# ファイナライザを理解する

## ～ファイナライザに起因するトラブルを避けるために～

2013 年 11 月 25 日

橋口　雅史

Java アプリケーションでファイナライザ（finalize()メソッド）を使用したことがあるプログラマーは多いと思います。しかし、ファイナライザの仕組みや注意点について、理解したうえで使っているでしょうか？

アプリケーション・プログラムでファイナライザを使用する場合は、ガーベジ・コレクションの挙動などについて理解しておく必要があります。そのため、ガーベジ・コレクションの仕組みと併せて、ファイナライザについて解説します。

## 1. はじめに

Java には、ファイナライザという仕組みが定義されています。ファイナライザは、簡単にプログラムに組み込めます。一方で、ファイナライザを間違って使うとトラブルを引き起こします。

本書は、ファイナライザの仕組みについて説明しています。しかし、ファイナライザの使用を推奨するものではありません。

ファイナライザは使い方が非常に難しいため、基本的にはファイナライザを使用しないでください。ファイナライザの使用を検討する前に、目的を実現可能なほかの方法を検討してください。どうしてもファイナライザを使う必要がある場合は、本書の内容を十分に理解した上で、必要最小限の範囲で使用してください。

### 本書の対象読者

本書は、Java 言語、Java API などの仕様や、Java に関する知識を持っている読者を対象にしています。本書では、Java に関する基本的な用語などについて、詳しく説明しません。用語などに関する情報は、それぞれの仕様書や一般書籍などを参照してください。

## 2. ファイナライザにおけるよくある誤解

ファイナライザの仕組みについては3章以降で詳しく説明しますが、ファイナライザに関してよく誤解されることを紹介します。これらの誤解を解くことも、本書の目的のひとつです。

### ファイナライザ実行のタイミング

ファイナライザを持つオブジェクトは、それがゴミになった時点で、すぐにファイナライザが実行されると誤解されることがあります。実際にはそのようなことはありません。ファイナライザの実行タイミングをアプリケーションでは制御できません。

ガーベジ・コレクションとファイナライザの関係

ファイナライザを持つオブジェクトは、そのオブジェクトが到達不能になったあと、ガーベジ・コレクション（GC）処理が 2 回実行されれば、オブジェクトのファイナライザが必ず実行され、そのメモリが回収されると誤解されることがあります。

世代別 GC を実装している Java VM においては、ファイナライザを持つオブジェクトが Old 領域に存在することがあります。Old 領域に存在するオブジェクトは、ファイナライザ到達可能になっても、フル GC が実行されるまで、ファイナライズ可能になりません。そのため、New 領域に対する GC が何度実行されても、オブジェクトはファイナライズ可能になりません。つまり、ファイナライザは実行されません。

また、GC 処理の最中、ファイナライザは処理されません。そのため、ファイナライズ待ちリストに大量のオブジェクトが存在する状態で GC が頻発した場合、いつまでもファイナライザが処理されないこともあり得ます。

## 3. ガーベジ・コレクション

ファイナライザは、ガーベジ・コレクション（GC）と密接な関係にあります。そのため、まず GC についての基本的な特徴を説明します。続いて、富士通のアプリケーションサーバーに採用している Java VM の GC 方式（世代別 GC）について説明します。

ガーベジ・コレクションの基本的な特徴

Java のメモリ管理では、GC が重要な役割を果たします。Java VM がオブジェクトを管理するメモリ領域（Java ヒープ領域）がいっぱいになると、Java VM は、GC 処理を実行します。GC 処理では、不要になったオブジェクトのメモリを開放し、ほかのオブジェクトによってメモリを使用できるようにします。

Java VM は、オブジェクトが必要かどうかを、オブジェクトに対する参照が存在するかどうかで判断します。つまり、Java アプリケーションによって使用されているクラスのクラス変数や、スタック上の変数から、参照をたどって到達できるオブジェクトは、その時点で必要であると言えます 。逆に、参照関係が途切れている場合、そこから先のオブジェクトには到達できないので、不要になったものと言えます（不要になったオブジェクトのことを「ゴミ」と言うこともあります）。

