知识就是这样，你想搞懂一个东西，你必须先把它之前的东西都给整明白。

所以，先说一下创造并执行java应用的步骤(来自书本：*Java™ How to Program, Tenth Edition,Early Objects，ISBN-10: 0-13-380780-0*

*ISBN-13: 978-0-13-380780-6)*

1. ***Creating（editing） a Program***
2. ***Compiling a Java Program into Bytecodes***
3. ***Loading a Program into Memory***

In Phase 3, the JVM places the program in memory to execute it—this is known as **loading***.*The JVM’s **class loader** takes the .class files containing the program’s bytecodes

and transfers them to primary memory. It also loads any of the .class files provided

by Java that your program uses. The .class files can be loaded from a disk on your system

or over a network (e.g., your local college or company network, or the Internet).

4.0 ***Bytecode Verification***

***Why is verification necessary?***

https://docs.oracle.com/javase/specs/jvms/se8/html/jvms-4.html#jvms-4.10

1. ***Execution（problems may occur during this time）***

再说一下一些基本概念（来自网站：http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/G1GettingStarted/index.html）：

**JRE：**

When you download Java, you get the **Java Runtime Environment (JRE)**. The **JRE** consists of the Java Virtual Machine (JVM), Java platform core classes, and supporting Java platform libraries. All three are required to run Java applications on your computer. With Java 7, Java applications run as desktop applications from the operating system, as a desktop application but installed from the Web using Java Web Start, or as a Web Embedded application in a browser (using JavaFX).

**Java Programming Language**

Java is an object-oriented programming language that includes the following features.

* Platform Independence - Java applications are compiled into *bytecode* which is stored in class files and loaded in a JVM. Since applications run in a JVM, they can be run on many different operating systems and devices.
* Object-Oriented - Java is an object-oriented language that take many of the features of C and C++ and improves upon them.
* Automatic Garbage Collection - Java automatically allocates and deallocates memory so programs are not burdened with that task.
* Rich Standard Library - Java includes a vast number of premade objects that can be used to perform such tasks as input/output, networking, and date manipulation.

**Java Development Kit**

The **Java Development Kit (JDK)** is a collection of **tools** for developing Java applications. With the JDK, you can compile programs written in the Java Programming language and run them in a JVM. In addition, the JDK provides tools for packaging and distributing your applications.

The JDK and the JRE share the Java Application Programming Interfaces ([Java API](http://docs.oracle.com/javase/7/docs/api/)). The Java API is a collection of prepackaged libraries developers use to create Java applications. The Java API makes development easier by providing the tools to complete many common programming tasks including string manipulation, date/time processing, networking, and implementing data structures (e.g., lists, maps, stacks, and queues).

**Java Virtual Machine**

The Java Virtual Machine (JVM) is an abstract computing machine. The JVM is a program that looks like a machine to the programs written to execute in it. This way, Java programs are written to the same set of interfaces and libraries. Each JVM implementation for a specific operating system, translates the Java programming instructions into instructions and commands that run on the local operating system. This way, **Java programs achieve platform independence.**

The first prototype implementation of the Java virtual machine, done at Sun Microsystems, Inc., emulated the Java virtual machine instruction set in software hosted by a handheld device that resembled a contemporary Personal Digital Assistant (PDA). Oracle's current implementations emulate the Java virtual machine on **mobile, desktop and server devices**, but the Java virtual machine does not assume any particular implementation technology, host hardware, or host operating system. It is not inherently interpreted, but can just as well be implemented by compiling its instruction set to that of a silicon CPU. It may also be implemented in microcode or directly in silicon.

