

2K-03 Java から Kotlin へのプログラム変換支援ツールの設計と実装

河野一真† 川端英之‡ 弘中哲夫‡

広島市立大学情報科学部情報工学科† 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いる [2]. しかしながら, Android Studio などの IDE の提供する自動コンバート機能での変換はユーザーの意図に沿わない変換になる可能性があり, その修正が煩雑になりがちである. これに対し, 我々は Java で開発されたアプリケーションの Kotlin へのスムーズな移行を支援することを目的とした変換支援ツールの開発を行う. 本論文のツールは既存のコンバータを使用して Java から Kotlin へのコンバートを行う前に, コンバートで問題が生じる箇所を指摘する. 具体的には, Kotlin への変換に際し nullable にせざるを得ない変数に対しての注意喚起やクラスの継承関係を考慮したフィールド名やアクセサの妥当性の確認などを行う. 本発表では, 本ツールの設計及びそのプロトタイプの実装について述べる.

2 Java-to-Kotlin 変換支援ツール

2.1 変換支援の必要性

Java から Kotlin へのコンバートを行った後のコードの修正が煩雑になりがちな理由として, 主に以下の 2 つが挙げられる.

1. Kotlin の目的である null 参照をなくしたコードが生成出来ない
2. Java の時と全く同じ動作を行うコードにならないことがある

例えば, 1 つ目の事例に該当するものとして, 既存コンバータでは初期化が宣言時にされていないフィールドの一部は, null を許容する nullable 型になってしまうことが挙げられる. Kotlin は null 参照による `NullPointerException` の可能性を排除することを目的としている言語である [3] のに対し, 機械的なコンバートによって null の可能性を内包する nullable 型が大量に存在する状態は言語の目的に反するものである.

また, 2 つ目の事例に該当するものとして, サブクラスにおいてスーパークラスで定義された set メソッドを使用している際に, パラメータの名前によってはコンバートが上手くいかず, `Unresolved Reference` もしくは `Val`

cannot be reassigned のエラーメッセージが出てくることがある.

Google Samples の Java プロジェクトを Kotlin にコンバートした結果, 確認できたエラーコードの多くが例に挙げた事例に当てはまるものだった. このことから, コンバート後の変換支援が必要であると分かる.

2.2 変換支援の方法

機械的なコンバートによってコードを自動変換することは, ユーザーの意図に沿わない変換になる可能性がある. このため, 本論文のツールでは, コンバートの作業を行う前にユーザーに対して修正が必要になる箇所とその修正案を提示することで, ユーザーに修正を促すという方法をとる.

ここでユーザーの支援を行う際に必要な情報は以下のものである.

- コンバート後に修正が必要となり得るコードの場所
- どういったエラーが出るか
- 可能ならば Java の時点で可能な修正案, コンバート後にしか対処不可ならコンバート後の修正案

なお, この時に修正案が複数ある場合はそれらを全てユーザーに提示し, どれを採用するかはユーザーの判断に委ねる.

修正が必要な記述は複数のプログラムファイルにまたがる可能性があるため, 修正箇所の検出は複数のファイルにまたがる Java プログラムを同時に扱える必要がある.

2.3 変換支援ツールのプロトタイプの仕様

本ツールの主要な機能は以下の 2 つである.

- プログラム内に Java to Kotlin コンバートによって修正が必要になり得る箇所があるかどうかを判断する機能
- 修正が必要と判断した場合に, 想定されるエラーメッセージと適切な修正案を出力

ツールへの入力, コンバートを行う予定の Java プログラムが入ったディレクトリの絶対パスとする. なお, このプログラムを含む Java プロジェクトがビルドを行うことが出来ない場合は, このツールの処理対象としない. また, 今回は jar などで提供されるライブラリに依存したコードに対しての判定には対応しない.

ツールの出力に関しては次のように定める. まず, 入力として読み込まれる Java プログラム中の Kotlin 変換後に支障をきたす記述を本稿では「注意喚起対象」と呼ぶことにする. そして注意喚起対象に対して, 該当ソースコードの箇所, 予想されるエラーメッセージ, ソースコードの書き換えを促すメッセージ等を出力する.

