2J-01

# IntentServiceを用いた非同期処理へのAndroidアプリケーションリファクタリング手法

萩原優樹　高田眞吾
慶應義塾大学 理工学部 情報工学科

## 1　序論

Androidアプリケーション（以下，アプリ）の市場においてUIの応答性などアプリのパフォーマンスはユーザの印象に大きな影響を与える．しかし多くのアプリはパフォーマンスを悪化させるバグ（以下，パフォーマンスバグ）に悩まされている．そのバグを取り除く方法としてAsyncTaskやIntentServiceを用いて非同期処理を行うことが挙げられる．本研究では，パフォーマンスバグとなりうる箇所をリファクタリングすることにより，IntentServiceを適用する手法を提案し，これを支援するツールを実装した．

## 2　背景

AndroidはLinuxベースのオープンソースのOSであり，その処理形式はシングルスレッドモデル（1つのスレッド上で処理を行う形式）を採用している．Androidアプリでは1つのメインスレッド上でUIウィジェットのイベントハンドラなどが実行され，処理が進む．このメインスレッド上で時間のかかる処理（ダウンロード処理など）を行うとメインスレッドがブロックされ，その後の処理に進まなくなってしまう．メインスレッドがブロックされた状態ではUIの表示遅延などが発生し，パフォーマンスバグに繋がる [5]．この事態を防ぐためには，メインスレッドとは別のスレッド上で時間のかかる処理を行うこと（非同期化）が必要となる．

## 3　関連研究

### 3.1　パフォーマンスバグの特定

Thanapornら [4] はアプリ内のパフォーマンスバグを特定するツール（Responsiveness Analysis Tool）を作成した．このツールはコードを静的に解析し，メインスレッドをブロックする可能性があるメソッド（以下，ブロッキングメソッド）の特定を行う．メインスレッドをブロックし得るAPIとしてShengqianら [3] はネットワーク，ストレージ，データベース，ビットマップの4つのAPIを挙げた．Responsiveness Analysis Toolはそれら4つのAPIに属するメソッドをコード中から特定する．

### 3.2　非同期処理へのリファクタリング

San Miguelら [2] はブロッキングメソッドを特定し，AsyncTaskを用いた非同期処理へとリファクタリングするツール（Asyncfactor）を作成した．なお，ブロッキングメソッドの特定にはResponsiveness Analysis Tool[4]を用いている．しかしAsyncTaskはアクティビティ（アプリのUI）が終了すると中断されてしまうため，長時間の処理の非同期化には向かない．そこで，Linら [1] はAsyncTaskをIntentServiceにリファクタリングする手法について研究を行った．IntentServiceはアクティビティの状態に関係なく動作し，自分の仕事がなくなるまで非同期処理を行うことができる．そのため，IntentServiceを使用することで数秒以上かかるような長時間の処理の非同期化が可能である．

San Miguelら [2] の研究ではブロッキングメソッドをAsyncTaskで非同期化し，Linら [1] はAsyncTaskをIntentServiceに置き換えることで非同期化していた．しかし，これら既存研究ではブロッキングメソッドをIntentServiceで直接非同期化することはできないという課題がある．

## 4　提案

本研究ではブロッキングメソッドをコード中から特定し，IntentServiceを用いた非同期処理へとリファクタリングを行う手法を提案する．提案手法のアーキテクチャを図1に示す．

---

**Refactoring Android Applications for Asynchrony using IntentService**
Yuki Hagiwara, Shingo Takada
Department of Information and Computer Science, Faculty of Science and Technology, Keio University
223-8522, Kanagawa, Japan
yuki_hagiwara@keio.jp

