

JVM调优实战

学习目标

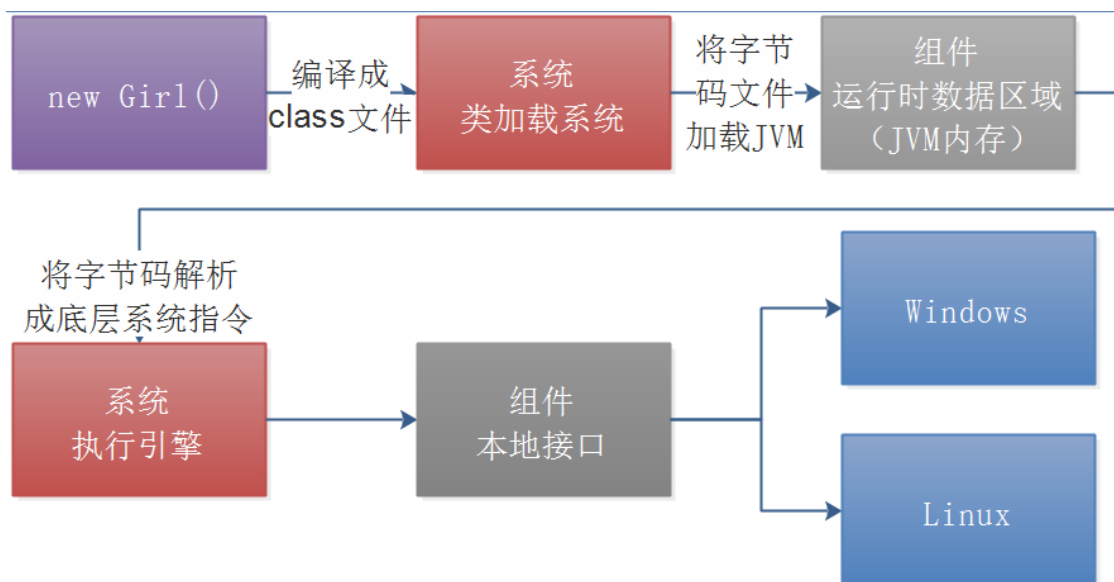
- JVM介绍
- 为什么要学习JVM
- JVM内存模型
- JVM调试实战

1 JVM介绍

JVM是Java Virtual Machine (Java虚拟机) 的缩写, JVM是一种用于计算设备的规范, 它是一个虚构出来的计算机, 是通过在实际的计算机上仿真模拟各种计算机功能来实现的。Java虚拟机包括一套字节码指令集、一组寄存器、一个栈、一个垃圾回收堆和一个存储方法域。JVM屏蔽了与具体操作系统平台相关的信息, 使Java程序只需生成在Java虚拟机上运行的目标代码 (字节码), 就可以在多种平台上不加修改地运行。JVM在执行字节码时, 实际上最终还是把字节码解释成具体平台上的机器指令执行。

JVM包含两个子系统和两个组件, 两个子系统为Class loader(类装载)、Execution engine(执行引擎); 两个组件为Runtime data area(运行时数据区)、Native Interface(本地接口)。

- Class loader(类装载): 根据给定的全限定名类名(如: java.lang.Object)来装载class文件到Runtime data area中的method area。
- Execution engine (执行引擎) : 执行classes中的指令。
- Native Interface(本地接口): 与native libraries交互, 是其它编程语言交互的接口。
- Runtime data area(运行时数据区域): JVM的内存区域。