Design and Implementation of a Tool for Supporting Java-to-Kotlin Conversion

Kazuma Kouno† Hideyuki Kawabata‡ Tetsuo Hironaka‡

†Department of Computer and Network Engineering, Hiroshima City University

‡Graduate School of Information Sciences, Hiroshima City University

```

finished field checking
check start:case1
"AppCompatActivity" is probably a library. Please check the detail of the library if necessary.
check finished:case1
check start:case2
check finished:case2
check start:case3
check finished:case3
check start:case5
check finished:case5
check start:case7
line 192
This code may give the following error after converting to Kotlin: Val cannot be reassigned.
It is recommended to rename the argument "intent" in the method "onNewIntent".
check finished:case7
check finished:MainActivity

```

`public class MainActivity extends AppCompatActivity`

`@Override
protected void onNewIntent(Intent intent){
 this.setIntent(intent);`

図 1: 注意喚起対象 3 及び 5 の検出ケースがある時の動作の様子 (ライブラリ依存のコードの可能性あり)

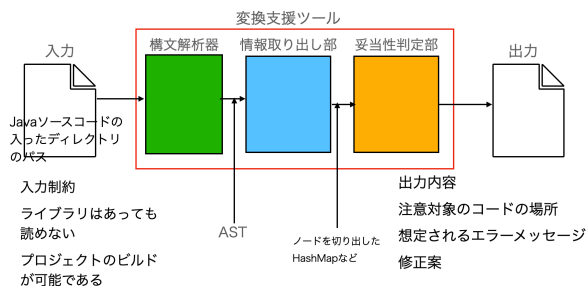


図 2: 外部設計・内部構造

3 Java-to-Kotlin 変換支援ツールの設計と実装

3.1 システム構成

このツールの内部は図 2 に示す通り 3 つに分かれる。

1. 構文解析器

ここでは、入力として与えられたパスが指し示すディレクトリの中に入っている複数のプログラムを 1 つ 1 つ解析し、抽象構文木を生成する。

2. 情報取り出し部

ここでは、構文解析器で作成した抽象構文木を visitor パターンを使用して辿り、判断に必要な情報を取り出して ArrayList や HashMap に保存する。判断に必要な情報は以下の通りである。

- 各クラスのフィールド/メソッド情報
- 各クラス間の継承関係
- インターフェース、実装クラスの関係

3. 妥当性判定部

ここでは、取り出した情報を元にプログラム一つ一つについてチェックを行い、注意喚起対象がプログラム内に存在するかどうかを見ていく。

この 3 つに関して、構文解析器では Java ファイル単位、情報取り出し部と妥当性判定部はクラス・インターフェース単位で動作を行う。

3.2 本ツールでの注意喚起対象の詳細

構文解析器と情報取り出し部を用いて取り出した情報を元に、注意喚起対象が存在するかどうかを妥当性判断部にてクラス単位で判断する。

ユーザーに注意を促す注意喚起対象を以下に示す。

1. 初期化が行われていないフィールド宣言
2. 継承しているクラスもしくはインターフェースに同じ名前の独立したフィールドが存在するクラス定義
3. inner/nested クラスで定義された private メソッドの enclosing クラスでの使用
4. getter/setter メソッドとフィールド名の結び付き
5. スーパークラスもしくはインターフェースに getter/setter メソッドを持つクラスのメソッド呼び出し箇所とそのパラメータの名前

3.3 実装環境・実行例

実装環境は以下の通りである。

- Intel Core i7
 - macOS Big Sur 11.1
 - IntelliJ IDEA 2020.2.4
 - Kotlin version 1.4.20-RC
 - JavaParser 3.16.1
 - JVMTarget = 1.8
- また、動作の様子を表したものを図 1 に示す。

4 まとめ

本論文の変換支援ツールを Google Samples のプログラムに適用したところ、3.2 節で挙げた注意喚起対象への対処が行えることを確認できた。しかしながら、3.2 節に列挙したもの以外にもコンバート後に修正が必要になり得る状況が見出された。この箇所を検出できるようにすること及びツール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。また、IDE のプラグインとしての実装やライブラリを読み込む機構の導入も検討事案である。

参考文献

- [1] Ryoichi Shibata, Atsuya Sonoyama, Masato Oguchi, Takeshi Kamiyama, Akira Fukuda, Saneyasu Yamaguchi: Java Android Application Performance Improvement by Kotlin DEX Bytecode Analysis without JIT Compiler, IEEE International Conference on Consumer Electronics - Taiwan (ICCE-Taiwan), 2020.
- [2] Matias Martinez, Bruno Gois 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>