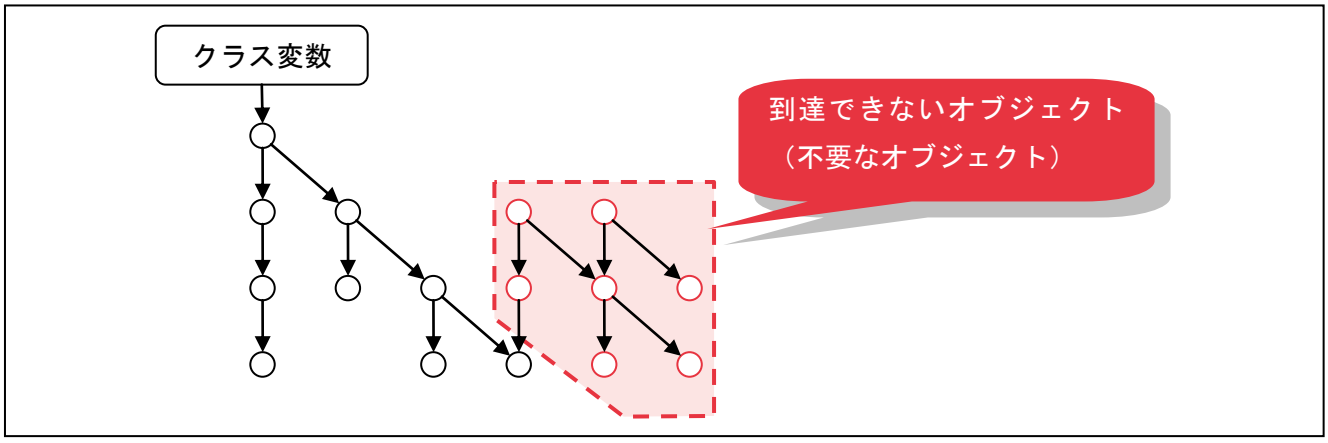


**図 1 クラス変数から到達できないオブジェクトの例**

世代別ガーベジ・コレクション

富士通のアプリケーションサーバーに採用している Java VM は、Java ヒープを世代で管理して、GC 処理の効率化を図っています。

Java ヒープは、New 世代、Old 世代、および Permanent 世代のみっつの領域に分かれています。Permanent 世代領域は、ロードしたクラスの情報などが格納される領域です。ファイナライザの説明とは関係しないため、ここでは説明を省略します。

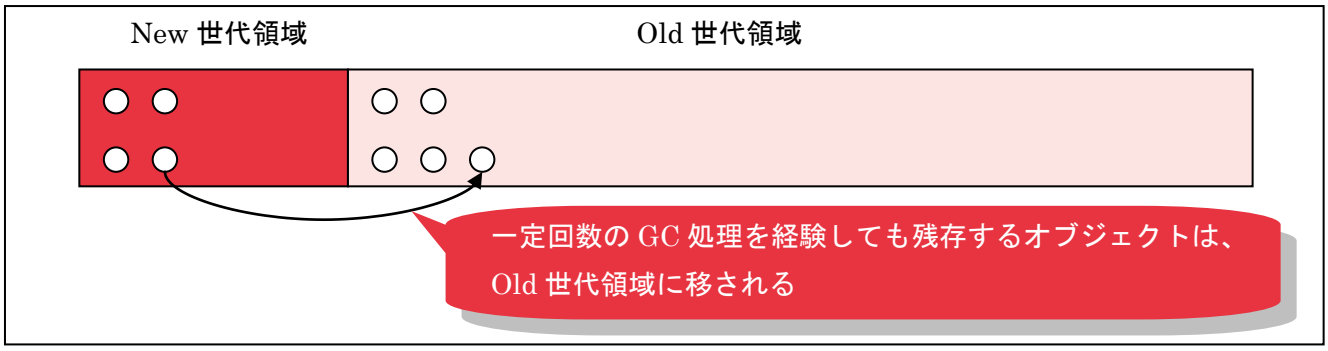


**図 2 世代別ガーベジ・コレクション**

Java アプリケーションが生成を要求したオブジェクトのメモリは、通常、New 世代領域に割り当てられます。New 世代領域がいっぱいになるまでオブジェクトが生成されると、Java VM は、GC 処理を実行します。GC 処理が終了すると、参照が存在するオブジェクトだけが New 世代領域に残ります。

