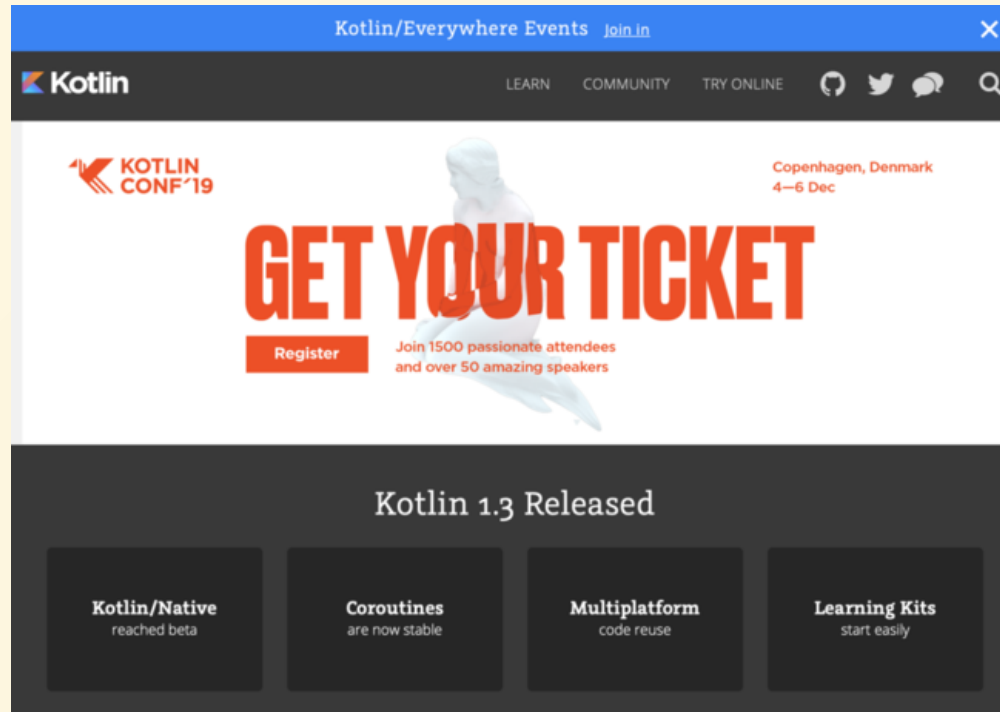


Kotlin Updates in LL2019

2019-08-24 [@eyasuyuki](#)

Kotlinとは(1/3)

<https://kotlinlang.org/>



Kotlinとは(2/3)

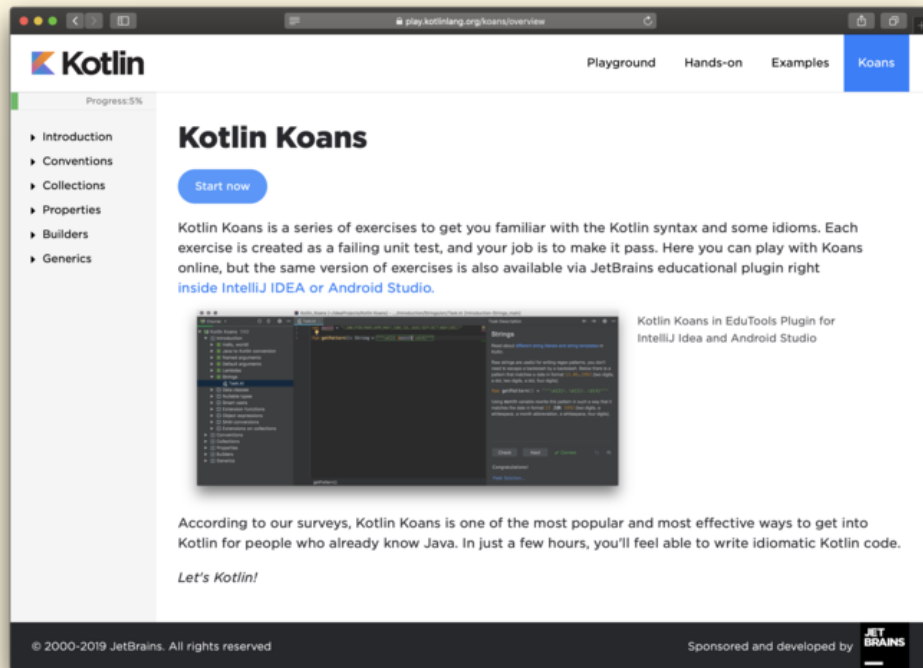
- JetBrainsが開発したコンパイラ言語
 - JVMバイトコードへのコンパイル
 - JavaScriptへのトランスパイル
 - LLVM中間表現へのコンパイル
- Null安全
- スマートキャスト
- データクラス (ボイラープレートからの解放)
- 高速なコンパイル (Scalaに対する優位性)

