Java 概論

第 1.0 版

作　成　者	中村広二
作　成　日	2019 年 10 月 3 日
最終更新日	2019 年 10 月 3 日

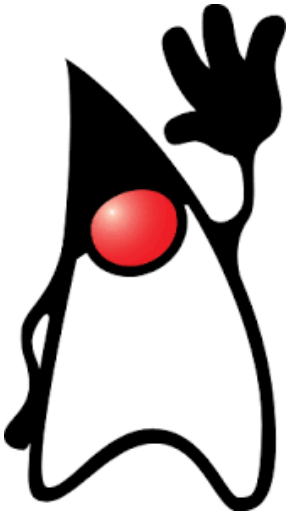
Java について

Java（ジャバ）は、狭義ではプログラミング言語であり、
広義では Java 言語を中心とした Java プラットホームを指す。
Java 関連技術を Java テクノロジと総称する。
1990 年代に，アメリカでワークステーションと呼ばれるコンピュータ市場を独占した会社があった。
Sun Microsystems だ。1991 年～1995 年にかけて、Sun Microsystems が、Green という名前のプロジェクトとして Java は開発した。

※科学計算や設計に用いられた高性能のコンピュータ「SPARC　station 2」
現在は Oracle のサーバ技術を支えている。富士通が製造という噂も・・・

※2010 年　Sun Microsystems は Oracle に買収された。UNIX ユーザーのあこがれの的だった Sun Microsystems のロゴ。

※Java のキャラクター「DUKE」。Sun Microsystems はイベントなどでこの DUKE のノベリティーを数多く作りプレゼントにした。オークションサイトでは今でも取引されている。



「Java ＝ オブジェクト指向」であり、オブジェクト指向を実装する言語と言い切っても、過言ではない。

Java は、従来のソフトウェアが抱えていた移植性問題の解決を図り、特定の環境に依存しない理想的なクロスプラットフォーム・プログラムの実現を目指して開発された。

このことを一言で
「Write once, run anywhere」　（一度書けば、どこでも実行できる）

「Write once, run anywhere」について書かれている英語の本
Java Everywhere: Write Once run Anywhere with DukeScript (English Edition)

Java ソフトウェアは、家電機器や乗用車の組み込みシステムから、マイクロ制御装置、携帯機器、パーソナルコンピュータ、サーバマシン、スマートカードといった様々な環境に普及している。
企業内の業務システムは、汎用機(COBOL など)のシステムから、Java で開発したシステムに置き換わっているケースが多い。

携帯電話・スマートフォンのような小さなものから銀行の大型コンピュータ、SONY 製の DVD のプレイヤーや Web システムまで、Java でつくられるようになっている。

３つの Java プラットフォーム

1999 年、Java のプラットフォームのための構成が，ソフトウエア開発業者，サービス提供業者，デバイス製造業者の３つのマーケットために再構築された。

Java SE (Java Platform, Standard Edition)

このプラットフォームを活用すれば、デスクトップとサーバだけでなく、今日要求される組み込みシステム上でも動作する Java アプリケーションが開発できる。
機能豊かなユーザー・インタフェース、パフォーマンス、一度プログラムを書けばどこでも動かせる汎用性、移植性に加えて、不可欠なセキュリティも提供している。

Java EE (Java Platform, Enterprise Edition)

世界中に支店を持つ銀行や商社のような大企業の情報システムを動かすための様々な機能を持った低リスクのプラットフォームを提供している。
2018 年 3 月 Oracle は Java EE に見切りを付け、今も高い人気を誇るこの Java のエンタープライズミドルウェアプラットフォームの Eclipse Foundation への移管に着手した。今回、Eclipse Foundation の管理下で、Java EE が「Jakarta EE」に改名されることが決定した。

Java ME (Java Platform, Micro Edition)

携帯電話、携帯情報端末 (PDA)、TV セットトップ・ボックス、プリンタなどの組込みデバイスやモバイル・デバイスの上で動くアプリケーションのためのプラットフォーム。
Android との違いは、Android は、Google による OS も含めたプラットフォームで、Java ME は OS は含まない。いろいろなモバイル機器の OS の上での Java ME は存在している。

Java のこれまで

1995 年の SunWorld で、Sun Microsystems は、Java によるブラウザ WebRunner が、HotJava という名で世界に公開されたところから、Java の歴史は始まる。
1995 年は、Micorsoft が Windows95 を発表し、DOS/V パソコンの販売が始まった年でもある。パーソナルコンピュータ(パソコン、PC)時代の幕開けであり、インターネット時代の到来の年だった。

インターネット時代の到来は、オープンソースソフトウェア（Open Source Software、略称: OSS）を飛躍的に発展させた。OSS は、Linux を普及させ、Java などのオブジェクト指向言語を、汎用機で開発してきたソフトエア開発手法を大きく転換した分岐点となった。

〈Java のこれまでの経過〉

1991 年　Sun Microsystems の中で Green という名前のプロジェクトとして、
後に Java となる Green OS と Oak プログラミング言語の開発がスタート
目標は「ハードウエアに依存しない新しいソフトウエア開発プロセスの仕組みを生み出すこと」
ソフトウエア開発プロセスとは、現在の Java の基本である次の３点を含む