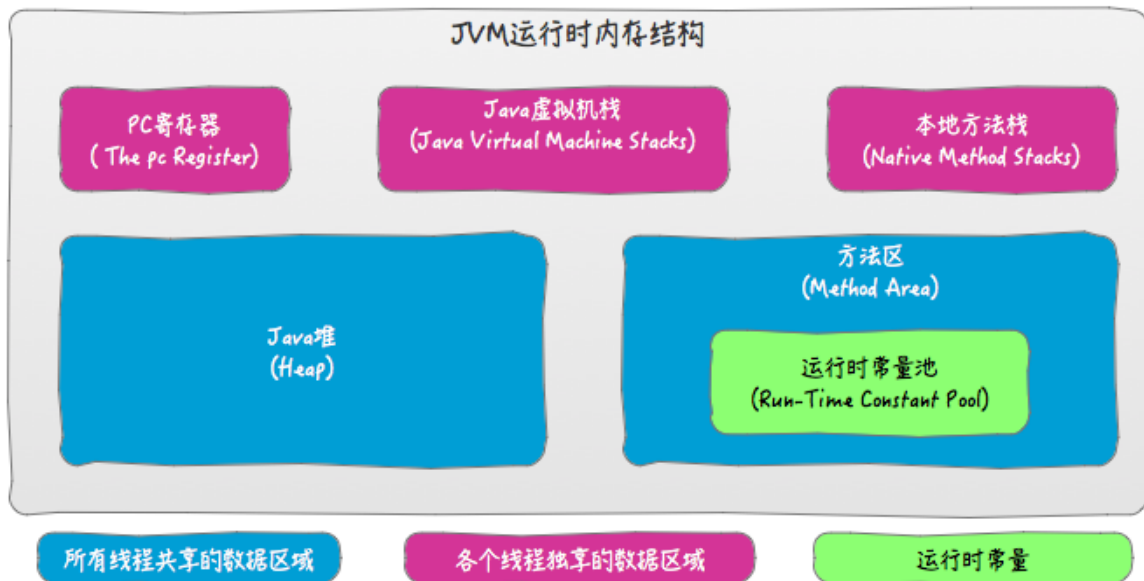
2 为什么要学习JVM

JVM按上面的介绍, 还提供了一个非常重要的功能就是大家熟知的内存管理。相对于C、C++语言每次去申请空间都要手动地去 allcate,使用结束之后还要手动free掉。JVM从这个层面释放程序员们对内存分配和管理的精力, 只需要去关注业务逻辑地实现, 不再需要花费更多的心力在内存分配和回收上。

JVM帮我们管理了内存是帮我们省事了, 但是当代码写的不恰当的时候会导致内存溢出或者内存泄漏, 这个时候, 如果你对JVM一无所知, 就**无法快速定位和处理问题?**

3 JVM内存模型

3.1 JVM运行时内存区域结构



1. PC寄存器

每个线程启动的时候，都会创建一个程序计数器(Program Counter)，就是一个指针，指向方法区中的方法字节码（用来存储指向下一条指令的地址，也即将要执行的指令代码），由执行引擎读取下一条指令。

2. 栈 JVM Stack

各种基本数据类型（boolean、byte、char、short、int、float、long、double）、对象引用（引用指针，并非对象本身）

3. 本地方法栈 Native Method Stack

Java调用非Java代码的接口，该方法的实现由非Java语言实现。

4. 方法区 Method Area

用于存储虚拟机加载的：静态变量+常量+类信息+运行时常量池（类信息：类的版本、字段、方法、接口、构造函数等描述信息）

5. 堆 Java Heap

所有的对象实例以及数组都要在堆上分配，此内存区域的唯一目的就是存放对象实例

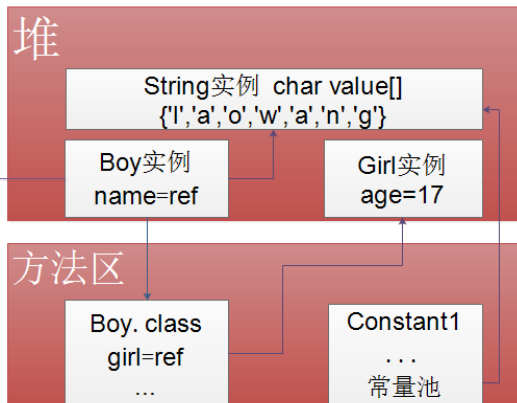
堆是Java虚拟机所管理的内存中最大的一块。所有线程共享的一块内存区域，在虚拟机启动时创建

