon
        implementation rootProject.ext.coroutineCommon
        implementation rootProject.ext.serializationCommon
        implementation rootProject.ext.ktorClient
```

名前を指定している

/common/build.gradle

MPP - Gradle設定

```
        implementation rootProject.ext.serializationCommon
        implementation rootProject.ext.ktorClient
        implementation rootProject.ext.ktorClientJson
    }
}
androidMain.dependencies {
}
iosMain {
    dependencies {
        implementation rootProject.ext.coroutineNat
        implementation rootProject.ext.serializationNative
        implementation rootProject.ext.ktorClientIos
        implementation rootProject.ext.ktorClientJsonIos
    }
}
jsMain {
    dependencies {
        implementation rootProject.ext.kotlinJs
        implementation rootProject.ext.coroutineJs
        implementation rootProject.ext.serializationJs
```

同じ名前

/common/build.gradle

MPP - Gradle設定

```
apply plugin: 'kotlin-multiplatform'
apply from: 'android.gradle'

kotlin {
  android()
  iosArm64() {
    binaries {
      framework()
    }
  }
  jvm()
  js()
  sourceSets {
    commonMain {
      dependencies {
        implementation rootProject.ext.kotlinCommon
        implementation rootProject.ext.coroutineCommon
        implementation rootProject.ext.serializationCommon
        implementation rootProject.ext.ktorClient
```

もし何も指定しない場合

/common/build.gradle

MPP - Gradle設定

```
        implementation rootProject.ext.serializationCommon
        implementation rootProject.ext.ktorClient
        implementation rootProject.ext.ktorClientJson
    }
}
androidMain.dependencies {
}
iosArm64Main {
    dependencies {
        implementation rootProject.ext.coroutinesNative
        implementation rootProject.ext.serializationNative
        implementation rootProject.ext.ktorClientIos
        implementation rootProject.ext.ktorClientJsonIos
    }
}
jsMain {
    dependencies {
        implementation rootProject.ext.kotlinJs
        implementation rootProject.ext.coroutineJs
```

フルネームを書く必要がある

/common/build.gradle

MPP - Gradle設定