1. オブジェクトを通したメモリアクセス(メモリへのユーザ・アクセスを制限する機能)
2. ガベージ・コレクション(メモリの確保と開放を自動的に管理する機能)
3. 仮想マシン(CPU や OS に依存しないでプログラムを実行する機能)

1992 年～1993 年　Green プロジェクトチームは目標を Stae7 携帯情報端末の開発に置く。
現在のタブレットのような手に乗る対話型の端末の開発だった。
Green OS・Oak 言語・ライブラリーが含まれていて、これまでにない画期的なものだった。
行われたデモンストレーションは、Java が CPU や OS に依存しないプログラミング言語に秘められた潜在能力の片鱗をうかがわせていた。
その後ケーブルテレビのセットトップ・ボックスの開発にチームは力を注ぐようになる。

1994 年　技術者だった Patrick Naughton と Jonathan Payne は Oak 言語を使って、グラフやアニメーションを状況に応じてグラフィカルに表示させる画期的なブラウザ「WebRunner」を開発

1995 年　SunWorld で、Java によるブラウザ WebRunner が、HotJava という名で世界に公開。
また当時人気の高かったブラウザ Netscape が Java のサポートをアナウンスした。
Java はインターネット時代の幕開けとともに、世界のインターネット・ユーザーの注目を浴びるようになる。

1996 年　初めての JavaOne カンファレンスが開催された。6 千人の聴衆が最新の Java テクノロジーについて学ぶためにこのカンファレンスに集まった。
同年最初の正式バージョン Java 1.0 がリリースされる。
世界中の開発者が、Java のプログラミング環境をダウンロードして開発することが可能になった。

1999 年 Java のプラットフォームのための構成が、ソフトウエア開発業者／サービス提供業者／デバイス製造業者の３つのマーケットために再構築される。

2005 年　Java の開発者が 450 万人を超える。
Java を搭載した機器は 25 億個、Java が組み込まれた Smart Card は 10 億枚生産される。
デファクトスタンダードな言語と認められる。

2007年　Googleがアプリケーション開発にJavaを採用したAndroidを発表。専用のJava仮想マシンを搭載。

2010年　Oracle社がSun Microsystems社を買収。Javaの所有はOracleとなる。

2011年　Java SE 7リリース

2014年　Java SE 8リリース

2017年　Java SE 9リリース

2018年　Java SE 10リリース

<Javaのバージョン>

J2SE 5.0以降からは、「製品バージョン」と「開発者バージョン」という、同一のバージョンでありながら異なるバージョン番号を与えられている。

バージョン名	コードネーム	製品バージョン	開発者バージョン	公開日
JDK 1.0	Oak	1		1996/1/23
JDK 1.1		1.1		1997/2/19
J2SE 1.2	Playground	1.2		1998/12/8
J2SE 1.3	Kestrel	1.3		2000/5/8
J2SE 1.4	Merlin	1.4		2002/2/6
J2SE 5.0	Tiger	5	1.5	2004/9/30
Java SE 6	Mustang	6	1.6	2006/12/11
Java SE 7	Dolphin	7	1.7	2011/7/28
Java SE 8	Spider	8	1.8	2014/3/18
Java SE 9				2017/9/21
Java SE 10				2018/3/20

▽　参考）https://en.wikipedia.org/wiki/Java_version_history

※オープンソースソフトウェア（OSS）とは、利用者の目的を問わずソースコードを使用、調査、再利用、修正、拡張、再配布が可能なソフトウェアの総称である。1950年代のコンピュータ上でソフトウェアが稼働するようになった頃、学術機関・研究機関の間でソフトウェアのソースコードはパブリックドメインで共有されていた。1970年代前後よりソフトウェア開発は徐々に商業となり、ソフトウェアの再頒布を禁止するプロプライエタリ・ソフトウェア、ソースコードを非公開とするクローズドソースの文化ができあがった[2]。1980年代より利用者がソフトウェアのソースコードを自由に利用できないことをストレスに感じた人たちはフリーソフトウェア財団やオープンソース・イニシアティブを立ち上げ、ソースコードを一般に公開してソフトウェアの利用者による利用・修正・再頒布を許すことによるソフトウェア開発の発展を提唱し、オープンソースソフトウェアの文化ができあがった。

JDK、JRE、JVM

JDK (Java Development Kit)

Java ベースのアプリケーションの開発に使用できるソフトウェアー式。

JRE (Java Runtime Environment)