3.2 java实例对象在内存中的分配

```
public class Boy {
    private String name;
    private static Girl girl = new Girl();
    public Boy(String name) {
        this.name = name;
    }
    public static void main(String[] args)
    {
        Boy boy = new Boy("laowang");
        int age = 18;
        System.out.println(age);
    }
}
class Girl {
    private int age = 17;
}
```

虚拟机栈

age=18
boy=ref
args=null



从图中我们可以看出，普通的java实例对象内存分配，主要在这三个区域：虚拟机栈、堆、方法区。

3.2.1 从内存区域来分析

- **虚拟机栈**:只存放局部变量
- **堆**:存储对象的实例
- **方法区**：存放Class信息和常量信息。

3.2.2 从变量的角度来分析

局部变量:

- 基本类型的值直接存在栈中。如age=18
- 如果是对象的实例，则只存储对象实例的引用。如Boy=ref

实例变量：存放在堆中的对象实例中。如Boy的实例变量 name=ref

静态变量：存放在方法区中的常量池中。如Boy.class中的Girl=ref

4 JVM调试实战

4.1 调试工具

VisualVM是JDK自带的一款全能型性能监控和故障分析工具，包括对CPU使用、JVM堆内存消耗、线程、类加载的实时监控，内存dump文件分析，垃圾回收运行情况的可视化分析等，对故障排查和性能调优很有帮助。

环境：安装JDK1.8并配置

打开dos窗口输入: jvisualvm , 如下图:



4.2 数据库准备

```
/**创建数据库 以及 美女表**/  
CREATE DATABASE IF NOT EXISTS `itcast_jvm` DEFAULT CHARSET utf8 COLLATE  
utf8_general_ci;
```

```

DROP TABLE IF EXISTS `girl`;
CREATE TABLE `girl` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT COMMENT 'ID',
  `name` varchar(100) DEFAULT NULL COMMENT '女孩姓名',
  `age` varchar(3) DEFAULT NULL COMMENT '年龄',
  `address` varchar(100) DEFAULT NULL COMMENT '地址',
  PRIMARY KEY (`id`)
) ENGINE=MyISAM AUTO_INCREMENT=0;

/**创建存储过程造数据**/
DROP PROCEDURE IF EXISTS proc_girl;
DELIMITER $$
SET AUTOCOMMIT = 0$$
CREATE PROCEDURE proc_girl()
BEGIN
DECLARE v_cnt DECIMAL (10) DEFAULT 0 ;
dd:LOOP
    INSERT INTO girl (name,age,address)
VALUES (CONCAT('美女技师',v_cnt),FLOOR(18 + (RAND() * 7)),CONCAT('东莞',v_cnt,'号'));
    COMMIT;
    SET v_cnt = v_cnt+1 ;
    IF v_cnt = 10000000 THEN LEAVE dd;
    END IF;
END LOOP dd ;
END;$$
DELIMITER ;

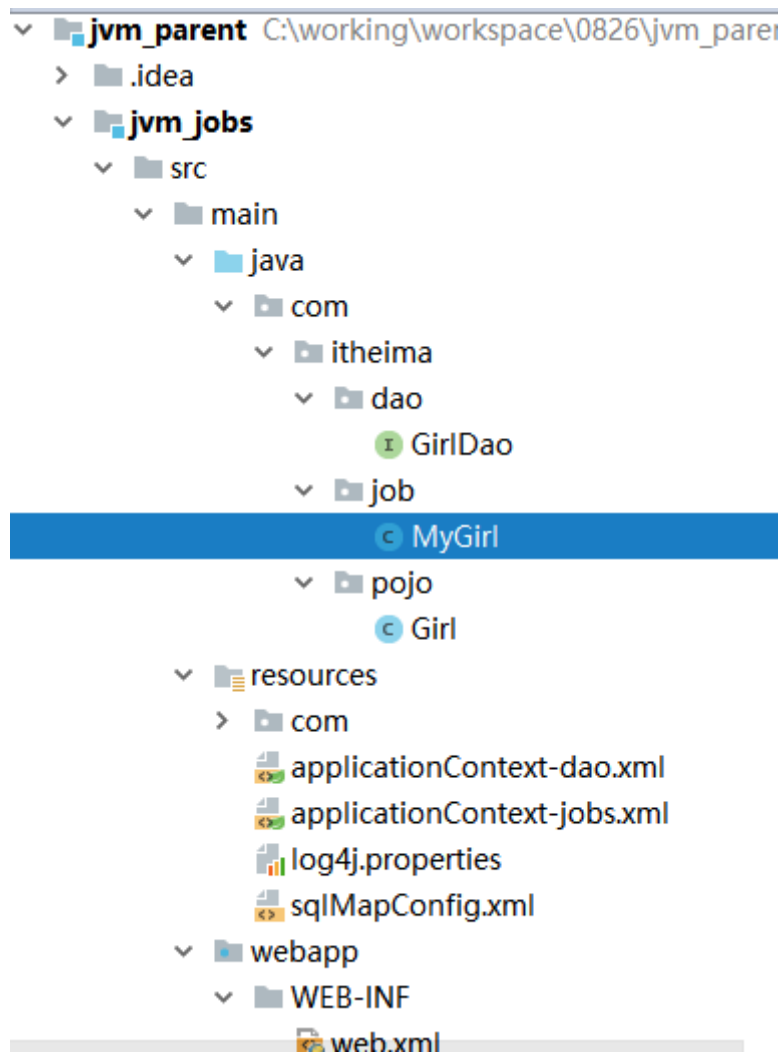
/**执行存储过程 大概执行10分钟左右**/
call proc_girl;

/**测试语句**/
select * from girl where name like '%美女技师%' order by age limit 10,100

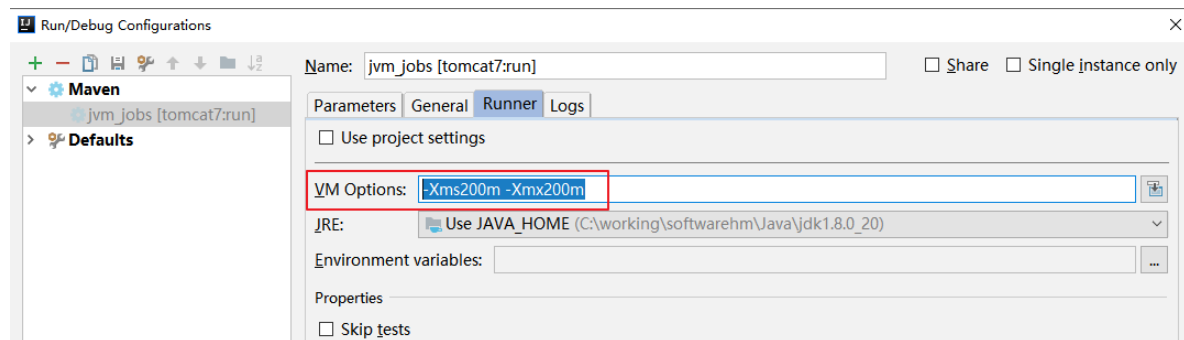
```