```
        implementation rootProject.ext.serializationCommon
        implementation rootProject.ext.ktorClient
        implementation rootProject.ext.ktorClientJson
    }
}
androidMain.dependencies {
}
iosMain {
    dependencies {
        implementation rootProject.ext.coroutineNative
        implementation rootProject.ext.serializationNative
        implementation rootProject.ext.ktorClientIos
        implementation rootProject.ext.ktorClientJsonIos
    }
}
jsMain {
    dependencies {
        implementation rootProject.ext.kotlinJs
        implementation rootProject.ext.coroutineJs
        implementation rootProject.ext.serializationJs
```

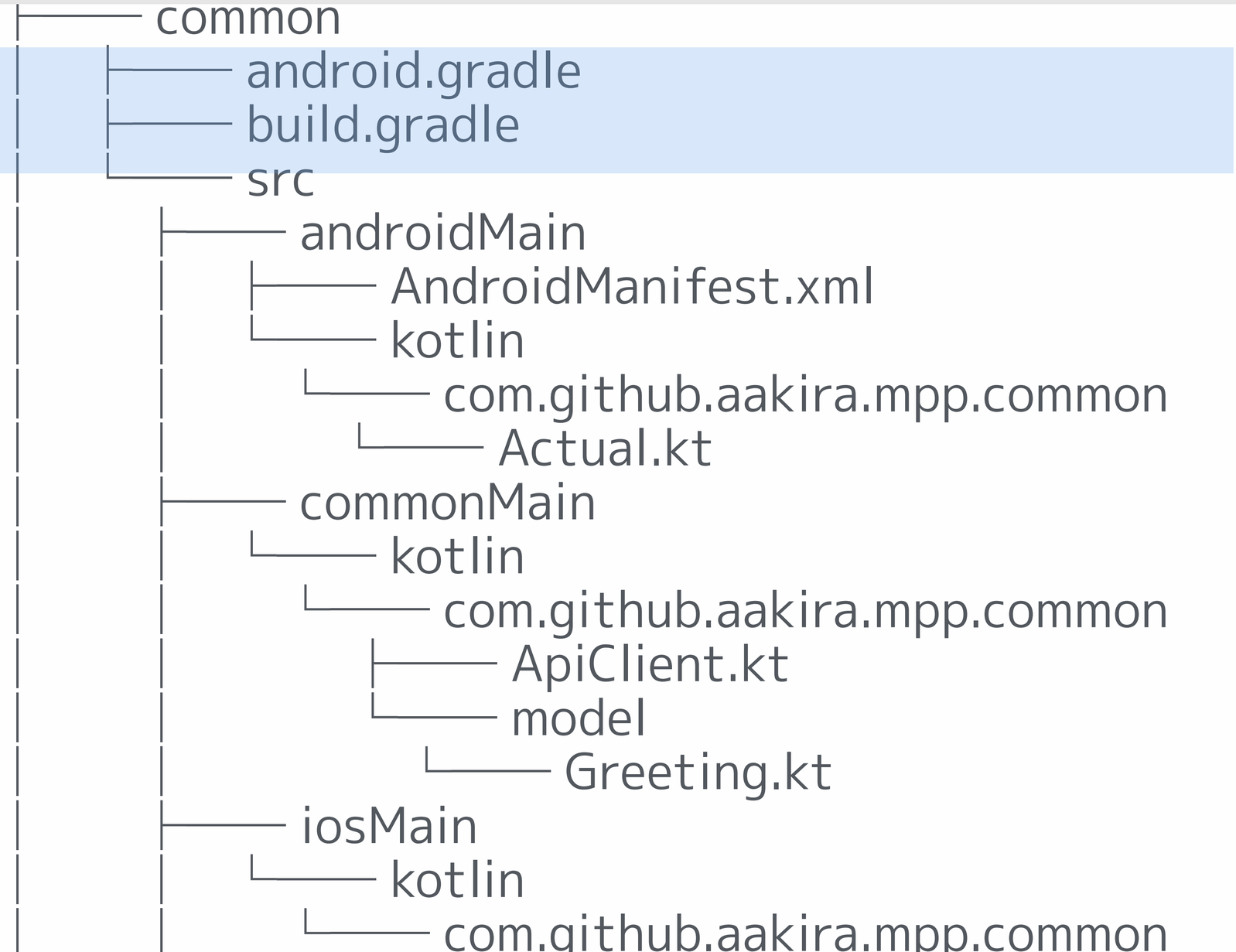
/common/build.gradle

MPP - Gradle設定

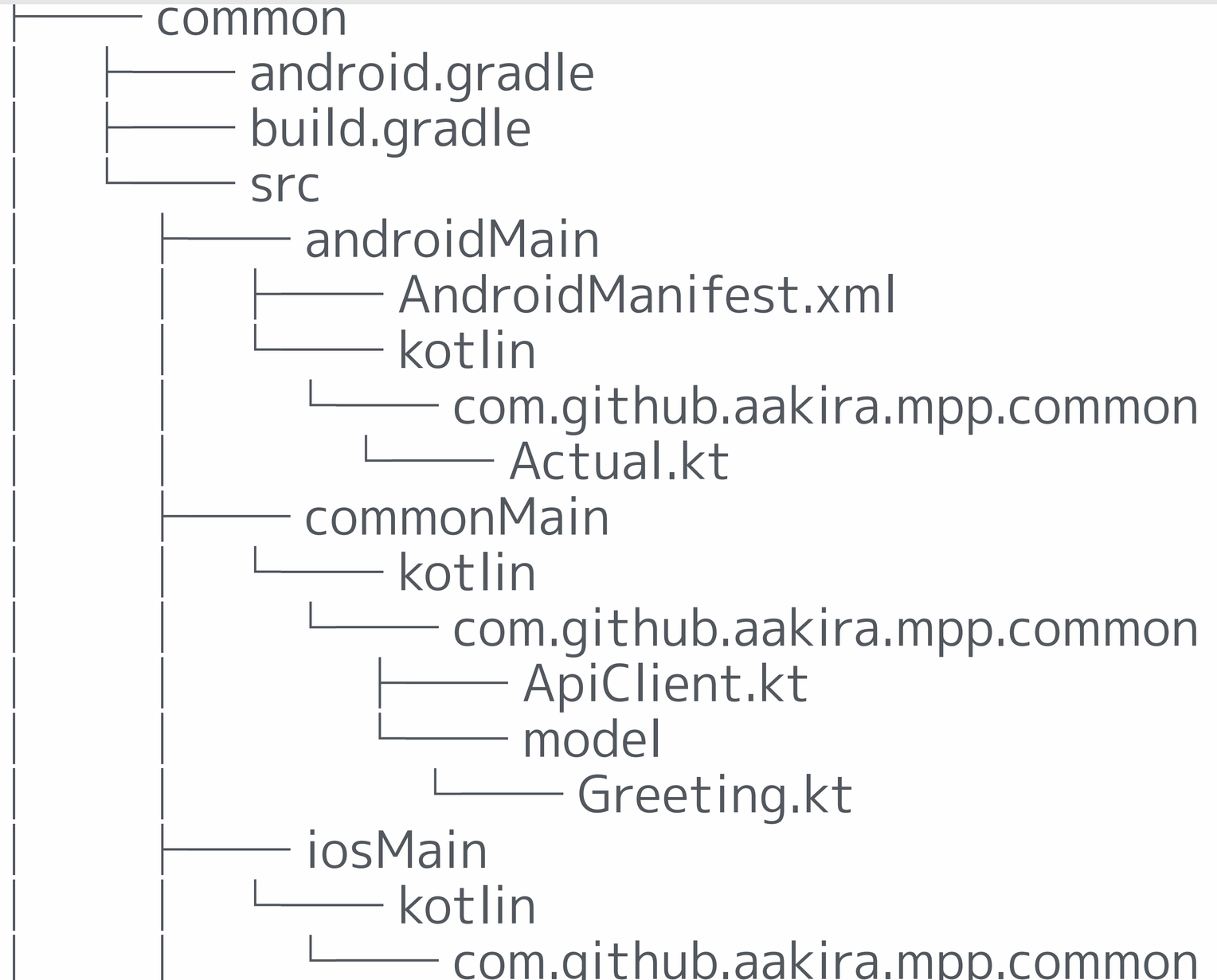
```
    jsMain {
      dependencies {
        implementation rootProject.ext.kotlinJs
        implementation rootProject.ext.coroutineJs
        implementation rootProject.ext.serializationJs
        implementation rootProject.ext.ktorClientJs
        implementation rootProject.ext.ktorClientJsonJs
      }
    }