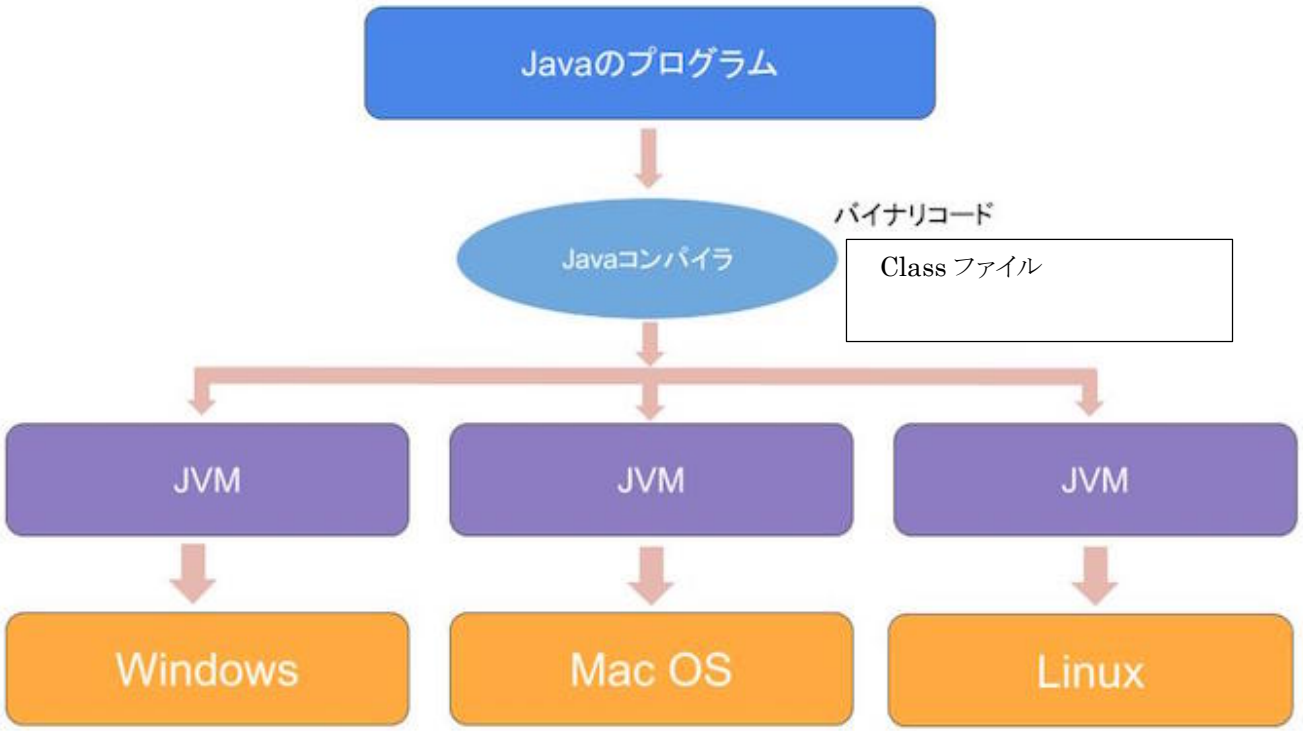
Java プログラミング言語を使って記述されたアプレットやアプリケーションを実行するのに必要な環境
実際に Java プログラムを実行する Java 仮想マシン の実装。

JVM(Java Virtual Machine)

Java 仮想マシン。Java は一度プログラムを書けば、どのマシンでも Java が動くことを思想として作られた言語（「Write once, run anywhere」）。
そのため、JVM のような中間コードをプラットフォーム別に、命令文を変換する仮想マシンが実装される。
JVM は「Java 仮想マシン」「JavaVM」とも言われる。

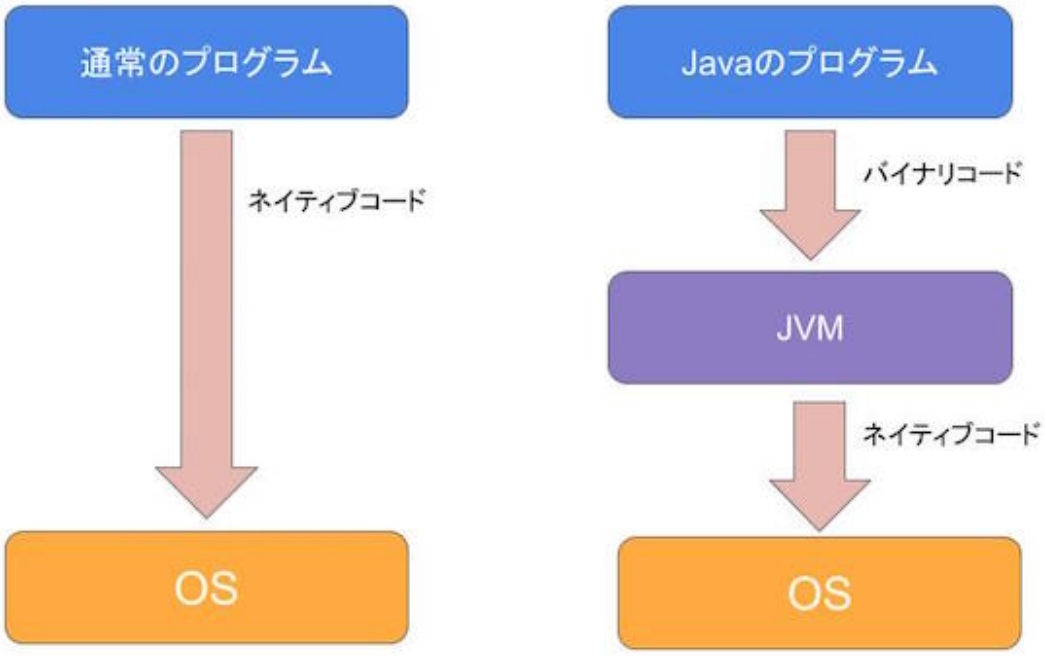
JVM（Java 仮想マシン）の仕組み

JVM の仕組み



JVM があれば Java プログラムを解釈し、対象の OS で実行可能な形式のコードに変換して動かすことができる。
Java で作ったプログラムをプラットフォームに依存しないで動かすためには、JVM 上で Java を動かすことは必須。
Java で作ったプログラムは OS ではなく JVM とやりとりする。OS ごとに特別な設定などは不要。JVM があれば、OS が Windows、Mac、Linux など、環境に依存せずに動かせる。