4.3 启动工程

在idea工具打开代码目录中的jvm_parent工程，如下图：

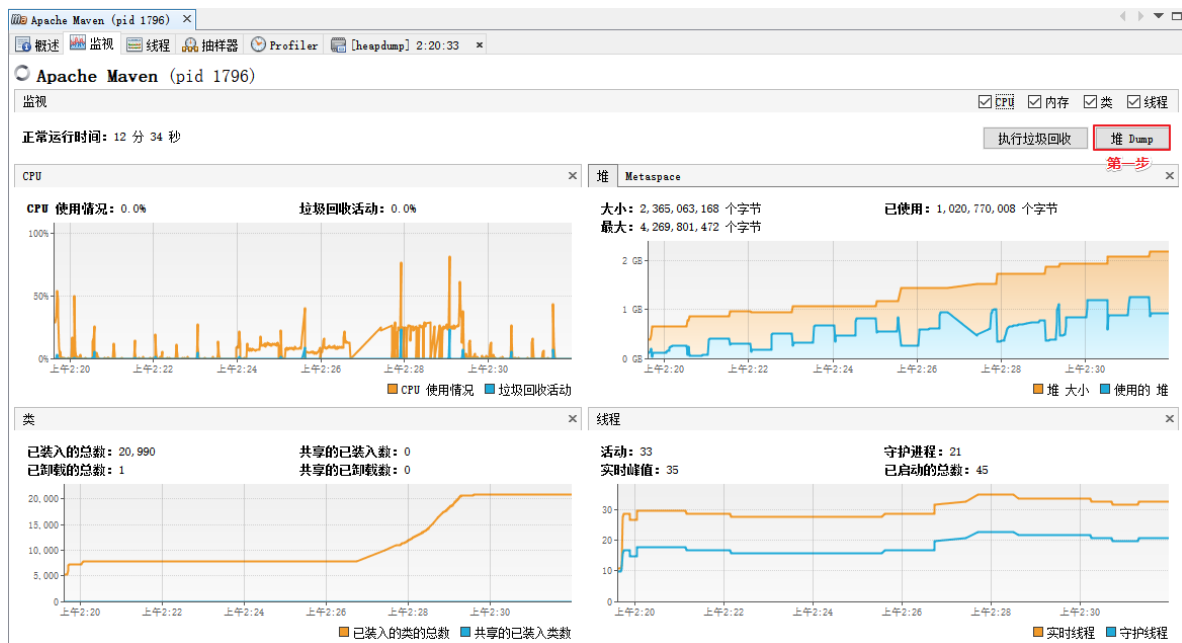


并设置-Xms200m -Xmx200m（堆和栈内存大小设置），如下图

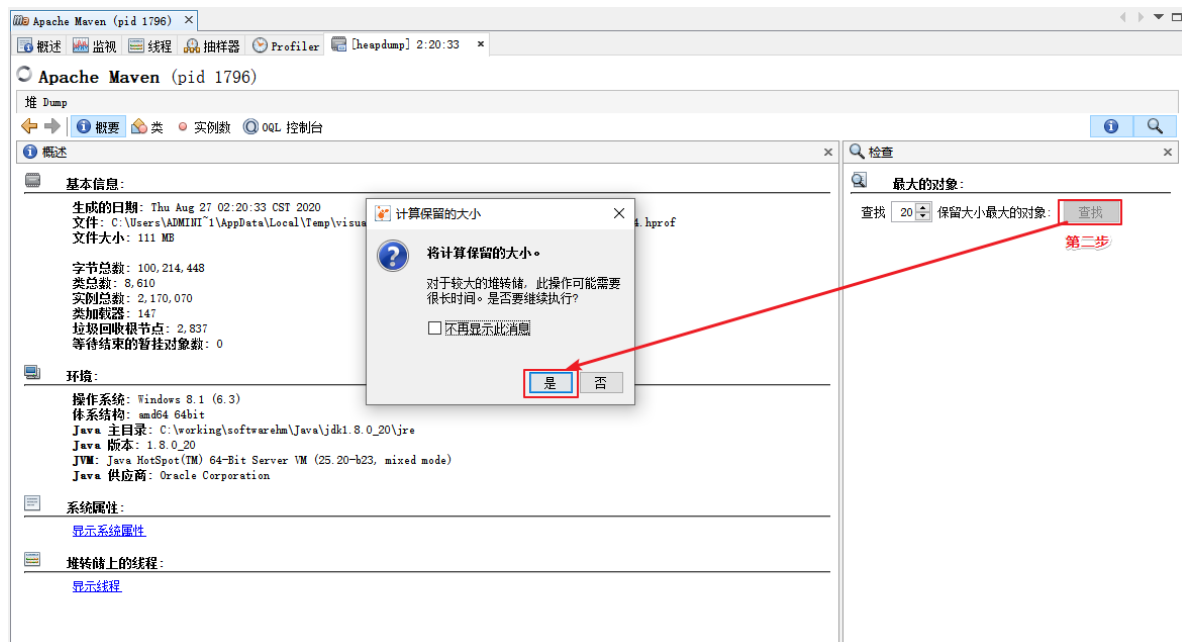


4.4 JVM分析

4.4.1 进入堆页面



4.4.2 查询最大的对象



4.4.3 重点关注靠前的对象

检查		最大的对象:	
查找	20	保留大小最大的对象:	查找
类名	保留大小		
com.itheima.job.MyGirl#1	49,431,984		
java.util.ArrayList#13000	49,431,952		
java.lang.Object[]#13111	49,431,920		
org.eclipse.aether.DefaultRepositoryCache#1	5,874,044		
java.util.concurrent.ConcurrentHashMap#38	5,874,020		
java.util.concurrent.ConcurrentHashMap\$Node[]#23	5,873,920		
org.codehaus.plexus.classworlds.realm.ClassRealm#5	4,668,559		
java.util.Vector#132	4,436,460		
java.lang.Object[]#11075	4,436,424		
org.apache.tomcat.util.bcel.classfile.ConstantUtf8\$1#1	4,026,092		
class org.apache.tomcat.util.bcel.classfile.ConstantUtf8	4,026,092		
com.sun.tools.javac.file.ZipFileIndex#70	2,752,196		
com.sun.tools.javac.file.ZipFileIndex\$Entry[]#72	2,241,426		
java.util.concurrent.ConcurrentHashMap\$Node#7335	2,088,654		
java.util.Collections\$SynchronizedMap#2	2,088,610		