GC 処理が一定回数繰り返されたあとも New 世代領域に残っているオブジェクトは、GC 処理によって Old 世代領域に移動されます。

Old 世代領域のメモリは、Old 世代領域がいっぱいになるまでガーベジ・コレクトされません。この特徴は、ファイナライザの挙動にも影響してきます。

## 4. ファイナライザ

ファイナライザとは、finalize()メソッドのことです。finalize()メソッドは、Java 言語におけるすべてのクラスの継承元である java.lang.Object クラスに定義されています。すなわち、すべてのクラスは finalize()メソッドを持ちます。しかし、java.lang.Object クラスの finalize メソッドは何もしません。finalize メソッドは、継承先のクラスでオーバーライドされたときに意味を持ちます。

ファイナライザは、C++言語のデストラクタと同じものだと考えられることがあります。しかし、C++言語のデストラクタと Java 言語のファイナライザはまったく異なります。

C++言語のデストラクタは、スタック上に作成されたオブジェクトが自動的に破棄される場合や、メモリ上に作成されたオブジェクトが delete 演算子によって明示的に破棄される場合に呼び出されます。つまり、デストラクタを呼び出すタイミングをアプリケーションで制御可能です。

Java 言語のファイナライザは、オブジェクトのライフサイクル、すなわち GC 処理と密接に関わっています。GC 処理は、アプリケーションでは制御できません。そのため、ファイナライザを呼び出すタイミ

ングも、アプリケーションでは制御できません。

ファイナライザについて理解するには、オブジェクトのライフサイクルを理解する必要があります。そのため、ファイナライザが実行される仕組みを説明する前に、Java VM がファイナライザを持たないオブジェクトをどのように制御するか説明します。次に、Java VM がファイナライザを持つオブジェクトをどのように制御するか説明します。

ファイナライザを持たないオブジェクトのライフサイクル

本書では、ファイナライザを持たないオブジェクトのことを、便宜的に「通常のオブジェクト」とします。通常のオブジェクトには、オブジェクトへの参照が存在する状態（到達可能）と、オブジェクトへの参照が存在しない状態（到達不能）が存在します。