Java のプログラムを動かすためには

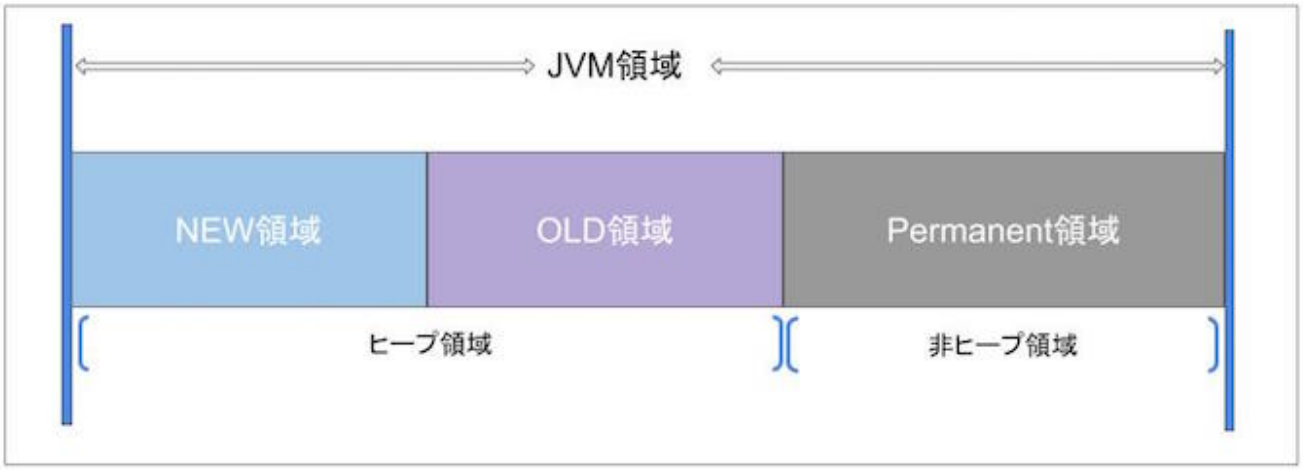


Java プログラムを実行するには、エディタなどで記述したプログラムをコンパイルして Java プログラムを生成(class ファイル)する。プログラミング言語は一般的に、プラットフォームや OS ごとにコンパイラが用意されていて、特定の環境でしか動かすことができない。
しかし、Java プログラムは前述したようにプログラムと OS の間に JVM を仲介させて動作する。

そのため、Java のコンパイルは他のプログラミング言語と違い、JVM が理解できる Java バイナリコードと呼ばれるマシン語に変換する。(Java バイナリコードは、中間的なコードという意味で中間コードとも言う)

Java 仮想マシンはバイナリコードを読み込んで、OS 上で Java プログラムを実行するために、ネイティブコードと呼ばれるコードに変換する。

JVM の構成



JVM には大きく分けて 3 つ領域が存在し、それぞれ OS のメモリ領域を割り当てられて管理する。
ヒープとは、OS やアプリケーションなどの、システムが利用するメモリ領域のことを指す。
JVM のヒープ領域はオブションで自由に設定することが可能。3 つの領域にはそれぞれ役割がある。

Permanent 領域（非ヒープ領域）

Permanent 領域（パーマメント領域）には Java プログラムのクラスやメソッドなど情報が格納される。そのため、規模の大きいプログラムで JVM にクラスを大量にロードする場合は、Permanent 領域のメモリサイズを大きくする必要がある。

New 領域（ヒープ領域）

New 領域は、さらに 3 つの領域に分割されており、オブジェクトがインスタンス化された際にまず Eden 領域に配置される。Eden 領域のメモリがいっぱいになると、オブジェクトを退避するため From 領域と To 領域に格納される。