    jvmMain {
      dependencies {
        implementation rootProject.ext.kotlinJvm
        implementation rootProject.ext.coroutine
        implementation rootProject.ext.serialization
        implementation rootProject.ext.ktorClientJvm
        implementation rootProject.ext.ktorClientJsonJvm
      }
    }
  }
```

/common/build.gradle

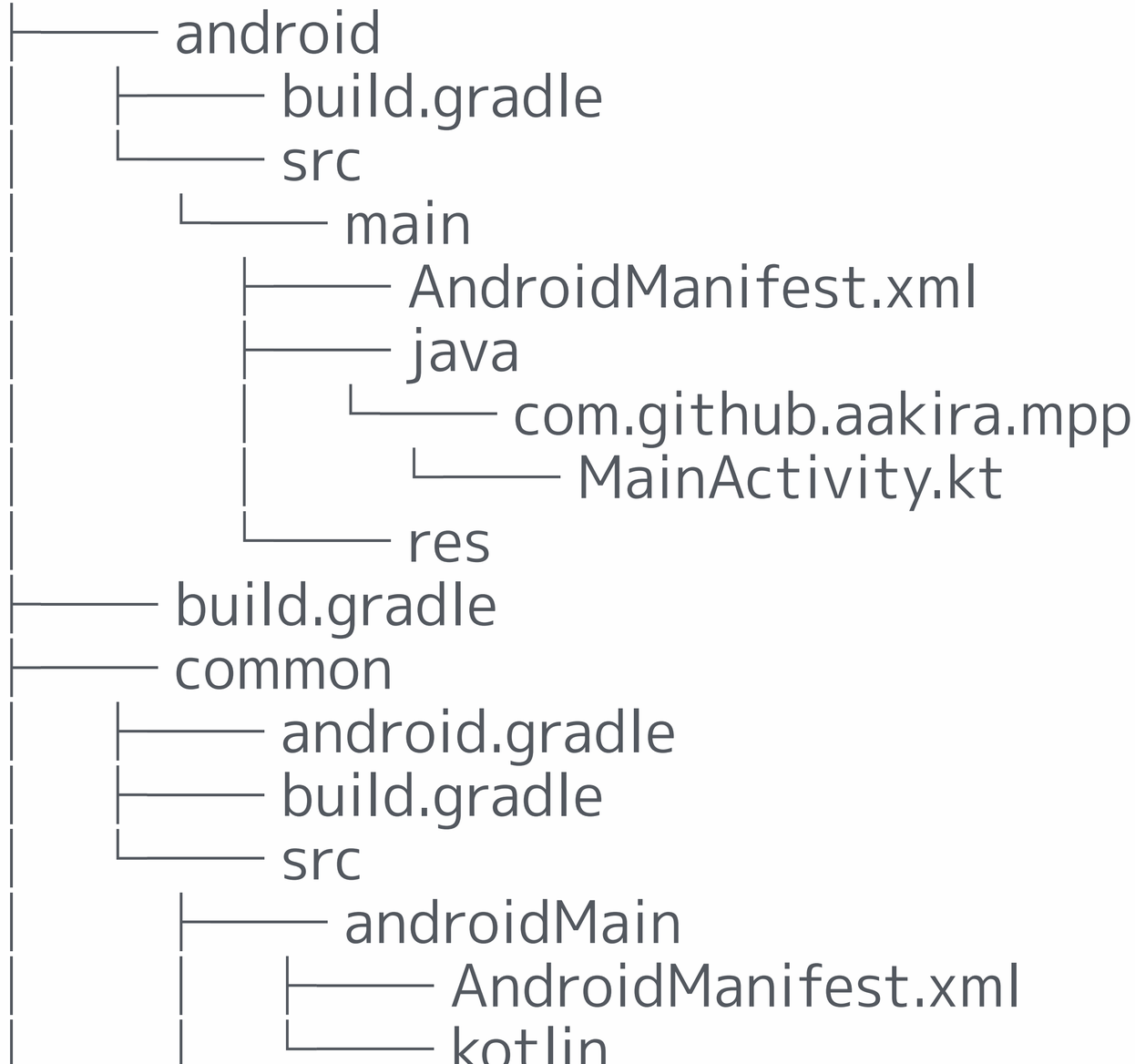
MPP - Gradle設定



MPP - Gradle設定



MPP - Gradle設定



MPP - Gradle設定

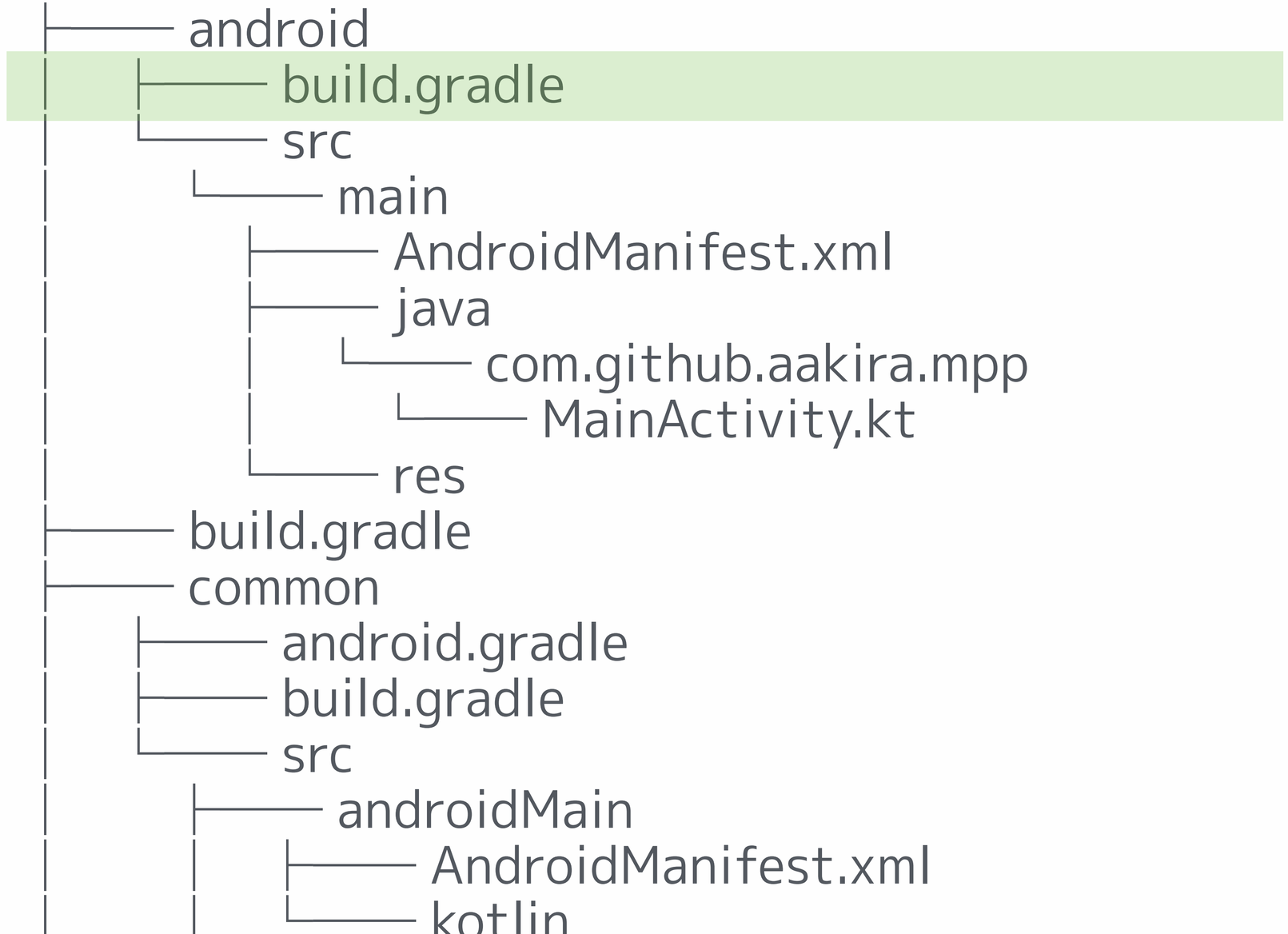
```
├── android
│   ├── build.gradle
│   └── src