**表 1 通常のオブジェクトの状態一覧**

| オブジェクトの状態 |
| --- |
| 到達可能（reachable） |
| 到達不能（unreachable） |

Java アプリケーションによってオブジェクトの生成が要求されてから、そのメモリが回収されるまでの状態は、次のように遷移します。

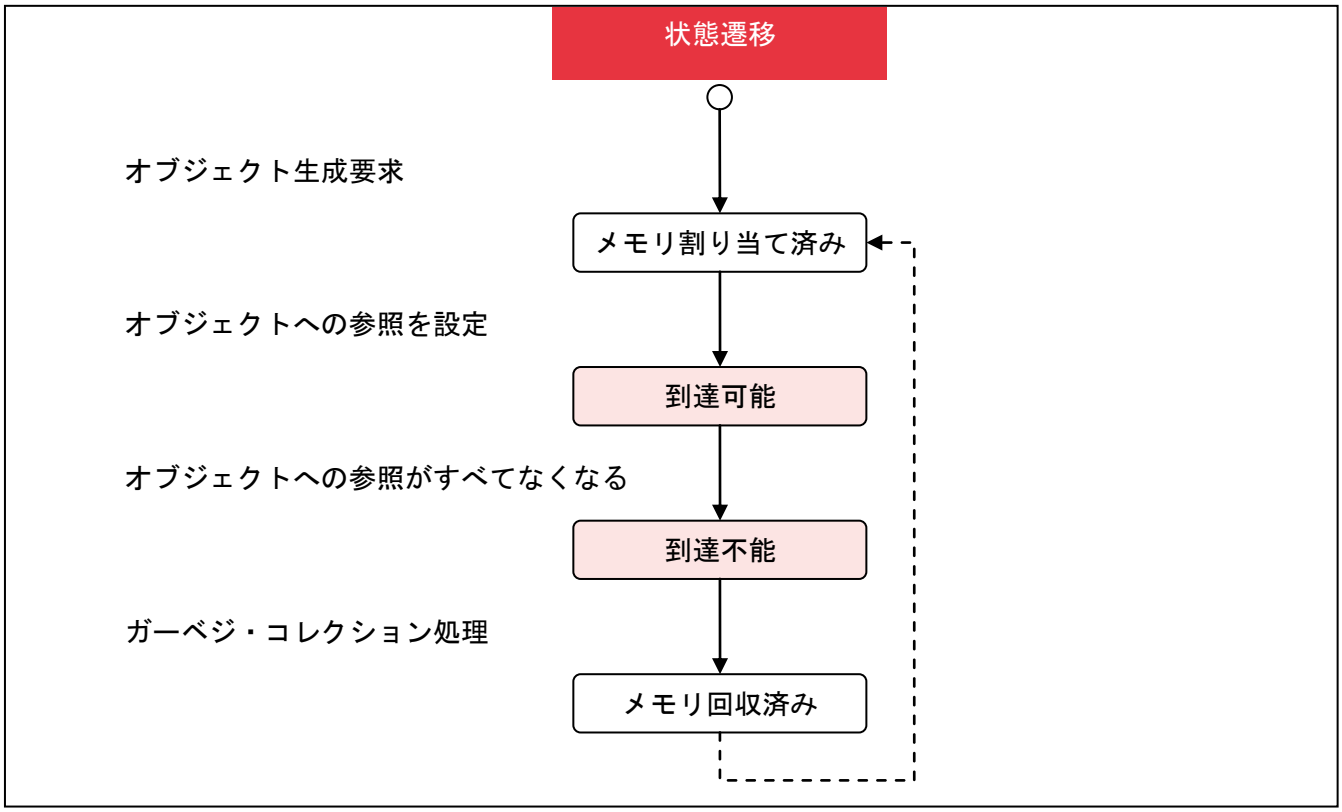


**図 3 通常のオブジェクトの状態遷移**

オブジェクトが到達不能な状態に遷移すると、それ以降の GC 処理によってそのオブジェクトのメモリは回収されます。回収されたメモリは、ほかのオブジェクトを生成するときのメモリとして再利用されま

す。

ファイナライザを持つオブジェクトの管理方法

ファイナライザを持つオブジェクトの管理方法は、通常のオブジェクトと異なります。一般的な Java VM では、ファイナライザを持つオブジェクトを特別なリストで管理します。このリストを便宜的に「ファイナライザ管理リスト」と呼びます。Java VM は、ファイナライザを持つオブジェクトのメモリを割り当てると同時に、ファイナライザ管理リストでも管理します。

ファイナライザ管理リストは、Java アプリケーションからは操作できません。リストの目的は、ファイナライザを呼び出すまで、Java ヒープ内にオブジェクトを留めておくことです。

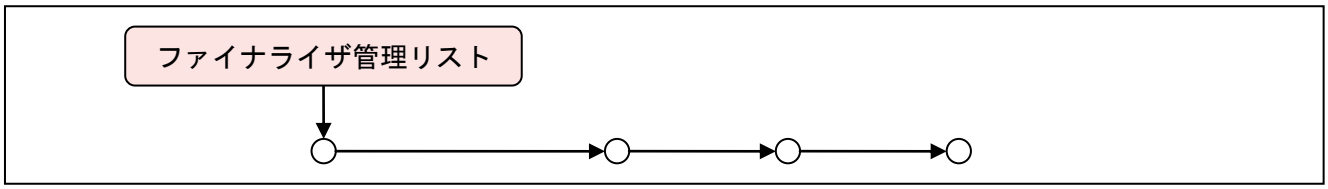


**図 4 ファイナライザ管理リストの概念図**

オブジェクトに到達可能な参照が、ファイナライザ管理リストだけになると、Java VM は、そのオブジェクトのメモリを回収するまえにファイナライザを実行します。ファイナライザを実行するため、Java VM は、オブジェクトをファイナライザ管理リストから別のリストにつなぎかえます。このリストのことを便宜的に「ファイナライズ待ちリスト」と呼びます。

