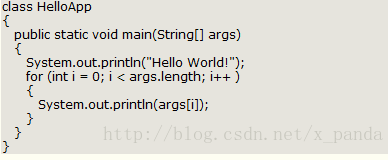


JVM有两种机制:

1，类装载子系统：装载具有适合名称的类或接口，装载在JVM的内存空间，JVM的内存空间由方法区、Java堆、Java栈、本地方法栈、指令计数器及其他隐含寄存器组成。

2，执行引擎：负责执行包含在已装载的类或接口中的指令。

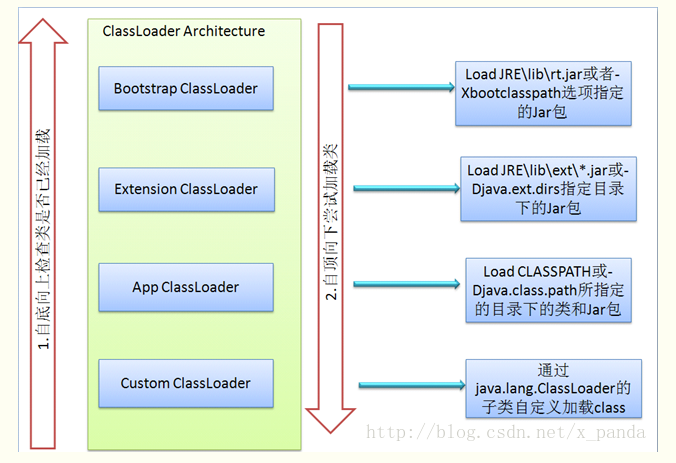
以下以以下代码为例：



编译后在命令行模式下键入： java HelloApp run virtual machine

将通过调用HelloApp的方法main来启动java虚拟机

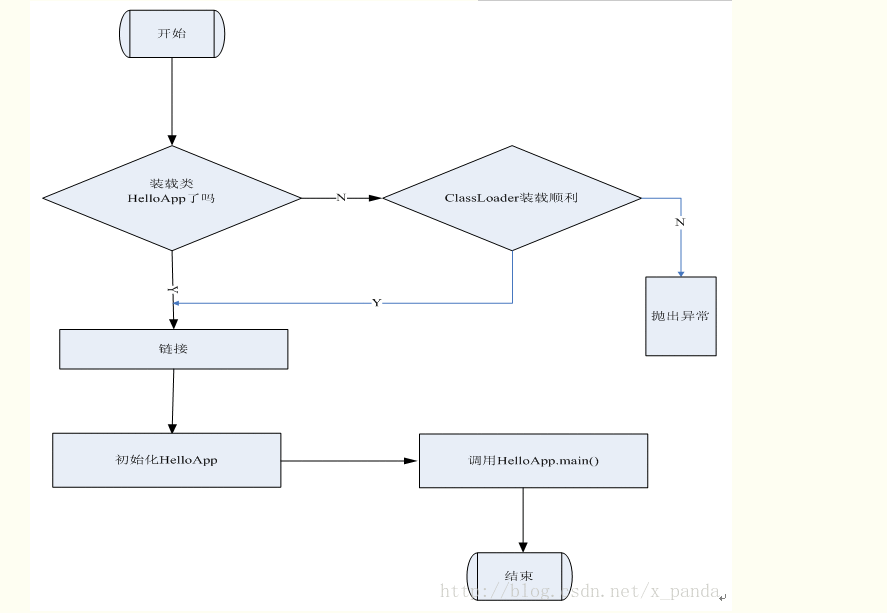
首先进入类加载子系统：



加载过程中会先检查类是否被已加载，检查顺序是自底向上，从Custom ClassLoader到BootStrap ClassLoader逐层检查，只要某个classloader已加载就视为已加载此类，保证此类只所有ClassLoader加载一次。而加载的顺序是自顶向下，也就是由上层来逐层尝试加载此类。

类执行：

开始试图执行类HelloApp的main方法，发现该类并没有被装载，也就是说虚拟机当前不包含该类的二进制代表，于是虚拟机使用ClassLoader试图寻找这样的二进制代表。如果这个进程失败，则抛出一个异常。类被装载后同时在main方法被调用之前，必须对类HelloApp与其它类型进行链接然后初始化。链接包含三个阶段：检验，准备和解析。检验检查被装载的主类的符号和语义，准备则创建类或接口的静态域以及把这些域初始化为标准的默认值，解析负责检查主类对其它类或接口的符号引用，在这一步它是可选的。类的初始化是对类中声明的静态初始化函数和静态域的初始化构造方法的执行。一个类在初始化之前它的父类必须被初始化。整个过程如下：



在举一个例子来说明到底内存的分配情况：

public class Demo01 {

public static void main(String[] args) {

A a = new A();

System.out.println(a.width);

}

}

class A{

public static int width=100; //静态变量，静态域 field

static{

System.out.println("静态初始化类A");

width = 300 ;

}

public A() {

System.out.println("创建A类的对象");

}

}

