# 自动内存管理机制

## Java内存区域与内存溢出异常

### 运行时数据区域（内存模型）

#### 程序计数器

是最小的一块内存区域，它的作用是当前线程所执行的字节码的行号指示器

在虚拟机的模型里，字节码解释器工作时就是通过改变这个计数器的值来选取下一条需要执行的字节码指令，分支、循环、异常处理、线程恢复等基础功能都需要依赖计数器完成

每条线程都有一个独立的程序计数器，既线程私有

#### Java虚拟机栈

描述的是Java 方法执行的内存模型：每个方法被执行的时候，都会创建一个“栈帧”用于存储局部变量表(包括参数)、操作栈、方法出口等信息。每个方法被调用到执行完的过程，就对应着一个栈帧在虚拟机栈中从入栈到出栈的过程。声明周期与线程相同，是线程私有的

#### 本地方法栈

与虚拟机栈基本类似，区别在于本地方法栈是为Native方法服务

#### Java堆

该内存区域存放了对象实例及数组(所有new的对象)

是java虚拟机所管理的内存中最大的一块内存区域，也是被各个线程共享的内存区域，在JVM启动时创建。

由于现在收集器都是采用分代收集算法，堆被划分为新生代和老年代。

新生代（复制算法进行回收）主要存储新创建的对象和尚未进入老年代的对象。

老年代（标记、标记-整理算法进行回收）存储经过多次新生代GC(Minor GC)任然存活的对象

#### 方法区

也称”永久代” 、“非堆”，  它用于存储虚拟机加载的类信息、常量、静态变量、是各个线程共享的内存区域

##### 运行时常量池

是方法区的一部分，Class文件中除了有类的版本、字段、方法、接口等描述信息外，还有一项信息是常量池，用于存放编译器生成的各种符号引用，这部分内容将在类加载后放到方法区的运行时常量池中

### 对象在java堆中的维护

#### 对象的创建

##### 对象的内存分配

对象所需的内存大小在类加载完成后即可确定

指针碰撞：堆内存是规整的，并且由指针分为已使用和未使用两部分。此时移动指正即可

空闲列表：堆内存不规整时，无法指针碰撞，只能维护一个列表记录哪些内存是可用的。分配时找到一块足够大的内存即可

堆是否规整取决于垃圾收集器是否带有压缩整理功能

##### 对象的内存布局

###### 对象头

包括自身的运行时数据（哈希码、GC分代年龄、锁状态标识等）、类型指针（指向类元数据的指针）

###### 实例数据

对象真正储存的有效信息，既在程序代码中定义的各种字段类型

###### 对齐填充

占位符作用，保证对象内存大小是8字节的整数倍

##### 对象的访问定位

### 内存溢出

#### OutOfMemoryError异常

内存溢出异常，可用内存不满足申请分配内存时发生，会在各个内存区域发生

#### StackOverflowError异常

虚拟机栈和本地方法栈中，申请分配的栈深度大于虚拟机所允许的最大深度时发生，只在栈区域发生

# 垃圾收集器与内存分配策略

堆和方法区需要进行垃圾收集

## 对象死亡判断

### 引用计数法

对象被引用则+1，引用失效则-1

实现简单，但是无法解决循环引用的问题

### 可达性分析算法

GC Root是否可达

GC Root的对象包括：

虚拟机栈中引用的对象

方法区中类静态属性引用的对象

方法区中常量引用的对象

本地方法栈中引用的对象

## 垃圾收集算法

### 标记-清除算法

先标记要回收的对象、再统一回收

效率低、易产生大量不连续的内存碎片

### 复制算法

将存活对象复制到一侧，然后集中收集另一侧。已空间换时间

### 标记-整理算法

将被标记对象移动到一侧，然后进行回收

# 虚拟机执行子系统

## 虚拟机类加载机制

### 类的生命周期

加载、连接、初始化、使用、卸载

连接：验证、准备、解析

整个过程按顺序开始，但是过程交叉无序

#### 加载

1. 通过一个类的全限定名获取类的二进制字节流
2. 将类信息、常量、静态变量存放到方法区
3. 在内存中生成一个Class对象

#### 验证

文件格式验证、元数据验证、字节码验证、符号引用验证

#### 准备

为类变量分配内存并设置类变量初始值

#### 解析

将常量池中的符号引用替换为直接引用

#### 初始化

准备阶段虚拟机为类变量设置初始值，初始化是程序员在构造函数中进行类变量初始化

### 类加载器

类加载过程中加载阶段完成“通过一个类全限定名获取类的二进制字节流”。实现这个动作的代码块称为类加载器

类加载器和类本身一同确定类在虚拟机中唯一性

#### 双亲委派模式

除顶层启动类加载器外，所有的类加载器都应该有父类加载器。如果一个类加载器收到类加载的请求，首先会委派给父类加载器进行加载

解决基础类统一的问题（类在虚拟机中的唯一性由类加载器和类本身共同确定）

破坏双亲委派模式：osgi、jndi父类加载器调用子类加载器

##### 启动类加载器

将<JAVA\_HOME>\lib中的类库加载到虚拟机中，开发者不能直接使用

##### 扩展类记载器

将<JAVA\_HOME>\lib\ext中的类库加载到虚拟机中，开发者可以直接使用

##### 应用程序类加载器

加载用户类路径上指定的类文件开发者可以直接使用

### 分派

#### 静态分派

虚拟机（准确的说应该是编译器）在重载时通过参数的静态类型（声明类型）而不是实际类型作为判断条件

典型应用：方法重载

静态分派发生在编译阶段

#### 动态分派

运行时根据实际类型定位方法的分配动作

典型应用：方法重写

静态分派发生在运行阶段

### 代理模式

#### 静态代理

需要代理类和目标类实现相同的接口，通过代理类访问目标类的方法

缺点：需要额外维护和目标类相同数量的代理类

#### 动态代理

基于字节码生成技术，通过Proxy类的newProxyInstomce方法在内存中动态生成代理类

# 程序编译与代码优化

## 晚期（运行期）优化

# 高效并发

## Java内存模型和线程

### 主内存与工作内存

线程对变量的所有操作（读取和赋值）必须在工作内存（缓存或寄存器）中进行，工作内存通过load或save操作主内存

### Volatile型变量

可见性：变量值修改后立即刷新到主内存中，其他线程要使用变量时要先从主内存中更新最新值到工作内存中才能进行操作。

禁止重排序优化