このような処理は Scavenge GC と呼ばれています。New 領域がいっぱいになると Scavenge GC が頻繁に行われる。

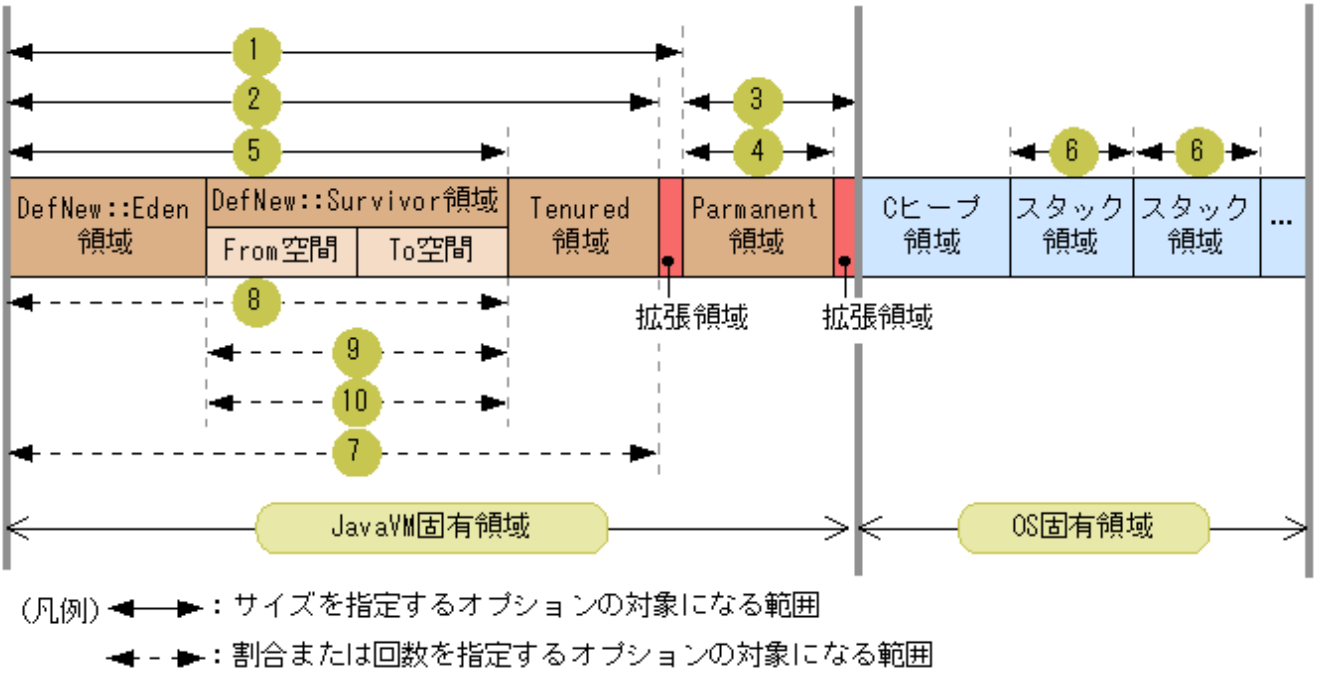
Old 領域（ヒープ領域）※Tenured 領域

New 領域に格納されているオブジェクトで、使用期間が長い情報を管理する。
Old 領域がいっぱいになると Full GC と呼ばれる処理が頻繁に行われます。Full GC は重い処理のため、実行されている間はシステムが止まってしまう。

JVM で使用するメモリ空間の構成と JVM オプション

JVM(JavaVM)で使用するメモリ空間の構成と，JVM オプションについて説明する。
JVM では，JVM 固有領域と OS 固有領域という，2 種類のメモリ空間を使用する。
JVM で使用するメモリ空間の構成を次の図に示す。なお、図中の番号①～⑩は、下方の表の項番と対応する。

【JVM で使用するメモリ空間の構成】



(凡例) ◄——► : サイズを指定するオプションの対象になる範囲
◄- -► : 割合または回数を指定するオプションの対象になる範囲

それぞれの領域について説明する。
DefNew::Eden 領域(エデン領域)、DefNew::Survivor 領域(サバイバ領域)、および Tenured 領域（テニュアド領域）を合わせた領域を、Java ヒープという。

DefNew::Eden 領域

new によって作成された Java オブジェクトが最初に格納される領域。

DefNew::Survivor 領域

DefNew 領域に格納されていた Java オブジェクトのうち、コピーガーベージコレクション実行時に破棄されなかった Java オブジェクトが格納される領域。
DefNew::Survivor 領域は、From 空間と To 空間で構成。From 空間と To 空間のサイズは同じ。

Tenured 領域(Old 領域)

長期間必要であると判断された Java オブジェクトが格納される領域。
DefNew::Survivor 領域で指定回数を超えてコピーガーベージコレクションの実行対象となり、破棄されなかった Java オブジェクトが、この領域に移動する。

Permanent 領域

ロードされた class などの情報が格納される領域。

C ヒープ領域

JVM 自身が使用する領域。JNI で呼び出されたネイティブライブラリでも使用する。

スタック領域

Java スレッドのスタック領域。

【表　JVM メモリ空間のサイズや割合などを指定する JVM オプション】

項番	オプション名	オプションの意味
①	-Xmx<size>	Java ヒープの最大サイズを設定。
②	-Xms<size>	Java ヒープの初期サイズを設定。
③	-XX:MaxPermSize=<size>	Permanent 領域の最大サイズを設定。
④	-XX:PermSize=<size>	Permanent 領域の初期サイズを設定。
⑤	-Xmn<size>	DefNew 領域の初期値および最大値を設定。
⑥	-Xss<size>	１スタック領域の最大サイズを設定。