│       └── main
│           ├── AndroidManifest.xml
│           ├── java
│           │   └── com.github.aakira.mpp
│           │       └── MainActivity.kt
│           └── res
├── build.gradle
├── common
│   ├── android.gradle
│   ├── build.gradle
│   └── src
│       ├── androidMain
│       │   ├── AndroidManifest.xml
│       │   └── kotlin
```

共通モジュールの読み込み

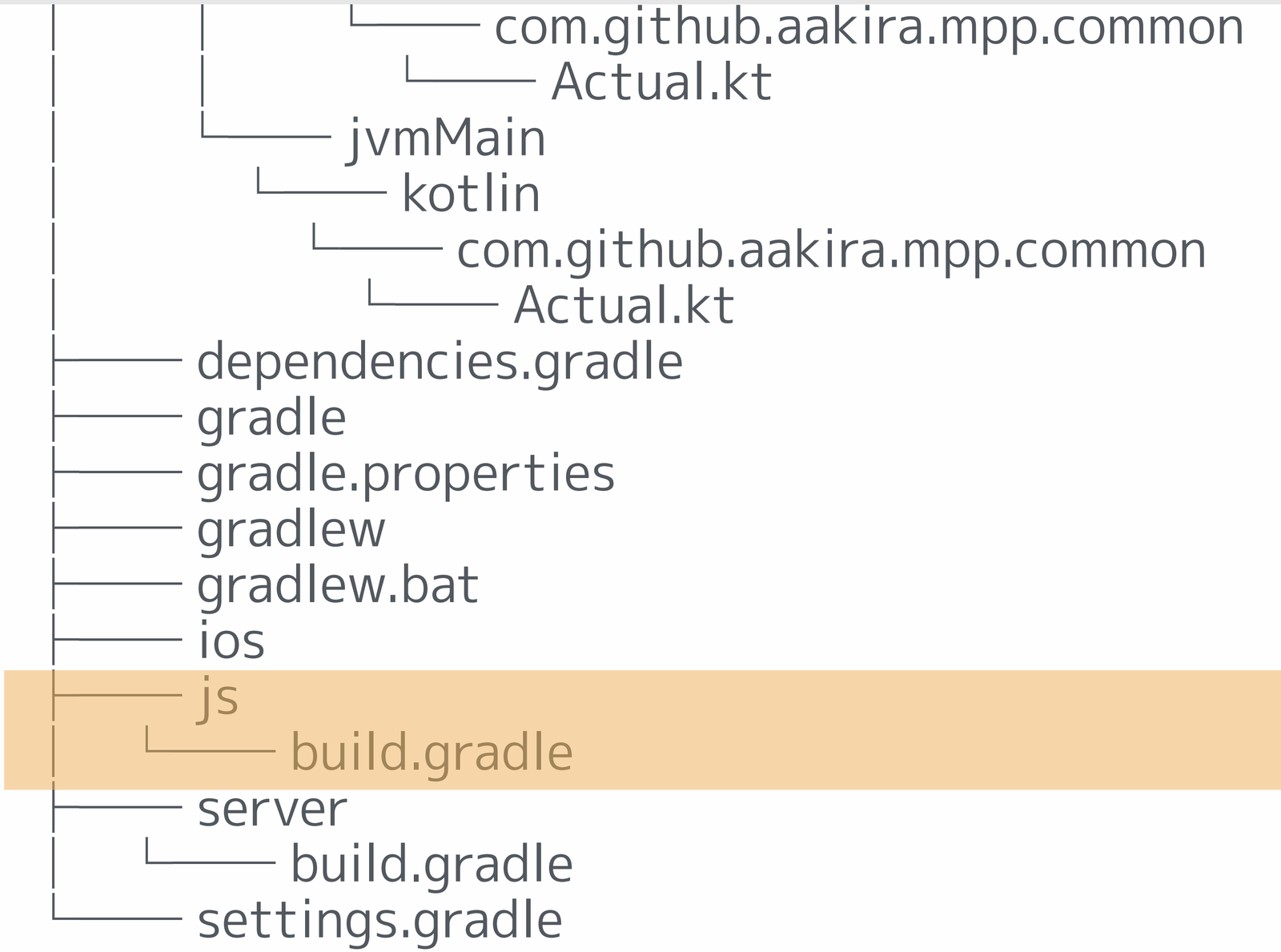
```
    buildTypes {
        release {
            minifyEnabled true
            proguardFiles getDefaultProguardFile('proguard-an
        }
    }
}

