非推奨となった機能を利用するアプリケーションにおける対応策の実施状況の調査

2012SE090 加藤雅也　2012SE166 森川敬太
指導教員：横森励士

1　はじめに

近年のソフトウェア開発では，第三者によって提供されたライブラリやフレームワークを利用することが必要不可欠である．提供されたライブラリなどで不具合が生じた場合，提供者は更新を行い問題を解消する．この時，ライブラリ利用者は更新内容に沿って何らかの対応を迫られることがあるが，対応を行う利用者が少ないことを過去の研究 [1]，[2] が明らかにしている．不具合を放置することはセキュリティ上の問題を引き起こしたり，意図しない別の不具合を誘発する可能性につながると考えられる．

本研究では，JDK を利用した Java アプリケーションを分析し，ライブラリの更新によって非推奨となった機能が，実際にどれくらい利用されていたのか，後のアプリケーションの更新によって対応がなされているのか，それらがライブラリ側が推奨している対応にのっとったものであるかを調査する．このことから Java における非推奨の仕組みが，ライブラリの提供側の視点から更新を促す仕組みとして有効に作用しているのかを調査し，更新を促す仕組みにおいてさらに必要な事があるかを考察する．

2　関連研究

2.1　ライブラリと API

近年のソフトウェアでは，すでに実現されている要素や機能，ソフトウェアの設計方針をライブラリやフレームワークとしてソフトウェアに組み込んで，目的とする機能を実現している．ライブラリ側で提供している機能を実行するために必要なデータや記述方法を定めたものを API(アプリケーションプログラムインタフェース) と呼ぶ．ライブラリを利用する側は，ライブラリ側で提供されている機能群を API を介して実行する．ライブラリ側が更新される時に新たに機能が追加されるときはその機能に対応した API が追加される．

フレームワークの進化により提供される機能が増え，API が増加する一方で，すでに公開されている API を削除することは非常に難しい．理由としては，その機能を提供する API を削除することで，今までその機能を利用していたアプリケーションに大きな影響を与えるからである．幅広く利用されているアプリケーションほど機能の削除を慎重に行わなければならない．

2.2　Java における非推奨の仕組みについて

Java において提供されている，非推奨 (deprecated) と呼ばれる機能は，対象となるライブラリを利用しているユーザに向けて特定のクラスやメソッド等の使用を控えるように警告を行う機能である．ある一定の期間非推奨の段階を保ったのちに，ライブラリが機能の削除を行うことで，ライブラリ利用者側が受ける影響を最小限にすることができると考えられる．JavaSE8 現在，非推奨であることを伝える仕組みは 2 種類存在し，これらの記述を合わせて書き，仕様書やコンパイラ上で非推奨であることを示す方法が一般的な利用方法である．

1. Javadoc タグ
   JDK1.1 から利用されている機能であり，Java のソースコードから HTML 形式の API 仕様書を生成する，Javadoc とよばれるソフトウェアにおいて用いられるタグである．クラス，メソッドなどの宣言の前にコメントを配置し，コメント中で@deprecated タグを記述することで，後述するメソッドやクラスが非推奨であることを Javadoc 内で示す．非推奨であること，参考とするリンクや対応方法，代わりとなるメソッドをあわせて記述することが求められている．図 1 は実際の表記例で，将来的に一般利用される予定がなく，このメソッドはパッケージ private メソッドとしてのみ保持される予定である，ということを示している．また，@link タグによって API 仕様書内で該当メソッドの代替部品として remove(int)，removeAll() を提示し，API 仕様書へのリンクを生成している．
2. 注釈 (アノテーション)
   J2SE5.0 から導入された機能で，クラス，メソッドなどの宣言の前に@Deprecated というアノテーションを記載することで実現する．注釈を記述することで，ソースコード内で非推奨なクラス，メソッドなどが利用されている時に，コンパイラで警告表示を行う．図 1 では，delItems(int,int) メソッドが非推奨であることを示している．