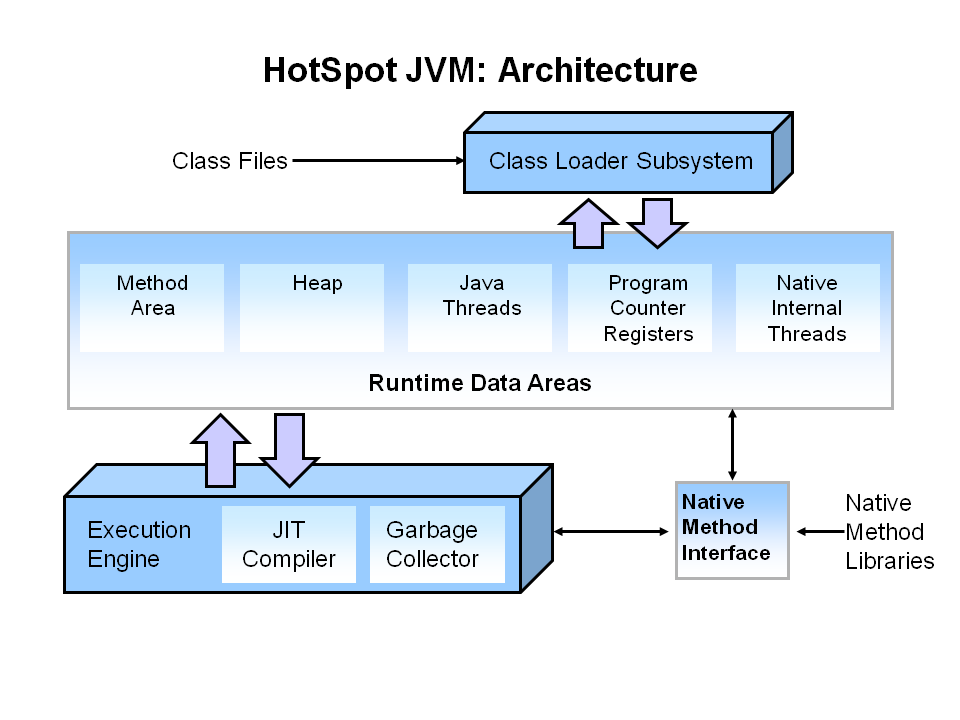
The Java virtual machine knows **nothing** of the Java programming language, only of a particular binary format, the class file format. A class file contains Java virtual machine instructions (or **bytecodes**) and a symbol table, as well as other ancillary information.

For the sake of security, the Java virtual machine imposes strong syntactic and structural constraints on the code in a class file. However, any language with functionality that can be expressed in terms of a valid class file can be hosted by the Java virtual machine. Attracted by a generally available, machine-independent platform, **implementors of other languages** can turn to the Java virtual machine as a delivery vehicle for their languages. (1)[The Java Virtual Machine](http://docs.oracle.com/javase/specs/jvms/se7/html/jvms-1.html)

再来看一下JVM的构造：

**Hotspot Architecture**

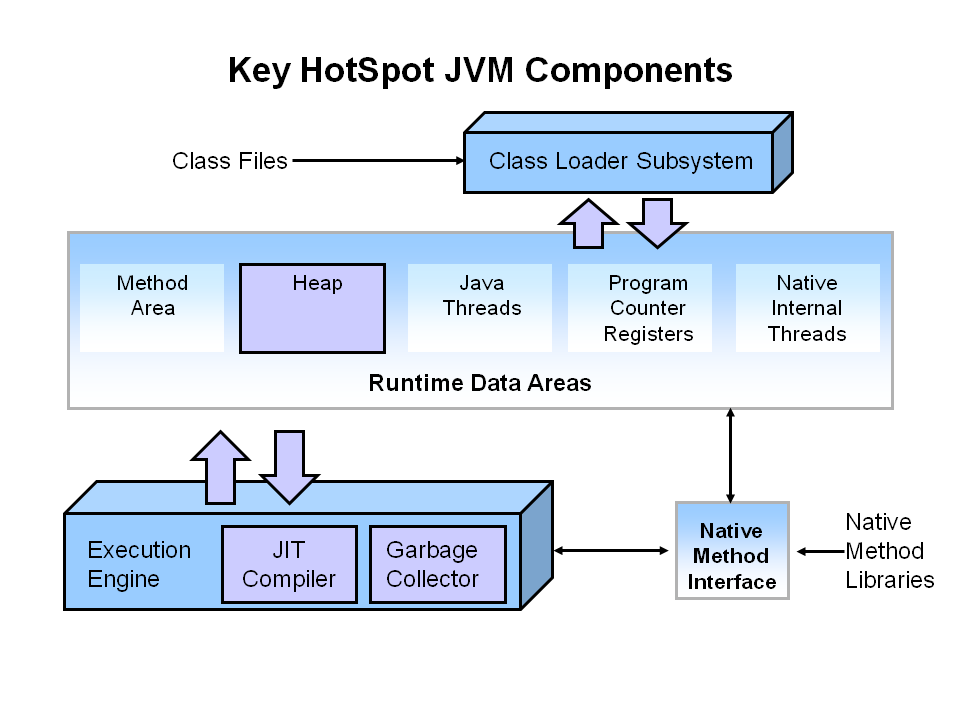
The HotSpot JVM possesses an architecture that supports a strong foundation of features and capabilities and supports the ability to realize **high performance** and massive scalability. For example, the HotSpot JVM JIT compilers generate **dynamic optimizations**. In other words, they make optimization decisions while the Java application is running and generate high-performing native machine instructions targeted for the underlying system architecture. In addition, through the maturing evolution and **continuous engineering** of its runtime environment and **multithreaded** garbage collector, the HotSpot JVM yields high scalability on even the largest available computer systems.