Kotlinとは(3/3)

- 関数型言語からの影響
 - ラムダ式
 - 高階関数
 - 末尾再帰
 - 多値
 - パターンマッチ
 - 型推論
 - ifなどが構文ではなく式(値を返す)
 - 中値記法

KotlinのTutorial

- Kotlin Koans <https://play.kotlinlang.org/koans/overview>
- Koan=禅の公案



Kotlin躍進のきっかけ

- 2017年のGoogle I/OでAndroidの開発言語として正式に採用された

Kotlinの歴史(1/2)

バージョン1.0未満は省略しました。

年月	バージョン	主なトピック
2016年2月	1.0	Javaとの完全互換 Java6バイトコードへのコンパイル
2017年5月		Androidの開発言語として正式採用
2017年3月	1.1	JavaScript対応 コルーチン(実験的) async/await, yield Java9サポート コンパイル速度の向上

Kotlinの歴史(2/2)

年月	バージョン	主なトピック
2017年11月	1.2	マルチプラットフォームプロジェクト(実験的)
2018年11月	1.3	コルーチン Kotlin/Native 契約(Contracts)

1.0から1.3までの大きなトピック

- マルチプラットフォーム対応
- コルーチン
- 契約(Contracts)

マルチプラットフォーム対応(1/2)

- 当初はJavaとの完全互換が大きなメリットだった
 - JavaとKotlinが混在しても動作するので部分的に移行することも可能
- 次いでJavaScriptへの対応が行われ、フロントエンドもKotlinで開発出来るようになった

マルチプラットフォーム対応(2/2)

- Kotlin/Nativeのリリースでネイティブコンパイルが可能になりJavaに依存しない道が開かれた
- 適用分野:
 - 当初はAndroid開発が主な用途
 - 現在ではサーバーサイドからフロントエンドまで全てKotlinで開発できる

マルチプラットフォームプロジェクト (実験的)

- 1つのプロジェクトで複数ターゲットのビルドが可能
- `build.gradle` などに記述
- `gradle init` でプロジェクトが作れる訳ではない(テンプレートがない)
- 参考文献: <https://kotlinlang.org/docs/reference/building-mpp-with-gradle.html>

Java対応

- 当初のメリット:
 - Java6でもラムダ式や関数型言語的なプログラミングができる
- Java8以降が広く使われるようになり当初のメリットは薄れた
- Kotlin自体もJava8以降のバイトコードに対応できるようになった

JavaScript対応

- とりあえずKotlinからJavaScriptにトランスパイルはできる
- TypeScriptやDartと比べて便利かというと...🤔
- Kotlin/NativeでWebAssemblyにコンパイルする方が有望...?
- 参考文献: <https://speakerdeck.com/subroh0508/jstoge-dou-sinagarakotlinfalseiketeruwen-fa-woxue-bu>

Kotlin/Native(1/2)

- KotlinからLLVM中間表現へコンパイル
 - Linux
 - Windows
 - Android NDK
 - iOS
 - macOS
 - WebAssembly

Kotlin/Native(2/2)

- 当然ながらJavaのライブラリは使えない
 - Kotlinで記述した標準ライブラリの整備が進んでいる
- メリット: iOSアプリがKotlinで書ける
- 参考文献:
<https://www.slideshare.net/TakakiHoshikawa/kotlinnative>

コルーチン(1/2)

- Kotlinのコルーチンはノンブロッキングで実行される関数
- 関数の途中で処理を中断したり再開したりできる
- `async`
 - `Deferred<T>` を返すコルーチンビルダー関数
 - コルーチンビルダー関数の中はノンブロッキングで実行される
- `await`
 - `Deferred<T>` のサスペンド関数
 - サスペンド関数はコルーチンの実行を中断する

コルーチン(2/2)

- `yield`, `yieldAll`
 - サスペンド関数。その時点で中断して値を返す
- Channel
 - コルーチン間で値を送受信できるキュー (like Golang)

async(1/2)

```
println("start async")
GlobalScope.async {
    // ノンブロッキングで実行されるため実行前にmain()が終わる
    println("Inside async")
}
println("end async")
```

実行結果:

```
start async
end async
```

async(2/2)

```
println("start async")
GlobalScope.async {
    println("Inside async")
}
Thread.sleep(1000) // これがないとasync実行前にmain()が終わる
println("end async")
```

実行結果:

```
start async
Inside async
end async
```

await(1/2)

```
val deferred = GlobalScope.async {  
    println(App().greeting)  
}  
deferred.await()
```