2.3　JDK における非推奨部品の変遷について

JDK とは，Java でソフトウェア開発を行うのに必要なソフトウェアを 1 つにまとめたパッケージである．JDK においても，前述の機能を用いて非推奨となるクラスやメソッドが定義されている．表 1 は，JDK の各バージョンにおいて非推奨とされたクラス，インタフェース，メソッドの数の推移を示したものである．表からはバージョンを追うごとに増加する傾向が確認でき，アプリケーションの更新に合わせて非推奨となった部品への対応が求められていることがわかる．

```
/**
* Delete multiple items from the list.
*
* @deprecated Not for public use in the future.
* This method is expected to be retained
* only as a package private method.
* {@link #remove(int)}
* {@link #removeAll()}
*/
@Deprecated public synchronized void delItems(int start, int end) {...}
```

図 1 Javadoc タグや注釈の表記例

表 1 非推奨と指定された機能の数

バージョン	リリース日	クラス	インタフェース	メソッド
JDK1.1	1997/ 2/19	2	0	132
J2SE1.2	1998/12/ 8	6	4	238
J2SE1.3	2000/ 5/ 8	4	4	260
J2SE1.4	2002/ 2/ 6	5	3	278
J2SE5.0	2004/ 9/30	21	16	318
JavaSE6	2006/12/11	21	17	334
JavaSE7	2011/ 7/28	21	17	343
JavaSE8	2014/ 3/18	23	18	358

2.4 ライブラリの更新時のアプリケーション側の対応について

サードパーティ製のソースコードがどのような影響を与えるのかを分析した研究は Benjamin らによる長寿命なソフトウェアシステムコンポーネントの影響の調査が行われた研究 [3] をはじめ数多く存在している．しかし，サードパーティ製のソースコードが更新された際の利用者の対応を分析した研究は多くない．Xia の研究 [1] では，zlib や libpng などのライブラリで提供されているソースコードを題材とし，それらがどのようにコピーペーストされ利用されているか，ライブラリ側の更新があった場合どう対応しているかを調査している．その結果，130 個程度のオープンソースプロジェクトがコピーペーストで利用されており，zlib で 3 割，libpng で 9 割が問題を抱えたままのライブラリのバージョンを利用していることが判明した．全体の 8 割がこういった問題を抱えている一方，残りの 2 割のプロジェクトはライブラリの問題に対して修正を行っていた．しかし，ライブラリ提供者の意図通りの修正が行われていたのはその 2 割のプロジェクトの 75% ほどであり，残りは意図に沿わない修正となっていたことがわかった．梶田・高井・高須らの研究 [2] では，SourceForge で公開されている TwitterAPI を利用したアプリケーションを対象とし，TwitterAPI1.0 が廃止されバージョン 1.1 へ移行した際にそれらのアプリケーションがどのような対応を行ったかについて調査が行われている．この研究では，TwitterAPI1.0 の廃止報告後に，保守もしくは保守後の休眠状態にあった 104 個のプロジェクトを分析しており，その内 6 個のプロジェクトにおいて移行に必要な対策が施されていたことがわかった．

3 調査内容

3.1 研究の動機

過去の研究 [1]，[2] では，ライブラリや API のバージョンが変更された際の新しいバージョンへの対応状況を調査したが，対応していたプロジェクトの割合は 6〜20% などと低いものであった．それらと比較して，非推奨の仕組みが提供されサポート体制が整った Java の JDK はライブラリ提供者が出来る限り可能な支援を行っていると考えられる．現状想定できる環境下で，変更が促されている部品への対応が最も高い割合であらわれるのではないかと考え，非推奨となる利用を認めながら変更を促すというアプローチで，移行がどれだけ進むのかの評価を行う．