The main components of the JVM include the class loader, the runtime data areas, and the execution engine.

**Key Hotspot Components**

The key components of the JVM that relate to performance are highlighted in the following image.



There are three components of the JVM that are focused on when **tuning** （if you are tuning a guitar, you are trying to make its sound better）performance. The *heap* is where your object data is stored. This area is then managed by the garbage collector selected at startup. Most tuning options relate to sizing the heap and choosing the most appropriate garbage collector for your situation. The JIT compiler also has a big impact on performance but rarely requires tuning with the newer versions of the JVM.

在这上面有注意到有一个比较特殊的地方，就是在runtime area里面，有一个叫Method Area的地方

以下为解释（来自《Inside the Java Virtual Machine 2nd Edition》 作者：Bill Venners ）：

Inside a Java Virtual Machine instance, information about **loaded types** is stored in a **logical** area of memory called the method area. When the Java Virtual Machine loads a type, it uses a class loader to locate the appropriate class file. The class loader reads in the class file--a linear stream of binary data--and passes it to the virtual machine. The virtual machine extracts information about the type from the binary data and stores the information in the method area. Memory for class (static) variables declared in the class is also taken from the method area.

All threads **share** the **same** method area, so access to the method areaís data structures must be designed to be thread-safe. If two threads are attempting to find a class named Lava, for example, and Lava has not yet been loaded, only one thread should be allowed to load it while the other one waits. （线程共享）

The method area can also be garbage collected. Because Java programs can be dynamically extended via class loader objects, classes can become "unreferenced" by the application. If a class becomes unreferenced, a Java Virtual Machine can unload the class (garbage collect it) to keep the memory occupied by the method area at a minimum. The unloading of classes--including the conditions under which a class can become "unreferenced"--is described in Chapter 7, "The Lifetime of a Class." （垃圾收集）

The size of the method area need not be fixed. As the Java application runs, the virtual machine can expand and contract the method area to fit the applicationís needs. Also, the memory of the method area need not be contiguous. It could be allocated on a heap--even on the virtual machineís own heap. Implementations may allow users or programmers to specify an initial size for the method area, as well as a maximum or minimum size. （可设置容量大小）

它被描述为logical area ，更准确的说，它应该是von Neumann设计中heap的一部分（从目前来看，我觉得它更像data了，在询问老师后，老师回答了一个过时的内容，就是说 他说static area(data and methods)is a subpart of heap called PermGen,然后永久代在java8的时候已经移除了，详见https://www.baidu.com/link?url=jzs3zqOfJCwqdMf0TnQe6okuvHkgRkXOLVGcin\_S5gHcUWyJAdRsBRYcyToLCgyqyzJq1326sE0zmMEeb6eyTq&wd=&eqid=e51f35f300029aeb000000035adef22c）。它有堆的性质，它只有一个，而且被所有线程共享，可以被垃圾收集，可设置容量大小

当JVM加载了一个class文件后，则class中的参数、类型等信息会存储在方法区中。程序运行时所有创建的对象存储在堆中。

The virtual machine will search through and use the **type** information stored in the method area as it executes the application it is hosting.

再稍微看一下栈和堆分别存了啥（来自https://blog.csdn.net/HoldonWithYourGoal/article/details/54924014）

栈：

1、一个线程有一个栈区，栈与栈之间的数据不共享；

        2、用于存基础数据类型和对象的引用； 3、栈分为3个部分：基本类型变量区、执行环境上下文、操作指令区(存放操作指令)。

★方法调用栈：每一个线程都有一个方法调用栈，用于跟踪线程执行过程中方法的调用过程。栈中的每一个元素称为一个栈帧，线程每执行一个方法，就向方法调用栈压入一个新的栈帧。栈帧中存的是该方法的参数、局部变量和临时变量。

堆：

1、JVM只有一个堆区，被所有线程共享；

2、用于存对象（存的全部都是对象，不是基本数据类型和对象引用，是对象本身）；

（所有的对象实例及数组都在堆上进行分配）

3、每个对象都包含一个与之对应的class的信息。(class的目的是得到操作指令)

**永久代**

总结：记这么多（虽然不是很多）是不现实的，重要的是体会von Neumann的思想，他们无论如何都是一个基本纲领的（跟着党走，发展有中国特色的社会主义之类的 :p

再分享一个看起来还行的链接：<https://blog.csdn.net/zhangqiluGrubby/article/details/59110906>

更改：

永久代是其他虚拟机的东西，hotspot在java8之后将取消永久代。