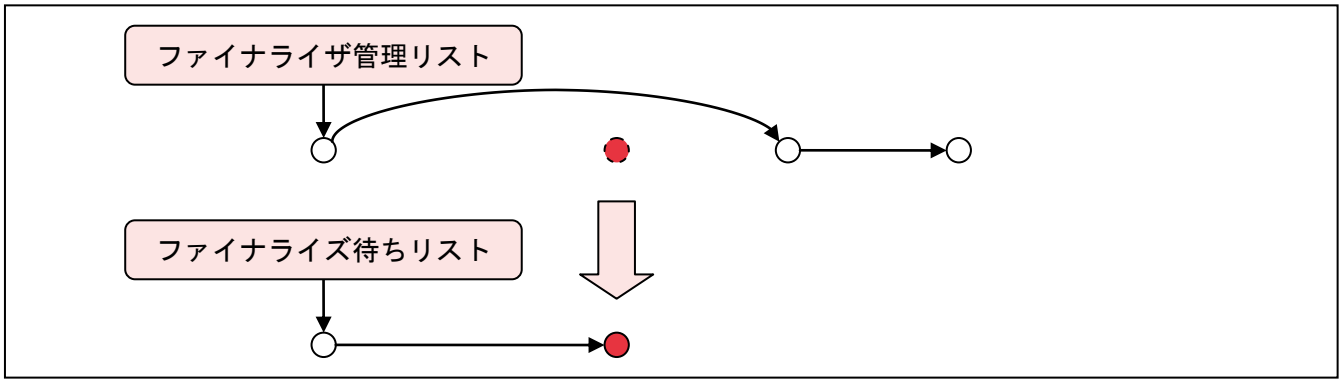


**図 5 ファイナライザ管理リストからファイナライズ待ちリストへの移動**

ファイナライズ待ちリストにつながれたオブジェクトは、Java VM によってファイナライザが実行されます。ファイナライザが実行されると、ファイナライズ待ちリストからの参照もなくなります。

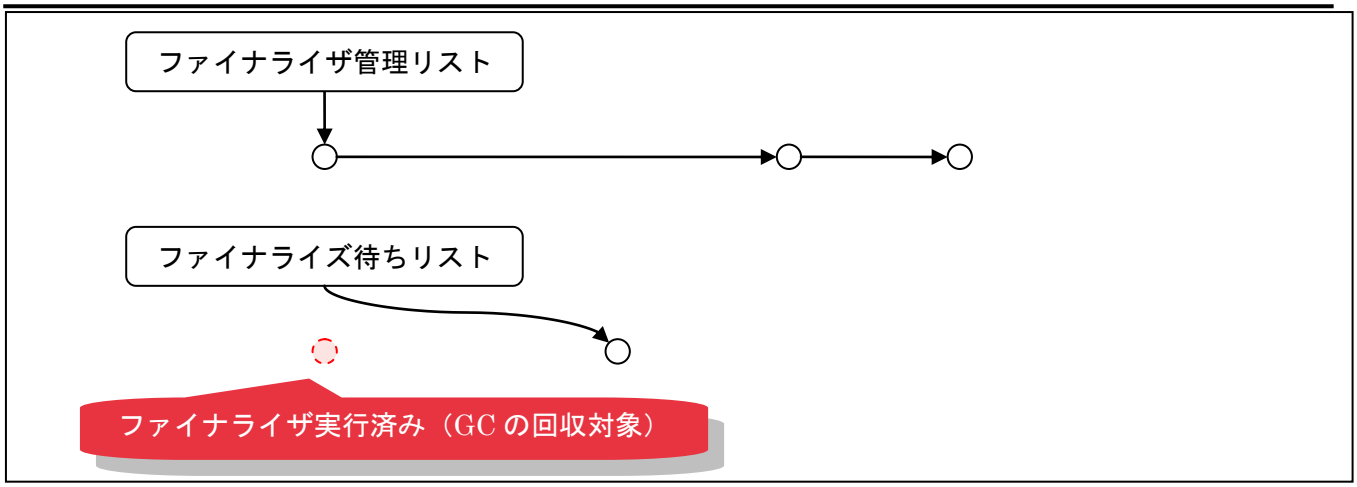


**図 6 ファイナライザ実行後のリストの状態**

このような状態になると、オブジェクトは GC の回収対象になります。

ファイナライザを持つオブジェクトのライフサイクル

ファイナライザを持つオブジェクトには、通常のオブジェクトに加えて「ファイナライザ到達可能」という状態が存在します。

**表 2 ファイナライザを持つオブジェクトの状態一覧**

| オブジェクトの状態 |
|---|
| 到達可能（reachable） |
| ファイナライザ到達可能（finalizer reachable） |
| 到達不能（unreachable） |