3.2 調査の手順

API 仕様書から JDK で非推奨とされる表 1 のクラス，インタフェース，メソッドの一覧を取得し，それらを非推奨となった機能群と定義する．以下の手順で，非推奨となった機能群の利用状況を調査する．

1. SourceForge で公開されているオープンソース開発プロジェクトのソースコードを取得する．
2. 上記のプロジェクトに対して，非推奨に指定された機能が利用されているかを調査し，非推奨となった機能を利用しているプロジェクトを抽出する．
3. 2 で抽出したプロジェクトそれぞれに対し，後続バージョンで非推奨となった機能が継続して利用されているか，修正が行われ使われなくなったかを調査する．
4. 修正が確認できたプロジェクトに対し，その修正方法を確認する．ライブラリ提供者が提供した修正方法であるか，プロジェクトの開発履歴中に修正項目として説明があるかを調査する．

3.3 調査項目

非推奨となるクラス，メソッドなどがどれだけ利用されていたか，修正されたかを調査するために，以下のリサーチクエスチョンを設定して，それらの割合を調べる．

Q1 非推奨に指定される機能を利用しているプロジェクトの数はどれぐらいか

- どれぐらいの数のプロジェクトで非推奨に指定される機能が利用されていたかを把握する．これにより，非推奨となるような機能は実際のプロジェクトで利用されているものなのかを調べる．

Q2 非推奨な機能はどのようなものが利用されていたか

- Q1 で利用されていた機能はどのようなものがあったかを把握する．これにより，比較的利用されていた非推奨となった機能を知ることができる．

Q3　上記の機能が非推奨になった後に，対応を行ったプロジェクトはいくつあるか

- それらのプロジェクトで非推奨となった機能の利用がほかの代替方法に置き換わっているかを把握する．これにより，利用者はライブラリの更新にどのぐらい関心を持ち，対応を実施しているのかを調べる．

Q4　各プロジェクトの対応はどのような手順でなされたか

- 利用している機能が非推奨になった際，利用者はどのような手順で対応を行ったのかを把握し，現在の非推奨の仕組みで対応策が正しく伝わったかを調べる．

Q5　該当プロジェクトのリリースノートもしくはそれに準ずるものに対応を行ったことが記述されているか

- 特定の機能が非推奨に指定された後に，開発者が対応すべき事柄として認識していたかを調べる．

Q6　対応方法は Javadoc の記述に沿った対応であるか

- 提供者の意図した通りの対応を行ったプロジェクトはどれぐらい存在するのかを把握する．これにより，現在のような注釈，Javadoc の表記方法で利用者に正しい対応例が伝わっているのかが分かる．

4　調査結果

SourceForge でソースコードが公開されているオープンソース開発プロジェクト 73 個のソースコードを入手し，調査を行った．各調査項目に基づいて結果を記述する．

Q1　非推奨に指定される機能は利用されていたか

Q2　非推奨な機能はどのようなものが利用されていたか

A　73 個のプロジェクトのうち 11 個のプロジェクトにおいて非推奨となったクラスやメソッドが利用されていた．それらのプロジェクトの一覧と各プロジェクトで利用されていた非推奨な機能の数をまとめたものを表 2 に示す．利用されている機能を分類したところ，3 つの非推奨クラスと 5 クラスにわたる 11 種類の非推奨メソッドが確認された．それらの一覧を表 3 に示す．表 3 では，利用されていた非推奨クラス，利用されていた非推奨メソッドおよびその所属クラスと，それを利用していたプロジェクトの一覧を示している．

