6L-04

Kotlin特有の機能を選択的に導入できるJava-to-Kotlinコンバータの設計と実装

河野一真† 川端英之† 弘中哲夫†
広島市立大学大学院情報科学研究科†

1 はじめに

Androidアプリケーション開発の推奨言語であるKotlinは，Javaとの連携が容易であり，簡潔かつ安全なコードが書ける [1] という利点からJavaからの移行が増えてきている [2]．本研究は，ユーザのニーズに応じた変換が可能なJavaからKotlinへの変換システムの実現を目的とした自動変換ツールJ2KConverterを開発する．J2KConverterは従来のIDE搭載コンバータと異なり，Javaとの互換性を優先した変換を行う基本機能をベースに，Kotlin独自の機能を活かした変換を選択的に適用出来る拡張機能を持つ．ユーザはJ2KConverterの使用により，目的に応じた変換結果を得ることが出来る．本発表では，J2KConverterの設計及び実装について述べる．

2 Java-to-Kotlin自動変換の現状の問題点と改善案

IDE搭載のコンバータによる自動変換後のKotlinプログラムの修正が必須であることは，Intellij IDEA付属コンバータを使用したKotlinへの変換の方法を記しているGoogle Developers Codelabsにも述べられている [4]．実際，Google Samples[5] のJavaプロジェクトをKotlinに自動変換した結果，多くの修正が必要だった．例えば，Kotlinではnull-safety機能により変数の型がnon-null型とnullable型に分けられ，後者に対する参照の際に後置演算子を用いて値がnullの場合の処理を明記する必要がある．だがIDEのコンバータでは，nullable型変数を参照するコードの一部において後置演算子が記述されない．

ユーザの思惑通りの出力を自動的な変換で得ることは難しく，変換後のプログラムに対する修正の要求はユーザの負担を増加させる．

これに対し，KotlinとJavaの互換性の高さに着目する．KotlinとJavaの文法は類似度が高く，変換の際にアルゴリズムレベルの大幅な変更を伴うコードは多くはないことから，本研究では以下のように考える．

- 特有の機能を可能な限り使用せず，互換性を優先することで，自動変換はおよそ可能である．
- アルゴリズムの変更を伴うような変換は，文法記述上不可避でないケースを除き，ユーザの意思が介在すべきである．

我々の開発するJ2KConverterが，基本機能と拡張機能に分けて実装を行い，ユーザに選択の余地を与えているのは，上記の考えに基づいてのものである．

3 J2KConverterの構想

J2KConverterにおける拡張機能では，Kotlinで導入された機能のうち，null-safety機能及びカスタムアクセサへの対応を選択可能にする．以下導入する拡張機能1〜3について述べる．null-safety機能への対応 (拡張1, 3) では，変数のnon-null化をサポートする．カスタムアクセサへの対応 (拡張2) では，Javaにおける自作アクセサ記述の排除を支援する．

基本 Kotlinへの変換を実現する．変数は全てnullable型と判断し，後置演算子は非null値アサーション演算 “!!”を使用する．

拡張1 データフローを考慮した上で、nullになり得ないと判断できるものをnon-null型に変換，もしくはユーザにこの変換を提案する．

拡張2 カスタムアクセサとして記述出来るアクセサメソッドに対し，カスタムアクセサへの変換を行う．

拡張3 プログラム中の参照している値がnullであるかどうかのチェックの記述に応じて，Kotlin変換時の後置演算子をセーフコール演算子 “?” やエルビス演算子 “?:”に変換できるようにする．

基本機能は，Javaとの互換性を重視しつつKotlinプログラムへの変換を実現すること，拡張機能はKotlinの文法上可能な書き方の中で，よりKotlinの仕組みを活かしたプログラム変換を提供することを目的としている．

拡張1ではデータフローを考慮した結果null値を取り得ない変数に対してnon-null型の変数へ変換することを行う．

拡張2では，通常のアクセサメソッドと違う動作をしつつも，アクセサメソッドとして機能しているメソッドをカスタムアクセサ化し，kotlin変数宣言時に暗黙のうちに定義されているアクセサメソッドの代わりに定義する変換を行う．

拡張3ではnullable型変数に変換すべきと判断した変数の参照に対して，よりnull-safety機能を活かした変換を行う．具体的には，参照される際の値がnullであるかどうかのチェックの記述に応じて，後置演算子を “!!”ではなく “?”や “?:”に出来るか否かを判断し，変換の出力を行う．

なお，拡張機能はそれぞれ独立である．例えば拡張3の機能を使用する際，拡張1の機能を使用して変換することも，使用せずに変換することも可能である．

Design and Implementation of a Java-to-Kotlin converter that supports selective incorporation of Kotlin-specific features
Kazuma Kouno† Hideyuki Kawabata† Tetsuo Hironaka†
†Graduate School of Information Sciences, Hiroshima City University

4 J2KConverterの設計と実装

J2KConverterの内部を図1に示す.