dependencies {
    implementation project(":common")

    implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk8:$
}
```

/android/build.gradle

共通モジュールの読み込み



共通モジュールの読み込み



共通モジュールの読み込み

```
plugins {
    id 'org.jetbrains.kotlin.js'
    id 'kotlin-dce-js'
}

kotlin {
}

[compileKotlinJs, compileTestKotlinJs].each { config ->
    config.kotlinOptions {
        moduleKind = 'umd'
        sourceMap = true
        metaInfo = true
    }
}

dependencies {
    implementation project(':common')

    implementation rootProject.ext.kotlinJs
}
```

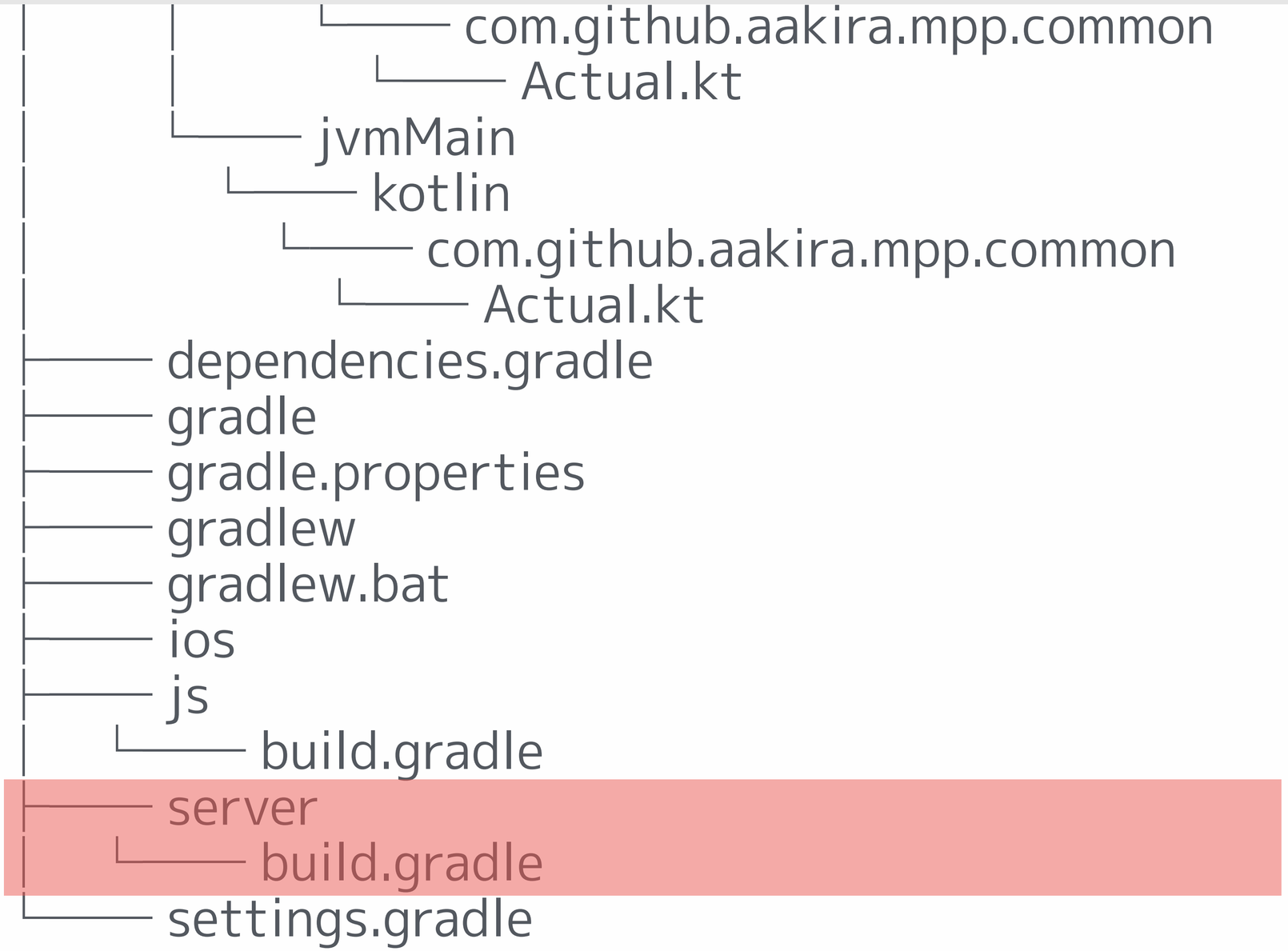
/web/build.gradle

共通モジュールの読み込み

```
        config.kotlinOptions {
            moduleKind = 'umd'
            sourceMap = true
            metaInfo = true
        }
    }

    dependencies {
        implementation project(':common')

        implementation rootProject.ext.kotlinJs
    }
```

/web/build.gradle

共通モジュールの読み込み



共通モジュールの読み込み