java.util.WeakHashMap#77	2,088,554		

4.4.4 找到grilDao接口

Apache Maven (pid 1796)

堆 Dump

实例数: 1 | 实例大小: 32 | 总大小: 32 | 保留大小: 49,431,984

com.itheima.job.MyGirl

实例	保留	字段	类型	值	保留
#1	49,431,984	this	MyGirl	#1	49,431,984
		girlList	ArrayList	#13000	49,431,952
		girlDao	\$Proxy34	#1	64
		h	MapperProxy	#1	40
		methodCache	ConcurrentHashMap	#133	638
		mapperInterface	Class	#829 - GirlDao	0
		sqlSession	SqlSessionTemplate	#1	128
		exceptionTranslator	MyBatisExceptionTranslator	#1	32
		sqlSessionProxy	\$Proxy33	#1	48
		executorType	ExecutorType	#1 SIMPLE	28
		sqlSessionFactory	DefaultSqlSessionFactory	#1	24
		configuration	Configuration	#1	39,431,984
		classLoader	WebappClassLoader	#1	83,431,984

引用

字段	类型	值	保留
this	MyGirl	#1	49,431,984
val	ConcurrentHashMap	#17651	44
targetObject	MethodInvocation	#1	130

4.4.5 找到PreparedStatementHandler

起始页 × Apache Maven (pid 2496) ×

概述 监视 线程 抽样器 Profiler [heapdump] 12:14:06 ×

Apache Maven (pid 2496)

堆 Dump

org.apache.ibatis.session.Configuration 实例: 1 | 实例大小: 316 | 总大小: 316 | 保留大小: 9,128