Q3　上記の機能が非推奨になった後に，対応を行ったプロジェクトはいくつあるか

Q4　各プロジェクトの対応はどのような手順でなされたか

A　11 個のプロジェクトでその後の対応状況を調査したところ，そのうち 4 個のプロジェクト，36% においてその後のバージョンでの対応が確認された．Joda-Time では，非推奨に指定された際の代替メソッドと非推奨なままのメソッドの 2 つが同一ファイルで併せて記述されており，非推奨となったメソッドの利用を回避する事ができることを確認できた．Luaj では，非推奨に指定されたクラスの部分が同様の働きをするクラスに置き換えられているのが確認できた．MegaMek では，非推奨に指定されたメソッドの部分が提示されている代替方法に置き換えられていることが確認できた．Jipe では，取得可能な最初期のバージョンのソースコード中に過去のバージョンで非推奨なメソッドが使われていたという表記が確認できた．そして，ソースコードが取得可能なバージョン以降ではそのメソッドに対応した部分は提示されている代替方法に置き換えられていることが確認できた．

Q5　該当プロジェクトのリリースノートもしくはそれに準ずるものに対応を行ったことが記述されているか

A　非推奨に指定された機能への対応が記述されているかについては，対応の行われた 4 つのプロジェクトのリリースノートを確認した．Joda-Time では非推奨について言及する記述はなかったものの，ver0.99 から Date クラスと Calendar クラスの 2 つを利用する旨の記述が確認できた．MegaMek では開発履歴上で警告が出ていた部分を一括で修正したという記述が ver0.34.0 にあり，その 1 つとして非推奨部分の修正が行われていた．Jipe では ver0.90c のソースコード中のコメントに過去に非推奨メソッドを利用し，現在は修正したという記述が確認できた．Luaj では記述は確認できなかった．

Q6　対応方法は Javadoc の記述に沿った対応であるか

A　対応が確認できた 4 つのプロジェクトでは，いずれも Javadoc に記述された代替方法に則った修正が行われており，提供者の意図した通りの対応を行っていた．

表 2　非推奨な機能を利用していたプロジェクトの非推奨機能の個数

プロジェクト名	ver	リリース日	クラス	インタフェース	メソッド
Areca Backup	5.0	2007/5/26	0	0	3(6)
Joda-Time	0.9	2003/12/16	0	0	4(7)
Pebble	1.0	2003/6/3	0	0	2(15)
Luaj	1.0	2009/6/20	1(1)	0	0
Drjava	-	2002/6/19	0	0	1(4)
FileBot	1.96.473	2011/7/14	0	0	1(5)
SQuirreL	1.0final0	2001/10/16	0	0	2(5)
FindBugs	1.2.1	2007/5/31	2(2)	0	2(3)
TV-Browser	2.2.6	2009/8/6	0	0	6(16)
MegaMek	0.30.0	2005/9/17	0	0	1(1)
Jipe	0.90c	2000/4/30	0	0	1(1)

5　考察

今回の研究では，73 個のプロジェクトを分析対象として分析を行った結果，11 個のプロジェクトで非推奨な機能が利用されており，その中の 4 個のプロジェクト，36% でライブラリ提供者の意図した対応が行われていたという結果を得られた．これは Xia の zlib，libpng を利用したプロジェクトに対する調査 [1] における修正があった割合であ

表 3　利用されていた非推奨となる機能と利用プロジェクトの一覧

クラス	
クラス名	利用プロジェクト
java.io.LineNumberInputStream	Luaj
java.io.StringBufferInputStream	FindBugs
java.rmi.server.RemoteStub	FindBugs
メソッド	
クラス名-メソッド名	利用プロジェクト
java.util.Date	
-getHours()	Areca Backup Joda-Time
-getMinutes()	Joda-Time
-getMonth()	Areca Backup
-getSeconds()	Areca Backup Joda-Time
-getYear()	Joda-Time
-parse(String)	Pebble
-setDate(int)	Pebble
java.io.File	
-toURL()	SQuirreL Drjava MegaMek
javax.swing.JList	
-getSelectedValues()	FileBot FindBugs
java.awt.datatransfer.DataFlavor	
-equals(String)	TV-Browser
java.awt.Toolkit	
-getFontList()	Jipe