```
plugins {
    id 'kotlin'
    id 'application'
}

group 'com.github.aakira.mpp'
version '0.0.1'

mainClassName = "io.ktor.server.netty.EngineMain"

sourceSets {
    main.kotlin.srcDirs = main.java.srcDirs = ['src']
    main.resources.srcDirs = ['resources']
}

dependencies {
    implementation project(':common')

    implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk8:$kotlin_version"

    def ktor_server_version = "1.2.2"
    implementation "io.ktor:ktor-server-netty:$ktor_server_version"
    implementation "io.ktor:ktor-gson:$ktor_server_version"
    implementation "ch.qos.logback:logback-classic:1.2.3"
}
```

/server/build.gradle

共通モジュールの読み込み

```
mainClassName = "io.ktor.server.netty.EngineMain"

sourceSets {
    main.kotlin.srcDirs = main.java.srcDirs = ['src']
    main.resources.srcDirs = ['resources']
}

dependencies {
    implementation project(':common')

    implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk8:$kotlin_version"

    def ktor_server_version = "1.2.2"
    implementation "io.ktor:ktor-server-netty:$ktor_server_version"
    implementation "io.ktor:ktor-gson:$ktor_server_version"
    implementation "ch.qos.logback:logback-classic:1.2.3"
}
```

/server/build.gradle

呼び出しコード

共通モジュール(expect)

```
internal expect val hostName: String
internal expect val coroutineDispatcher: CoroutineDispatcher

class ApiClient {
    private val httpClient = HttpClient()

    fun getGreeting(successCallback: (Greeting) -> Unit, errorCallback: (Exception) -> Unit) {
        GlobalScope.apply {
            launch(coroutineDispatcher) {
                try {
                    val result = httpClient.get<String> {
                        url {
                            protocol = URLProtocol.HTTP
                            host = hostName // expect value
                            port = 8080
                        }
                    }
                    val greeting = Json.parse(Greeting.serializer(), result)
                    successCallback(greeting)
                } catch (e: Exception) {
                    errorCallback(e)
                }
            }
        }
    }
}
```

/common/ApiClient.kt

共通モジュール(expect)

```
internal expect val hostName: String
internal expect val coroutineDispatcher: CoroutineDispatcher

class ApiClient {
    private val httpClient = HttpClient()

    fun getGreeting(successCallback: (Greeting) -> Unit, errorCallback: (Exception) -> Unit) {
        GlobalScope.apply {
            launch(coroutineDispatcher) {
                try {
                    val result = httpClient.get<String> {
                        url {
                            protocol = URLProtocol.HTTP
                            host = hostName // expect value
                            port = 8080
                        }
                    }
                    val greeting = Json.parse(Greeting.serializer(), result)
                    successCallback(greeting)
                } catch (e: Exception) {
                    errorCallback(e)
                }
            }
        }
    }
}
```

/common/ApiClient.kt

共通モジュール(expect)

```
    launch(coroutineDispatcher) {
        try {
            val result = httpClient.get<String> {
                url {
                    protocol = URLProtocol.HTTP
                    host = hostName // expect value
                    port = 8080
                }
            }
            val greeting = Json.parse(Greeting.serializer(), result)
            successCallback(greeting)
        } catch (e: Exception) {
            errorCallback(e)
        }
    }
}
```

/common/ApiClient.kt

共通モジュール(expect)

```
internal expect val hostName: String
internal expect val coroutineDispatcher: CoroutineDispatcher

class ApiClient {
    private val httpClient = HttpClient()

    fun getGreeting(successCallback: (Greeting) -> Unit, errorCallback
(Exception) -> Unit) {
        GlobalScope.apply {
            launch(coroutineDispatcher) {
                try {
                    val result = httpClient.get<String> {
                        url {
                            protocol = URLProtocol.HTTP
                            host = hostName // expect/common/ApiClient.kt
```

共通モジュール(actual)

```kotlin
internal actual val coroutineDispatcher: CoroutineDispatcher = Dispatchers.IO
```

/common/androidMain/Actual.kt

```kotlin
internal actual val coroutineDispatcher: CoroutineDispatcher = Dispatchers.Default
```

/common/jsMain/Actual.kt

```kotlin
internal actual val coroutineDispatcher: CoroutineDispatcher = Dispatchers.Default
```

/common/jvmMain/Actual.kt

共通モジュール(actual)

```
internal actual val coroutineDispatcher: CoroutineDispatcher = Dispatchers.IO
```

/common/androidMain/Actual.kt

```
internal actual val coroutineDispatcher: CoroutineDispatcher = Dispatchers.Default
```

/common/jsMain/Actual.kt