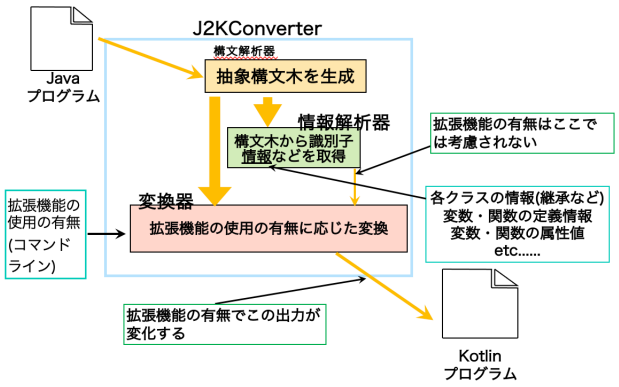

図 1: 外部設計・内部構造

J2KConverterは，あるディレクトリ内のプロジェクトを構成するJavaプログラムに対してまとめて変換を行う．その際，データフロー解析を用いた情報収集を行うため，内部の構造は2つに分かれる．

情報解析器 データフロー解析による情報収集を反復的に行う．解析は主に変数参照を行う箇所と変数への代入を行う箇所に行われ，各変数においてRead After Writeが保証されているか，またnullを取り得るか否かを判断する．

変換器 Kotlinプログラムを生成する．変換は構文木の捜査で行われ，文字列として保持した変換後のコードをファイルに書き込む．変換の際には拡張機能の使用の有無に応じて必要な情報を参照する．

例えば，図2(a)のコードを，基本機能だけで変換した結果は図2(b)に，拡張1と拡張3の機能を用いて変換した結果は図2(c)になる．変数`str`は初期値がないが，コンストラクタにおいて何かしらの値が代入されるため，外部からのアクセス時には必ず値を保持している．また，関数`func`内での代入もnullチェックをしているため，拡張1の機能を使用することで，lateinit修飾子を用いて宣言したnon-null型の変数として変換することが出来る．

一方変数`moji`は初期値がなく，場合によっては代入よりも先に変数の参照が行われる．また仮引数からの代入によりnull代入の可能性が存在する．よって初期値nullのnullable型の変数に変換し，関数呼び出しの際には演算子“`!!`”を用いる．拡張3の機能を用いて変換を行った場合は，関数呼び出しの際の演算子に“`?`”を用い，`moji`の値がnullだった際の処理にエルビス演算子を用いる．

開発環境は以下の通り：Intel Core i7, macOS 12.5, IntelliJ IDEA 2022.2.4 (Community Edition), Kotlin version 1.6.0, JavaParser 3.23.1, JVMTarget = 15.

5 まとめ

本稿ではJavaからKotlinへの自動変換機能J2KConverterの提案を行った．J2KConverterはJavaとの互換性を優先した変換を行う基本機能をベースに，Kotlin独自の機能を活かした変換を選択的に適用出来る拡張機能を持つ．

```
public class Sample{
    String str, moji;
    public Sample(){
        str = "nothing";
    }
    public Sample(String moji){
        str = "something";
        this.moji = moji;
    }
    void func(String str2){
       int len =
         moji != null ? moji.length() : -1;
      String tmp = str2 != null ? str2 : "";
    }
}
```

(a) 入力のJavaプログラム

```
class Sample {
    var str: String? = null
    var moji: String? = null
    constructor() {
        str = "nothing"
    }
    constructor(moji: String?) {
        str = "something"
        this.moji = moji
    }
    fun func(str2: String?) {
      val len
       =if(moji!! != null) moji!!.length else -1
       val tmp = if(str2!! != null) str2 else ""
    }
}
```

(b) 拡張機能を使用しなかった場合の出力Kotlinプログラム

```
class Sample {
    lateinit var str: String
    var moji: String? = null
    constructor() {
        str = "nothing"
    }
    constructor(moji: String?) {
        str = "something"
        this.moji = moji
    }
    fun func(str2: String?) {
        val len = moji?.length ?: -1
        val tmp = str2 ?: ""
     }
}
```

(c) 拡張機能を使用した場合の出力Kotlinプログラム

図 2: 入出力コードの例

これにより，ユーザの目的に応じた変換出力の提供を可能としている．現時点では，基本機能と拡張1の機能の実装がされており，拡張2は実装途中，拡張3は構想段階である．これらの機能の実装や新たな拡張機能の構想J2KConverterを評価するための客観的な指標の作成は今後の課題である．

参考文献

[1] R. Shibata, et al.: Java Android Application Performance Improvement by Kotlin DEX Bytecode Analysis without JIT Compiler, ICCE-Taiwan, 2020.

[2] M. Martinez and B.G. Mateus: How and Why did developers migrate Android Applications from Java to Kotlin? A study based on code analysis and interviews with developers, arXiv:2003.12730, Mar. 2020.

[3] Kotlin Foundation: Null Safety, Kotlin v1.4.21 Docs, https://kotlinlang.org/docs/reference/null-safety.html

[4] Converting to Kotlin: Handling Nullability, https://developer.android.com/codelabs/java-to-kotlin#5

[5] https://github.com/googlesamples. googlesamples.