る 20%，梶田らの TwitterAPI を利用したプロジェクトに対する調査 [2] における，TwitterAPI1.1 へ移行した割合である 5% と比較すると，高い対応率であったと考えられる．また，それらの修正方法のいずれも Javadoc の記述に沿った対応であることを考えると，現状提供されている仕組みはうまく働いていると考えられる．その反面，プログラムが動かないなどの問題が発生するまでは対処してもらうのが難しいというアプローチ自体の限界も考えられる．JDK のライブラリ更新時の対応率はほぼ限界値であったと推測するが，さらなる対応率向上のために変更の事例を具体的に紹介することで，より対応がなされるのではないかと考えられる．例えば，表 3 で表されたクラスごとに利用されていた非推奨機能の一覧より，java.util.Date クラスと java.io.File クラスのメソッドが 3 つのプロジェクトで利用されていることが明らかになっており，高頻度で利用されているであろう非推奨な機能が存在していることが推測できた．この 2 つのクラスに属したメソッドは提供者の意図した対応が行われており，高頻度で利用されている機能ほど非推奨機能を利用しなくするための対応策が分かりやすく，ライブラリ利用者にとって対応が行いやすくなっている可能性が推測できた．実際に確認したところ，該当の非推奨機能の対応策には代替となるメソッドがはっきりと明示されているため非常に対応しやすいと考えられる．逆に言えば，対応された実績がない機能ほどライブラリ利用者が対応を行うことが難しいと言える．そのため，対応策に基づく変更の事例を具体的に紹介するなど分かりやすく表記することで，非推奨な機能の利用は減るのではないかと考えられる．

5.1　非推奨機能の削除

2017 年 1 月現在利用されている JavaSE8 では，非推奨な機能をソースコード中に記述してもコンパイラの警告等を行うだけであり，機能自体の利用にこれといって制限は存在していない．しかし，2017 年 7 月に一般提供が予定されている JavaSE9 では機能自体の利用が行えなくなる可能性が予告されている．これまで数多くの機能が使用を控えるようにしてほしいなどという理由から非推奨に指定されてきたが，機能自体は 1 度も削除されたことはなかった．該当の機能を削除して利用できなくしてしまうと，それらを利用していたアプリケーションの一部もしくは全てが動作しなくなる恐れがあり，それを回避するための方策がとられてきたからである．現在削除を予定していると公表されているのは 6 つのメソッドのみであるが，今回のバージョンを皮切りとして後続のバージョンで他の機能も削除される可能性が考えられ，後に同じ調査を行った場合，アプローチが変わることで今回よりも高い対応率であったという結果が得られる可能性がある．

6　まとめ

本研究では将来的に非推奨に指定される機能を利用したプロジェクトを分析することによって，JDK の更新時に提供者の意図通りの対応を利用者が行うことができるのか，また更新を促す仕組みが有効であるかを分析した．調査の結果からは，現状提供されている仕組みは有効に働いていると考えられる．ただし，変更の事例を具体的に紹介するなどのサポート方法を行う余地があり，過去の機能が提供されなくなった場合には，移行を促すための仕組みを十分に用意することが求められると考えられる．

参考文献

[1] Pei Xia, Makoto Matsushita, Norihiro Yoshida, Katsuro Inoue: ”Studying Reuse of Out-dated Third-party Code in Open Source Projects”, Computer Software, Vol.30, No.4, pp.98-104, 2013

[2] 梶田祐紀，高井亮輔，高須健 :“TwitterAPI1.0 の廃止によるアプリケーションへの影響の分析,”南山大学情報理工学部 2014 年度卒業論文，2015.

[3] Benjamin Klatt, Zoya Durdik, Klaus Krogmann, Heiko Koziolek, Johannes Stammel, and Roland Weiss: ”Identify Impacts of Evolving Third Party Components on Long-Living Software Systems”, In Proceedings of the 16th Conference on Software Maintenance and Reengineering (CSMR), pp.461-464, 2012.