```
internal actual val coroutineDispatcher: CoroutineDispatcher = Dispatchers.Default
```

/common/jvmMain/Actual.kt

共通モジュール(actual)

```
internal actual val coroutineDispatcher: CoroutineDispatcher =
    NsQueueDispatcher(dispatch_get_main_queue())

internal class NsQueueDispatcher(private val dispatchQueue: dispatch_queue_t) :
    CoroutineDispatcher() {
    override fun dispatch(context: CoroutineContext, block: Runnable) {
        dispatch_async(dispatchQueue) {
            block.run()
        }
    }
}
```

/common/iosMain/Actual.kt

共通モジュール(actual)

```kotlin
internal actual val coroutineDispatcher: CoroutineDispatcher =
    NsQueueDispatcher(dispatch_get_main_queue())

internal class NsQueueDispatcher(private val dispatchQueue: dispatch_queue_t) :
    CoroutineDispatcher() {
    override fun dispatch(context: CoroutineContext, block: Runnable) {
        dispatch_async(dispatchQueue) {
            block.run()
        }
    }
}
```

/common/iosMain/Actual.kt

共通モジュール(actual)

```
internal actual val coroutineDispatcher: CoroutineDispatcher =
    NsQueueDispatcher(dispatch_get_main_queue())

internal class NsQueueDispatcher(private val dispatchQueue: dispatch_queue_t) :
    CoroutineDispatcher() {
    override fun dispatch(context: CoroutineContext, block: Runnable) {
        dispatch_async(dispatchQueue) {
            block.run()
        }
    }
}
```

/common/iosMain/Actual.kt

Client(Android)

```kotlin
private val handler = Handler(Looper.getMainLooper())

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

    ApiClient().getGreeting(
        successCallback = {
            handler.post { helloText.text = it.hello }
        },
        errorCallback = {
            handler.post { helloText.text = it.toString() }
        })
}
```

/android/MainActivity.kt

Client(Android)

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

    ApiClient().getGreeting(
        successCallback = {
            handler.post { helloText.text = it.hello }
        },
        errorCallback = {
            handler.post { helloText.text = it.toString()

        })
}
```

/android/MainActivity.kt

Client(iOS)

```swift
override func viewDidLoad() {
  super.viewDidLoad()

  let label = UILabel(frame:
    CGRect(x: 0, y: 0, width: view.frame.size.width,
           height: view.frame.size.height)
  )
  label.textAlignment = .center
  label.font = label.font.withSize(26)
  self.view.addSubview(label)

  ApiClient().getGreeting(
    successCallback: { response in
      label.text = response.hello
    }, errorCallback: { error in
      print(error)
    })
}
```

/ios/ViewController.swift

Client(iOS)

```
    label.font = label.font.withSize(26)
    self.view.addSubview(label)

    ApiClient().getGreeting(
      successCallback: { response in
        label.text = response.hello
      }, errorCallback: { error in
        print(error)
      })
}
```

/ios/ViewController.swift

Client(WEB)

```
fun main() {
    ApiClient().getGreeting(
        successCallback = {
            document.body?.textContent = it.hello
        },
        errorCallback = {
            console.log(it.toString())
        }
    )
}
```

/web/Main.kt

Client(WEB)

```
fun main() {
    ApiClient().getGreeting(
        successCallback = {
            document.body?.textContent = it.hello
        },
        errorCallback = {
            console.log(it.toString())
        }
    )
}
```

/web/Main.kt

Client(WEB)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Mpp Sample</title>
    <script type="text/javascript" language="JavaScript"
            src="/build/kotlin.js"></script>
    ...(略)
    <script type="text/javascript" language="JavaScript"
            src="/build/mpp-common.js"></script>
</head>

<body>
<script type="text/javascript" language="JavaScript"
        src="/build/mpp-web.js"></script>
</body>
</html>
```

/web/resources/index.html

Client(WEB)

```
    <script type="text/javascript" language="JavaScript"
            src="/build/kotlin.js"></script>
    ...(略)
    <script type="text/javascript" language="JavaScript"
            src="/build/mpp-common.js"></script>
</head>

<body>
<script type="text/javascript" language="JavaScript"
        src="/build/mpp-web.js"></script>
</body>
</html>
```

/web/resources/index.html

MPPサンプル

https://github.com/AAkira/mpp-example

まとめ

まとめ

- MPPは人類が求めていた答え
- Gradleのライブラリモジュールとして提供されるので、
  基本的にデメリットは無い
- Gradle力が少しだけ必要
- いずれはKotlin/Everywhereに

宣伝 - 本を書きました

技術評論社から今秋発売予定

Android, Server, Test, Coroutine, **MPP**
を現場のエンジニアが解説

著者: 愛澤、荒谷、木原、仙波、前川

Have a nice Kotlin!

@_a_akira