# **JVM**

# 0开篇

方法论

1. 学习优秀的人（从人学习）

缺点：不是自亲身经历，不够深刻

1. 有输入，有输出

做笔记，思维导图，做项目，分享给别人（这点最好）

# 1JVM介绍和HotSpot机讲解

# 2垃圾收集器

# 3类加载机制

类加载五步骤

1. 类型加载：查找并加载类的二进制数据。从磁盘上加载字节码文件至内存中
2. 连接：将不同类关联起来，确定连接关系

2.1 验证：确保被加载的类的正确性

2.2 准备：为累的静态变量分配内存，并将其初始化为默认值

2.3 解析： 把勒种的符号引用（间接引用）转换为直接引用

1. 初始化：为类的静态变量赋予正确的初始值
2. 使用
3. 卸载

加载，连接，初始化都是运行期完成的

如下几种情况下，java虚拟机将结束生命周期：

1. 执行了System.exit();
2. 程勋正常执行结束
3. 程序在执行过程中遇到了异常或错误而异常终止
4. 由于操作系统出现错误而导致Java虚拟机进程终止

Java程序对类的使用方式分为两种

1. 主动使用（七种情形）

1.1创建类的实例

1.2访问某个类或接口的静态变量，或者对该静态变量赋值

1.3调用类的静态方法

1.4 反射（如Class.forName(“”)）

1.5 初始化一个类的子类。初始化子类会初始化父类

1.6 Java虚拟机启动时被标明启动类的类（Java Test）

1.7 JDK1.7 开始提供的动态语言支持：java.lang.invoke.MethodHandle实例解析结果REF\_getStatic,REF\_pubStatic,REF\_invokeStatic句柄对应的类没有初始化，则初始化

除了以上其中情况，其他使用Java类的方式都被看做是对类的被动使用，都不会导致类的初始化

1. 被动使用

所有的Java虚拟机实现必须在每个类或接口被Java程序“首次主动使用”时才初始化他们

类的加载：类的加载指的是将类.class文件中的二进制数据读入到内存中，将其放在运行时数据区的方法区内，然后再内存中创建一个java.lang.Class对象（规范并未说明Class对象位于哪里，HotSpot虚拟机将其放在了方法区中）用来封装类在方法区内的数据结构

加载.class文件的方式

1. 从本地系统中直接加载
2. 通过网络下载.class问件
3. 从zip，jar等归档文件中加载.class文件
4. 从专有数据库中提取.class文件
5. 将java源文件动态编译为.class文件

代码例子1

|  |
| --- |
| package com.gardenia.jvm.classloader;  public class Mytest1 {   public static void main(String[] args) {  */\*\*  \* 执行结果  \* MyParent1 static block  \* hello word  \*/* System.*out*.println(MyChild1.*str*);  } }  class MyParent1 {  public static String *str* = "hello word";  static {  System.*out*.println("MyParent1 static block ");  } }  class MyChild1 extends MyParent1 {  static {  System.*out*.println("MyChild1 static block");  } }  上述代码，访问父类的str字段，输出父类静态代码块，父类静态字段，而子类完全没有初始化动作，说明：  *1.对于静态字段来说，只有指定定义了该字段的类才会被初始化 2.当一个雷在初始化时，要求其父类全部都已经初始化完毕了 3.子类MyChild1类被加载*  启动-XX:+TraceClassLoading 用与追踪类的加载信息    代码吗块二  在子类中增加静态属性str2，并且访问它  package com.gardenia.jvm.classloader;  public class Mytest1 {  public static void main(String[] args) {  */\*\*  \* 执行结果  \* MyParent1 static block   \* MyChild1 static block  \* welcome  \*/*  System.*out*.println(MyChild1.*str2*);  } }  class MyParent1 {  public static String *str* = "hello word";  static {  System.*out*.println("MyParent1 static block ");  } }  class MyChild1 extends MyParent1 {  public static String *str2* = "welcome";  static {  System.*out*.println("MyChild1 static block");  } }  上述：说明。访问子类，父类先初始化。 |

虚拟机设置启动参数的格式

-XX:+<option> 表示开启option选项

-XX:-<option> 表示关闭option选项

-XX:<option>=<value> 表示option选项的值设置为value

代码例子2

|  |
| --- |
| 片段1  package com.gardenia.jvm.classloader;  public class MyTest2 {  public static void main(String[] args) {  System.*out*.println(MyParent2.*str*);  }  }  class MyParent2{  public static String *str* = "hello world";  static{  System.*out*.println("MyParent2 static block");  } }  结果：    片段2  package com.gardenia.jvm.classloader;  public class MyTest2 {  public static void main(String[] args) {  System.*out*.println(MyParent2.*str*);  }  }  class MyParent2{  public static final String *str* = "hello world";  static{  System.*out*.println("MyParent2 static block");  } }  结果：    片段一和片段二是差别是字段str一个被final修饰，  final修饰：常量在编译阶段会存入到调用这个常量的方法所在的类的常量池中，本质上，调用类并没有直接引用到定义常量的类，因此并不会触发定义常量的类的初始化  注意：这里是指将常量存放到了MyTest2的常量池中，之后的MyTest2与MyParent2就没有任何关系了；甚至，将MyParent2的class文件删除 |

助记符：

ldc表示将int，float或是String类型的常量值从常量池中推送至栈顶



|  |
| --- |
| 片段1  package com.gardenia.jvm.classloader;  public class MyTest2 {  public static void main(String[] args) {  System.*out*.println(MyParent2.*s*);  }  }  class MyParent2{  public static final String *str* = "hello world";  public static final short *s* = 7;  static{  System.*out*.println("MyParent2 static block");  } }  bipush:表示将单字节(-128到127)的常量值推送至栈顶    片段2  package com.gardenia.jvm.classloader;  public class MyTest2 {  public static void main(String[] args) {  System.*out*.println(MyParent2.*i*);  }  }  class MyParent2{  public static final String *str* = "hello world";  public static final short *s* = 7;  public static final int *i* = 128;  static{  System.*out*.println("MyParent2 static block");  } }  sipush:将一个短整型常量值(-32768到32767)推送至栈顶    片段3  package com.gardenia.jvm.classloader;  public class MyTest2 {  public static void main(String[] args) {  System.*out*.println(MyParent2.*m*);  }  }  class MyParent2{  public static final String *str* = "hello world";  public static final short *s* = 7;  public static final int *i* = 128;  public static final int *m* = 1;  static{  System.*out*.println("MyParent2 static block");  } }  iconst\_1:表示短整型的iconst\_1-iconst\_5 （1-5）数字6开始使用bipush |

# 4线程与锁

# 5JIT与GC日志生成与分析

# 6虚拟机监控工具等

# 7其他知识点