⑦	-XX:NewRatio=<value>	DefNew 領域に対する Tenured 領域の割合を設定。 <value>が 2 の場合は，DefNew 領域と Tenured 領域の割合が，1:2 になる。
⑧	-XX:SurvivorRatio=<value>	DefNew::Survivor 領域の From 空間と To 空間に対する DefNew::Eden 領域の割合を設定。 <value>に 8 を設定した場合は，DefNew::Eden 領域，From 空間，To 空間の割合が，8:1:1 になる。
⑨	-XX:TargetSurvivorRatio=<value>	ガーベージコレクション実行後の DefNew::Survivor 領域内で Java オブジェクトが占める割合の目標値を設定。
⑩	-XX:MaxTenuringThreshold=<value>	コピーガーベージコレクション実行時に，From 空間と To 空間で Java オブジェクトを入れ替える回数のしきい値を設定。 設定した回数を超えて入れ替え対象になった Java オブジェクトは，Tenured 領域に移動。

注 <size>の単位はバイト。

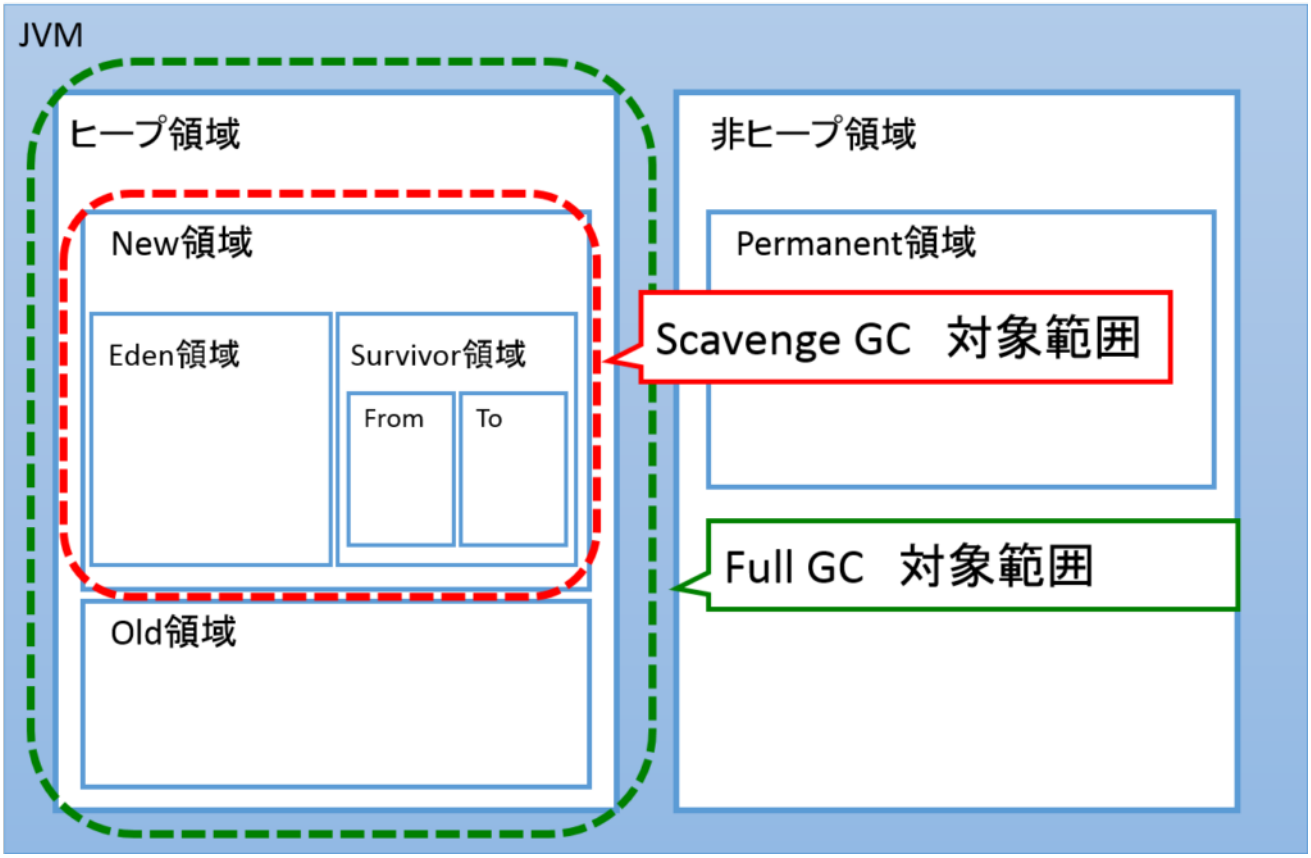
Java の GC（Scavenge GC と Full GC）

GC（ガーページコレクション）とは、Garbage Collection(ごみ収集)の略であり、メモリ上のごみオブジェクトを破棄するための機能である。

例えば C 言語であればプログラム中で確保したメモリはソースコードで明示的に開放する必要があり、コーディングを忘れるとメモリリークの原因となってしまう。Java では GC が存在するおかげでプログラマが明示的にメモリを開放しなくとも、メモリ上から自動的に不要なオブジェクトを破棄してくれるので、コーディングミスによるメモリリークの発生を防ぐことができる。
Java の GC には Scavenge GC と Full GC の 2 種類が存在する。2 つの違いを簡単に言うと、Scavenge GC が「日頃のお掃除」であるのに対し、Full GC は「年末の大掃除」のような違いがある。