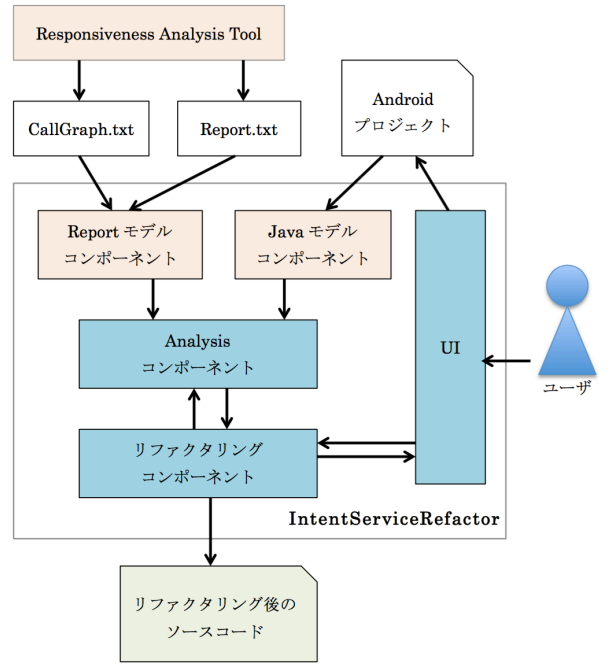


図 1: 提案手法のアーキテクチャ

ブロッキングメソッドの特定には Responsiveness Analysis Tool[4] を用いた．また，提案ツール（IntentServiceRefactor）は Asyncfactor[2] を改良することで実装を行った．なお，図 1 の Report モデルコンポーネント（Responsiveness Analysis Tool から出力されるファイルをまとめるコンポーネント）と Java モデルコンポーネント（Android プロジェクト内のコンパイル単位のソースコードをまとめるコンポーネント）は Asyncfactor で実装されているものを用いた．

次に提案ツールの実装について述べる．図 1 において新たに実装した箇所は，Analysis コンポーネント，リファクタリングコンポーネント，UI の 3 箇所である．これら 3 か所のうち，Analysis コンポーネントとリファクタリングコンポーネントについて述べる．

- Analysis コンポーネント
  リファクタリングの対象となりうるメソッドの特定を行う．リファクタリングの対象はブロッキングメソッド自体ではなく，そのブロッキングメソッドを呼び出し，かつアクティビティ上で実行されるメソッドが対象となる．その理由は対象メソッドの実行箇所から IntentService を呼び出すためである（IntentService の呼び出しはアクティビティ上で行う）．

- リファクタリングコンポーネント
  リファクタリング対象メソッドに対し，実際にリファクタリングを行う．次の 2 つの処理を行う．
  - IntentService をスーパークラスとして持つ Java クラスを新規作成．
  - 対象メソッドを実行している箇所を IntentService を呼び出す記述で置換．

リファクタリング前後のコード例を図 2 に示す．

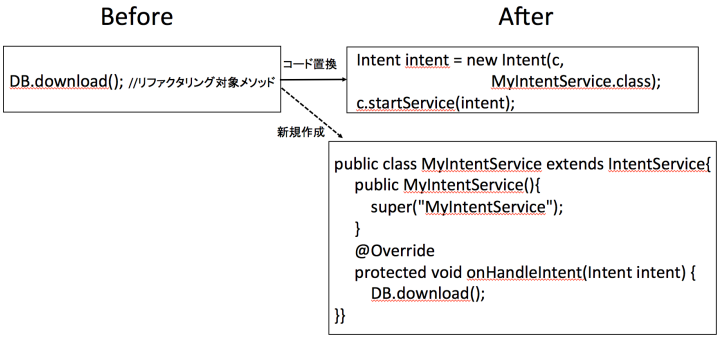


図 2: リファクタリング前後のコード例

## 5 結論

ブロッキングメソッドをアプリコード中から特定し，IntentService を用いた非同期処理へと直接リファクタリングを行うツールを提案した．

今後の課題としては，ブロッキングメソッドの種類に応じて AsyncTask と IntentService のどちらへもリファクタリングを可能にし，一度のツール実行でより多くのパフォーマンスバグを取り除くことが挙げられる．

## 参考文献

[1] Yu Lin, Semih Okur, and Danny Dig. Study and Refactoring of Android Asynchronous Programming. In *ASE 2015*, pp. 1–12, 2015.

[2] Jose Lorenzo San Miguel. Asyncfactor: Asynchronous Refactoring Support for Android Applications. pp. 1–4, 2015. White Paper, Keio University.

[3] Yang Shengqian, Yan Dacong, and Rountev Atanas. Testing for Poor Responsiveness in Android Applications. In *MOBS 2013*, pp. 1–6, 2013.

[4] Ongkosit Thanaporn and Shingo Takada. Responsiveness Analysis Tool for Android Application. In *DeMobile 2014*, pp. 1–4, 2014.

[5] Liu Yepang, Xu Chang, and Cheung Shing-Chi. Characterizing and Detecting Performance Bugs for Smartphone Applications. In *ICSE 2014*, pp. 1013–1024, 2014.