また、ファイナライズの状態という属性も考慮する必要があります。

**表 3 ファイナライズの状態一覧**

| ファイナライズの状態 |
|---|
| 未ファイナライズ（unfinalized） |
| ファイナライズ可能（finalizable） |
| ファイナライズ済（finalized） |

Java アプリケーションによってオブジェクトの生成が要求されてから、そのメモリが回収されるまでの状態は、次のように遷移します。

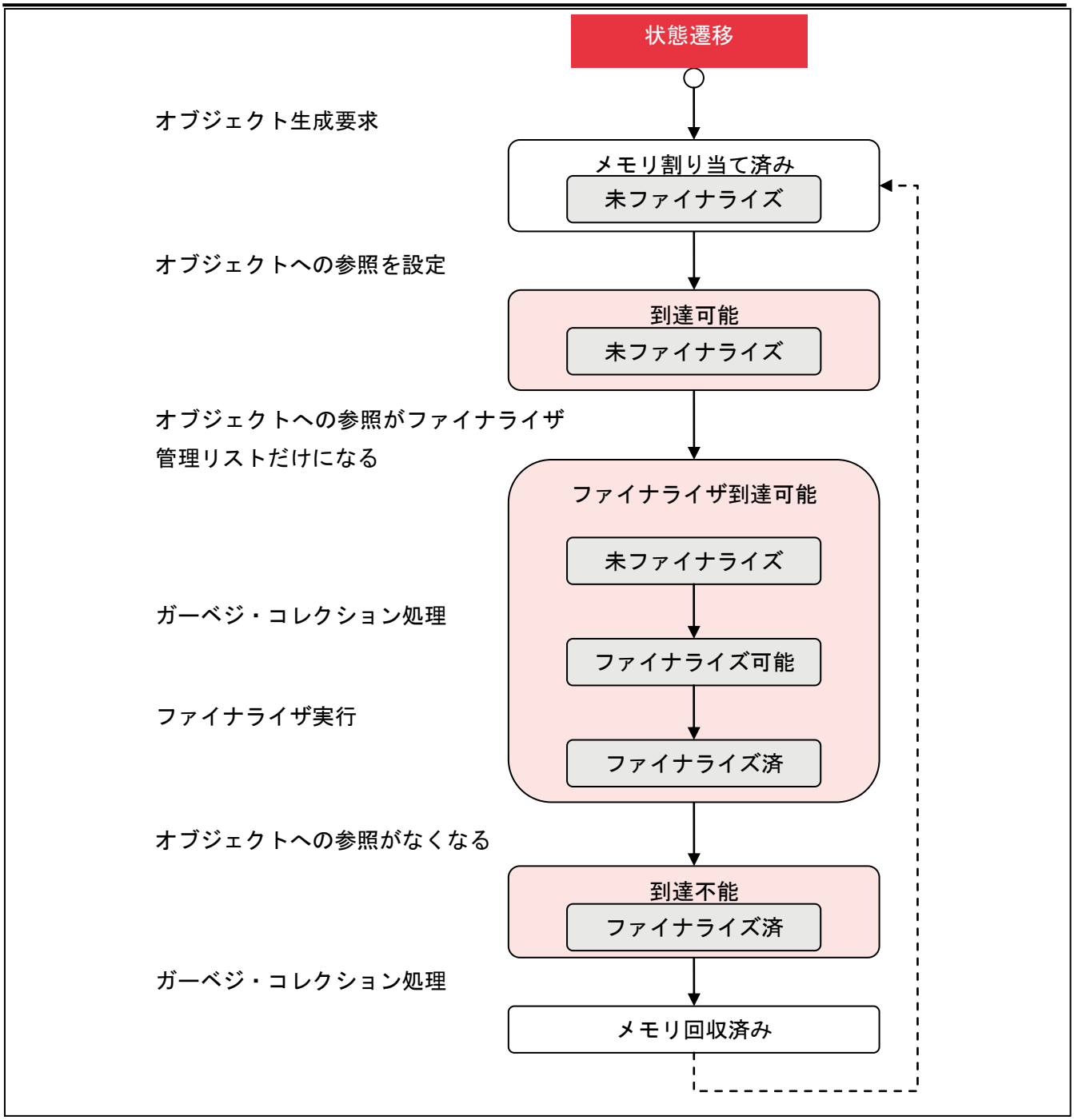


**図 7 ファイナライザを持つオブジェクトの状態遷移**

オブジェクトへの参照がファイナライザ管理リストだけになると、オブジェクトの状態は「ファイナライザ到達可能」に遷移します。「ファイナライザ到達可能」の状態に遷移したオブジェクトは、それ以降に実行される GC 処理によって、ファイナライズ待ちリストにつながります。それにより、ファイナライズの属性は「ファイナライズ可能」に変化します。

ファイナライズ待ちリストに繋がったオブジェクトは、Java VM によってファイナライザが実行されます。ファイナライザが呼び出されると、オブジェクトのファイナライズの属性は「ファイナライズ済」に変化します。そして、ファイナライズ待ちリストからの参照がなくなります。つまり、オブジェクトに対す

る参照がなくなります。
　オブジェクトへの参照がなくなると、状態は「到達不能」に遷移します。そして、それ以降に実行される GC 処理によって、オブジェクトのメモリが回収されます。

　このように、ファイナライザを持つオブジェクトのメモリが回収されるまでには、通常のオブジェクトよりも多くの状態遷移が必要です。また、アプリケーションで制御することができません。

## 5.　ファイナライザに関する注意事項

　ファイナライザに関する注意事項について説明します。

### ファイナライザの実行順序は不定

　ファイナライザは、アプリケーションが実行されるスレッドとは別のスレッドで処理されます。また、ファイナライザが実行される順序は不定です。アプリケーションではその順序を制御できません。その特徴により、特に同期処理で問題が発生する可能性があります。
　例えば、ファイナライザ以外の処理とファイナライザのあいだで同期を取ろうとすると、プログラマーが意図しない順序でファイナライザが実行された結果、デッドロックに陥ることがあります。
　ファイナライザとファイナライザ以外の処理のあいだでは、同期処理を行わないでください。