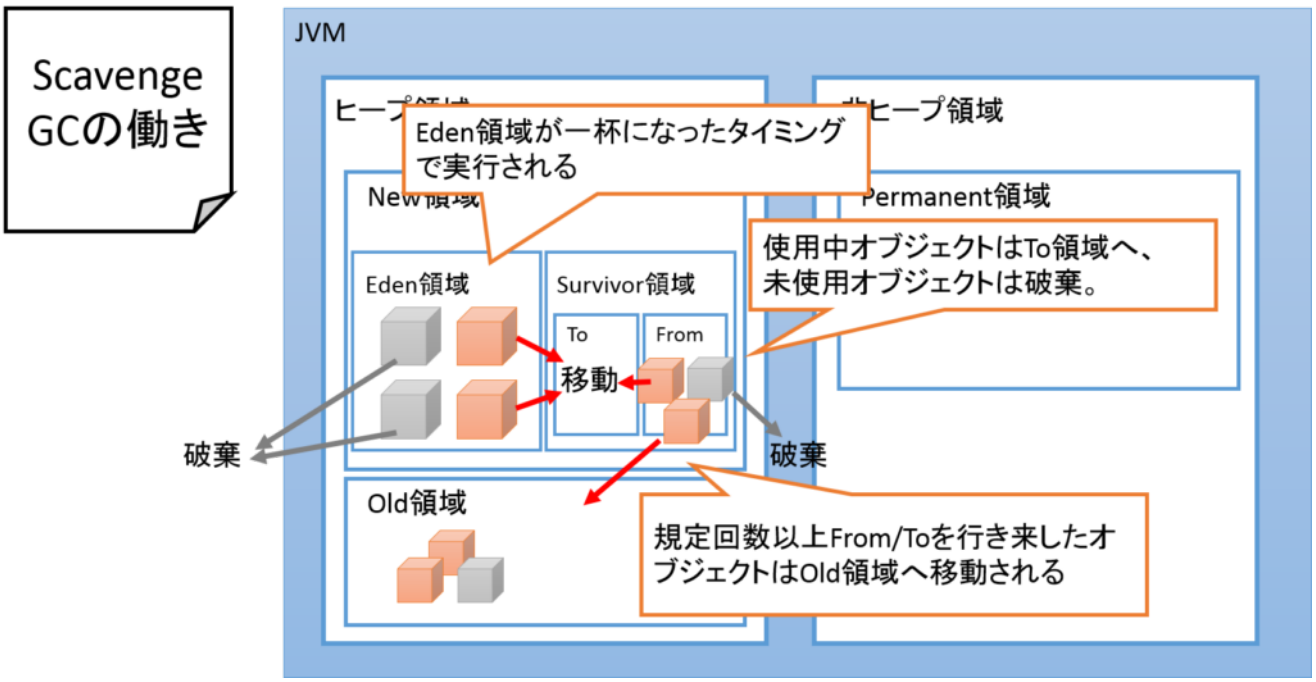
「日頃の掃除」は頻度は高いが、よく使う部分のみでかかる時間も少ない。対して「年末の大掃除」は頻度は年に一回だが、広い範囲を長い時間をかけて行う。そして、掃除する対象はどちらも JVM のヒープ領域内である。

【JVM のメモリ構造】



Scavenge GC と Full GC の違いの一つはその対象範囲である。Scavenge GC はヒープ領域の中でも New 領域にあるオブジェクトのみを対象とするのに対して、Full GC ではヒープ領域全体が処理対象となる。当然、その分だけ処理時間も長くなる。