コンパイルエラー:

```
> Task :compileKotlin FAILED  
e: /Users/yasuyuki/git/KotlinUpdates/src/main/kotlin/org/j  
avaopen/kotlin/updates/App.kt: (37, 17): Suspend function  
'await' should be called only from a coroutine or another  
suspend function  
  
FAILURE: Build failed with an exception.
```

await (2/2)

```
println("start runBlocking")
runBlocking {
    val defferd = GlobalScope.async {
        println(App().greeting)
    }
    defferd.await()
}
println("end runBlocking")
```

実行結果:

```
start runBlocking
Hello world.
end runBlocking
```

asyncな関数(1/2)

```
fun hello() = GlobalScope.async {  
    "Hello, async."  
} // Deferred<String>を返す
```

asyncな関数(2/2)

使い方:

```
println("start async function")
runBlocking {
    println(hello().await())
}
println("end async function")
```

実行結果

```
start async function
Hello, async.
end async function
```


契約(Contracts)

- スマートキャストの判定を改善するための宣言
- 関数が呼び出された後の状態を制約してスマートキャストが効くようにする

スマートじゃないキャスト(Java)

```
public void action(Animal animal) {  
    if (animal instanceof Cat) {  
        Cat cat = (Cat)animal;  
        cat.scratch();  
    } else if (animal instanceof Dog) {  
        Dog dog = (Dog)animal;  
        dog.bark();  
    }  
}
```

スマートキャスト

```
fun action(animal: Animal) {  
    when (animal) {  
        is Cat -> animal.scratch() // Catであることは自明なのでCatのメソッドが呼べる  
        is Dog -> animal.bark() // Dogであることは自明なのでDogのメソッドが呼べる  
    }  
}
```

契約(Contracts)のおさらい

- スマートキャストの判定を改善するための宣言
- 関数が呼び出された後の状態を制約してスマートキャストが効くようにする
- 参考文献: <https://speakerdeck.com/ntaro/kotlin-contracts-number-m3kt>

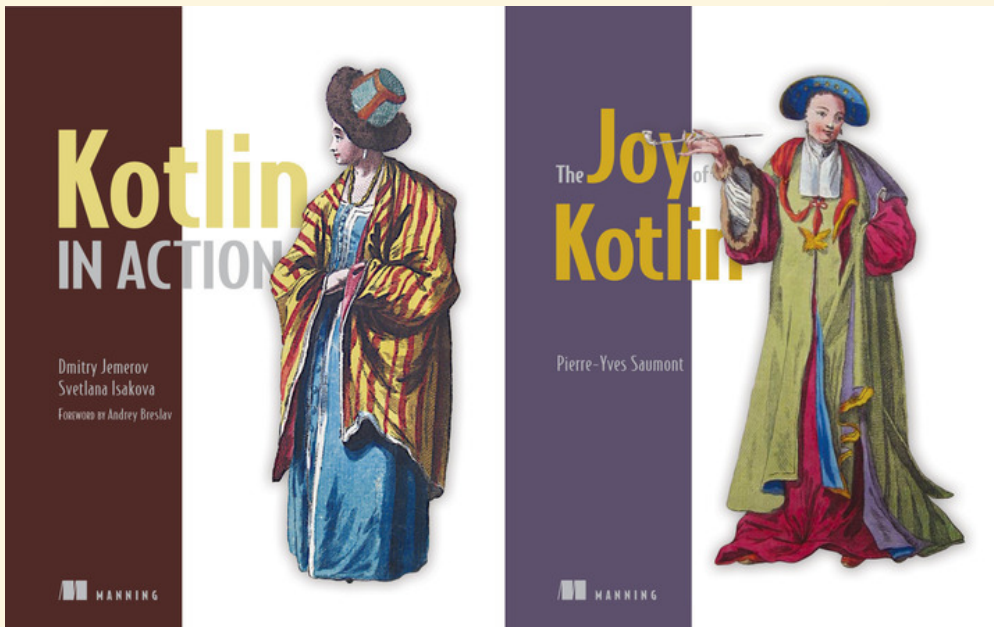
Kotlinの書籍(1/2)

- 長澤太郎 「Kotlinスタートブック」 2016 (通称赤べこ本)
- 長澤太郎 「Kotlin Webアプリケーション」 2017



Kotlinの書籍(2/2)

- Dmitry Jemerov/Svetlana Isakova "Kotlin in Action" 2017 (訳書あり)
- Pierre-Yves Saumont "Joy of Kotlin" 2019



このスライドのソースとサンプルコード

<https://github.com/eyasuyuki/KotlinUpdates>