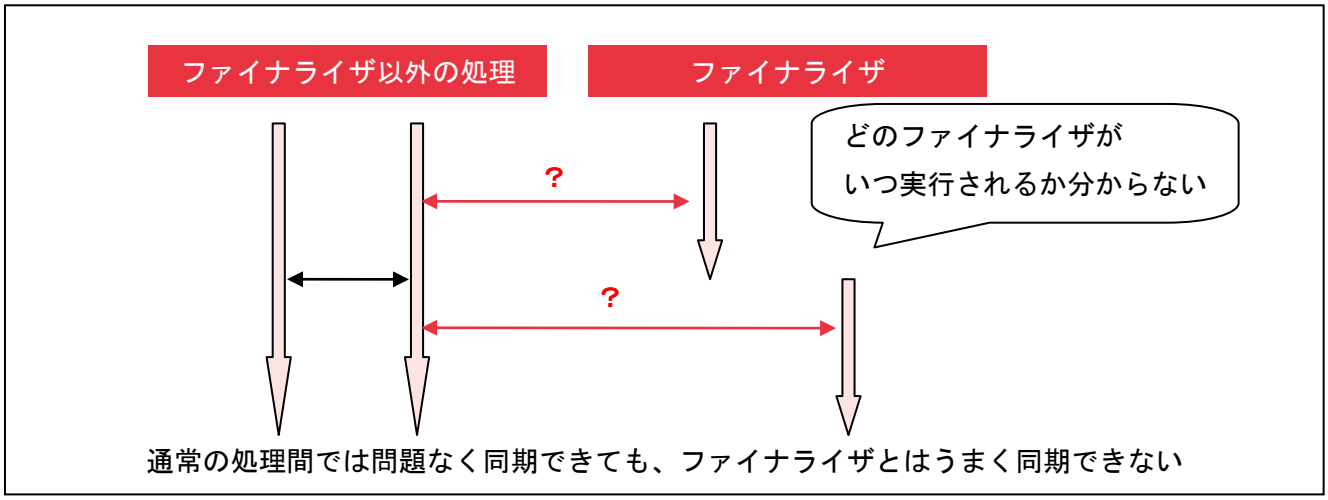


**図 8 ファイナライザの同期処理に関する問題**

### ファイナライザをリソース管理に使ってはいけない

　メモリなどのリソースをファイナライザで解放するようなプログラムは、リソースのリークを招く恐れがあります。
　ファイナライザの実行タイミングはアプリケーションで制御できません。また、Java VM がファイナライザをいつ実行するのかも分かりません。場合によっては、ファイナライザが実行されるまでに非常に長い時間がかかることがあります。そのような場合、リソースの解放をファイナライザに頼っていると、いつまでもリソースが解放されないことになります。
　オブジェクトのメモリがガーベジ・コレクタによって回収されたことを契機としてリソース解放などの処理を行いたい場合は、java.lang.ref パッケージを使用してください。

リソースの解放が必要なクラスには、解放用のメソッドを用意してください。そして、インスタンスの使用が終わったらそのメソッドを呼び出してください。ファイナライザは、そのメソッド呼び出しを忘れてしまったときでもリソースを解放する後始末の役目としてだけ用意してください。

java.io パッケージに含まれるクラスなどは、そのような実装になっています。それらのクラスのインスタンスに対しては、そのインスタンスの使用が終わったら原則として close()メソッドを呼び出してリソースを解放する必要があります。万が一 close()メソッドが呼び出されなかったときのために、ファイナライザでもリソースを解放します。

System.runFinalization()でファイナライザを確実に実行できるわけではない

Java には java.lang.System.runFinalization()というメソッドがあります。そのメソッドを呼び出せばファイナライザを実行できると思われることがありますが、そのようなことはありません。

runFinalization()メソッドは、先に説明した「ファイナライズ待ちリスト」に繋がっているオブジェクトのファイナライザを呼び出すものです。しかし、ファイナライズ待ちリストに繋がっていないオブジェクトに関しては何の効果もありません。

また、オブジェクトがファイナライズ待ちリストに繋がるタイミングは Java VM の処理に依存するため、runFinalization()メソッドをいくら実行してもファイナライザが実行されないことがあります。