【Scavenge GC の動き】

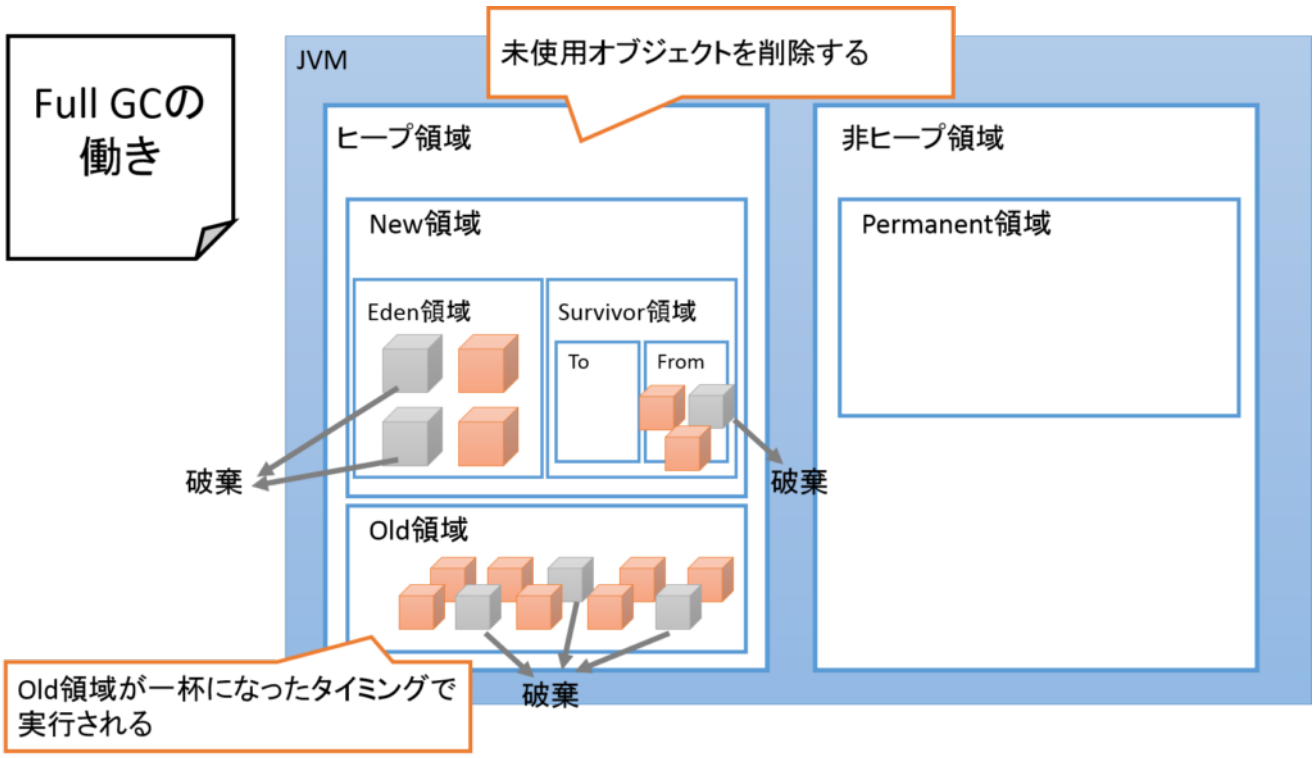


Scavenge GC で処理対象となるのは、New 領域上に存在するオブジェクトである。Scavenge GC は New 領域上の Eden 領域が一杯になったときに起動し、Eden 領域と Survivor 領域上の未使用オブジェクトを破棄し、使用中オブジェクトを To 領域へと移動させる。

次回に GC が起動した場合には From/To 領域として扱われる領域が入れ替わるため、長い間使用中のままのオブジェクトがあると、GC のたびに Survivor 領域内をいったりきたりすることになる。そして、Survivor 領域内を移動した回数が閾値（MaxTenuringThreshold）を超えると、Old 領域に移動されることになる。

Old 領域は Scavenge GC では処理対象とならないため、Full GC が稼動するまでは生き残り続けることになる。

【Full GC の動き】



Full GC は、Old 領域が一杯になったときに稼動する GC であり、文字通りヒープの全領域を対象とした GC である。Scavenge GC のようにオブジェクトの移動は行わず、不要なオブジェクトを破棄してメモリの空き領域を確保していく。Scavenge GC と比較すると、対象範囲の広さと圧縮処理などを含むため低速であるという特徴がある。

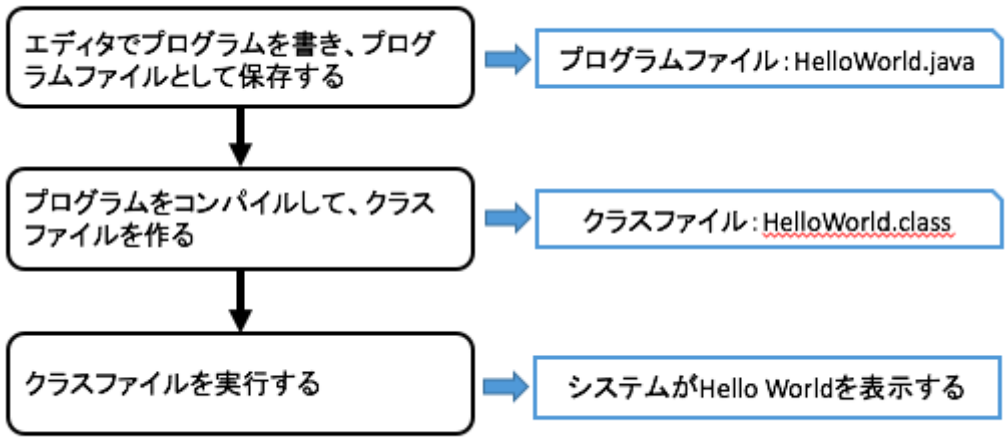
GC が原因でシステムの性能が想定どおりでないという場合は、ほぼこの Full GC の頻度が高すぎることが原因であるといっていい。

	Scavenge GC	Full GC
発生頻度	**高い** （数十秒に一度）	**低い** (数分～数十分に一度)
対象範囲	**狭い** （New領域のみ）	**広い** （ヒープ領域全体）
処理時間	**短い** (数ms～数十ms)	**長い** （数百ms～数s）
処理内容	①未使用オブジェクトの破棄 ②使用中のオブジェクトの移動	①未使用オブジェクトの破棄

「Hello Java!」

別紙のコンパイル環境を整える。

【順序】



次のコードを次の場所(c:¥work)に、HelloWorld.java と名前を付けて保存する。

□ c:¥work¥HelloWorld.java

```
public class HelloWorld{
    public static void main(String[] args){
      System.out.println("Hello Java!!");
    }
}
```

コマンドプロンプトを開き、ソース(HelloWorld.java)をコンパイルする。

```
> javac HelloWorld.java
```

プログラム(HelloWorld.class)を実行する。

```
> java HelloWorld
```

<END>