实例	字段	类型	值	保留
#1	this	Configuration		9,128
	cacheRefMap	HashMap	#8629	64
	incompleteMethods	LinkedList	#495	40
	incompleteResultMaps	LinkedList	#496	40
	incompleteCacheRefs	LinkedList	#497	40
	incompleteStatements	LinkedList	#498	40
	sqlFragments	Configuration\$StrictMap	#3	72

引用

字段	类型	值	保留
this (Java frame)	Configuration		9,128
configuration	DefaultResultSetHandler	#1	1,729
configuration	DefaultParameterHandler	#1	56
configuration (Java frame)	SimpleExecutor	#1	987
configuration	ParameterMapping	#1	96
configuration	ParameterMapping	#2	96
configuration	ResultMap	#2	770
configuration (Java frame)	DefaultSqlSession	#1	42
configuration (Java frame)	PreparedStatementHandler	#1	1,873
configuration	ResultMap	#1	950
configuration	MappedStatement	#2	1,158

显示实例
显示最近的垃圾回收根节点
在线程中显示
垃圾回收根节点
复制从根开始的路径

4.4.6 找到boundSql

起始页 × Apache Maven (pid 2496) ×

概述 监视 线程 抽样器 Profiler [heapdump] 12:14:06 ×

Apache Maven (pid 2496)

堆 Dump

org.apache.ibatis.executor.statement.PreparedStatementHandler 实例: 1 | 实例大小: 88 | 总大小: 88 | 保留大小: 1,873

实例	字段	类型	值	保留
#1	this	PreparedStatementHandler		1,873
	boundSql	BoundSql	#1	820
	metaParameters	MetaObject	#3	88
	additionalParameters	HashMap	#8929	348
	parameterObject	HashMap	#8921	348
	parameterMappings	ArrayList	#10372	328
	sql	String	select * from...	202
	rowBounds	WebappClassLoader	#1	828,192
	mappedStatement	RowBounds	#1	24
	executor	MappedStatement	#1	1,290
	parameterHandler	CachingExecutor	#1	120
	parameterHandler	DefaultParameterHandler	#1	56

引用

字段	类型	值	保留
this (Java frame)	PreparedStatementHandler		1,873
delegate (Java frame)	RoutingStatementHandler	#1	24

显示实例
在类视图中显示
复制从根开始的路径

4.4.7 分析Sql

根据实际情况进行优化

5 常见面试题

5.1 介绍下JVM内存模型

JVM 分为堆区和栈区，还有方法区，初始化的对象放在堆里面，引用放在栈里面，`class` 类信息常量池（`static` 常量和 `static` 变量）等放在方法区

- a. 方法区：主要是存储类信息，常量池（`static` 常量和 `static` 变量），编译后的代码（字节码）等数据
- b. 堆：初始化的对象，成员变量（那种非 `static` 的变量），所有的对象实例和数组都要在堆上分配
- c. 栈：栈的结构是栈帧组成的，调用一个方法就压入一帧，帧上面存储局部变量表，操作数栈，方法出口等信息，局部变量表存放的是 8 大基础类型加上一个应用类型，所以还是一个指向地址的指针
- d. 本地方法栈：主要为 `Native` 方法服务
- e. 程序计数器：记录当前线程执行的行号

5.2 说一说类加载的过程

类装载分为以下 5 个步骤：

- a. 加载：根据查找路径找到相应的 `class` 文件然后导入；
- b. 验证：检查加载的 `class` 文件的正确性；
- c. 准备：给类中的静态变量分配内存空间；
- d. 解析：虚拟机将常量池中的符号引用替换成直接引用的过程。符号引用就理解为一个标示，而在直接引用直接指向内存中的地址；
- e. 初始化：对静态变量和静态代码块执行初始化工作。