ファイナライザの処理中に発生した例外は無視される

Java 言語仕様 で定義されているように、ファイナライザの処理中に発生した例外は無視されます。また、その時点でファイナライザの処理は終了します。

例外は Java 実行環境やアプリケーションの異常によって発生することがありますが、ファイナライザは発生した例外を無視します。そのため、なんらかの異常を残したまま、アプリケーションが動き続ける可能性があります。

また、例外の発生によりファイナライザの処理が中断されるため、期待した処理が完了せず、思わぬ問題を引き起こすことがあります。

ファイナライザに記述する処理は、例外が発生しないものか、例外が発生しても無視してよいものに限ってください。

親クラスがファイナライザを使っていることがある

プログラムではファイナライザを使っていないのに、ファイナライザが原因でトラブルが発生することがあります。それは、プログラムで定義したクラスの継承元でファイナライザを使用している場合です。

そのようなトラブルを未然に防ぐのは難しいですが、有効な方法のひとつは、クラスの設計時に不要な継承関係を作らないことです。

アプリケーションの設計時に、Java のコアライブラリなどに含まれるクラスを継承する必要性を十分に検討してください。すなわち、オブジェクト指向における「is-a」関係ではなく、「has-a」関係による目的

の実現を検討してください。
　また、インタフェースの実装をクラスの継承で代用しないでください（プログラミングが簡単になるという理由だけでクラスの継承を選択しないでください）。インタフェースの実装が必要な場合は、まずその実現を検討してください。

GC が頻発するとファイナライザの処理が進まない
　ファイナライザは、GC の最中は処理されません。GC が頻発するような状況になると、ファイナライザの処理が一向に進まなくなります。
ファイナライザによるリソースの解放を期待するようなプログラムでは、「GC によるファイナライザ処理の遅延」→「ファイナライザ処理の遅延による Java ヒープ領域の圧迫」→「GC の発生」→「GC によるファイナライザ処理の遅延」→…という悪循環に陥ることがあります。

## 6. 参考文献

The Java® Language Specification Java SE 7 Edition
http://docs.oracle.com/javase/specs/jls/se7/html/index.html