5.3 哪些区域可能发生OutOfMemory异常？

- a. java堆(区域)-存放对象的实例。
异常信息：`java.lang.OutOfMemory Error:Java heap space`
参数：`-Xms`（初始化堆），`-Xmx`（最大堆），`-Xmn`（新生代）
- b. 虚拟机栈和本地方法栈(区域)-控制方法的执行。
异常信息：`java.lang.OutOfMemory Error`
参数：`-Xss`
- c. 方法区(区域)-存放Class的相关信息，如类名、访问修饰符、常量池、字段描述、方法描述等。
异常信息：`java.OutOfMemory Error:PermGen space`
参数：`-XX:PermSize-XX:MaxPermSize`
- d. 本机直接内存(区域)-如NIO，就是直接通过此内存实现。
异常信息：`java.lang.OutOfMemory Error`,有`allocate`、`Native`字样
参数：`-MaxDirectMemorySize`如不指定则与`-Xmx`一致。

5.4 垃圾收集的方法有哪些？

- a. 标记-清除算法：标记无用对象，然后进行清除回收。缺点：效率不高，无法清除垃圾碎片。
- b. 复制算法：按照容量划分二个大小相等的内存区域，当一块用完的时候将活着的对象复制到另一块上，然后再把已使用的内存空间一次清理掉。缺点：内存使用率不高，只有原来的一半。
- c. 标记-整理算法：标记无用对象，让所有存活的对象都向一端移动，然后直接清除掉端边界以外的内存。
- d. 分代算法：根据对象存活周期的不同将内存划分为几块，一般是新生代和老年代，新生代基本采用复制算法，老年代采用标记整理算法。

5.5 说一下 JVM 的主要组成部分及其作用？

JVM包含两个子系统和两个组件。

两个子系统为Class loader(类装载)、Execution engine(执行引擎)；

两个组件为Runtime data area(运行时数据区)、Native Interface(本地接口)。

a.Class loader(类装载)：根据给定的全限定名类名(如：java.lang.Object)来装载class文件到Runtime data area中的method area。

b.Execution engine(执行引擎)：执行classes中的指令。

c.Native Interface(本地接口)：与native libraries交互，是其它编程语言交互的接口。

d.Runtime data area(运行时数据区域)：这就是我们常说的JVM的内存。

作用：首先通过类加载器（ClassLoader）会把 Java 代码转换成字节码，运行时数据区（Runtime Data Area）再把字节码加载到内存中，而字节码文件只是 JVM 的一套指令集规范，并不能直接交给底层操作系统去执行，因此需要特定的命令解析器执行引擎（Execution Engine），将字节码翻译成底层系统指令，再交由 CPU 去执行，而在这个过程中需要调用其他语言的本地库接口（Native Interface）来实现整个程序的功能。

5.6 说一下JVM调优工具？

JDK 自带了很多监控工具，都位于 JDK 的 bin 目录下，其中最常用的是 jconsole 和 jvisualvm 这两款视图监控工具。

jconsole：用于对 JVM 中的内存、线程和类等进行监控；

jvisualvm：JDK 自带的全能分析工具，可以分析：内存快照、线程快照、程序死锁、监控内存的变化、gc 变化等。

5.7 线上环境CPU占用过高如何解决？

问题详细描述：生产环境上出现了OutOfMemoryError,伴随着这个问题随之而来的是Full GC，CPU 百分之百，频繁宕机重启等问题，严重影响业务的推广及使用。想优化代码，但是代码量很大，不可能把所有的代码都筛查一遍，如何定位问题？

```
ERROR [stderr] (RMI RenewClean-
[net.sf.ehcache.distribution.ConfigurableRMIClientSocketFactory@1d4c0])
Exception in thread "RMI RenewClean-
[net.sf.ehcache.distribution.ConfigurableRMIClientSocketFactory@1d4c0]"
java.lang.OutOfMemoryError: Java heap space
```

1. 重启服务
2. 通过JDK自带jvisualvm监控工具故障处理分析问题。
3. 通过堆页面查找保留大小最大的对象（前20的堆中的对象）。
4. 重点关注前几个对象
5. 分析问题，通过工具查询到sql或代码。
6. 对sql语句进行分析优化。
7. 重新发布版本。