前端编译器 ：java🡪class

后端运行期编译器即JIT编译器：class🡪机器码

静态提前编译器：java🡪机器码

虚拟机设计团队将对性能的优化集中到了后端的即时编译器中。

Javac编译器

Javac编译器不像HotSpot虚拟机那样使用C++语言（包含少量C语言）实现，它本身就是一个由java语言编写的程序。

编译过程大致可以分为3个过程：

1. 解析与填充符号表的过程
2. 插入式注解处理器的注解处理过程
3. 分析与字节码生成过程

解析与填充符号表

1. 词法、语法分析

词法分析是将原代码的字符流转变为标记（TOKEN）的集合，单个字符是程序编写过程的最小元素，而标记则是编译过程的最小元素。关键字、变量名、字面量和运算符等都可以成为标记。

语法分析是根据Token序列构造抽象语法树的过程。抽象语法树（AST）是一种用来描述程序代码语法结构的树形表示形式。语法树的每一个节点都代表着程序代码中的一个语法结构，例如包、类型、修饰符、运算符、接口、返回值甚至代码注释都是一个语法结构。

1. 填充符号表

符号表是由一组符号地址和符号信息构成的表格。符号表中所登记的信息在编译的不同阶段都要用到。

在语义分析中，符号表所登记的内容将用于语义检查（如检查一个名字的使用和原先的说明是否一致）和产生中间代码。

在目标代码生成阶段，当对符号名进行地址分配时，符号表时地址分配的依据。

注解处理器

注解与普通的java代码一样，是在运行期发挥作用的

如果这些插入式注解处理器在处理注解期间对语法树进行了修改，编译器将回到解析即填充符号表的过程重新处理，直到所有插入式注解处理器都没有再对语法树进行修改为止。

插入式注解处理器可以访问语法树的任意元素，程序员可以使用插入式注解处理器来实现许多原本只能在编码中完成的事情。

语义分析与字节码生成

语法树能够表示一个结构正确的源程序的抽象，但无法保证源程序是符合逻辑的。

语义分析的主要任务是对结构山正确的源程序进行上下文有关性质的审查，如类型审查。

1. 标注检查

检查的内容包括诸如变量使用前是否已被声明，变量与赋值之间的数据类型是否能够匹配等。

还有一个重要的动作——常量折叠

1. 数据及控制流分析

是对程序上下文逻辑更进一步的验证

编译时期的数据及控制流与类加载时的数据及控制流分析的目的基本上是一致的，但校验范围有所差别，有一些校验项只有在编译期或运行期才能进行。

1. 解语法糖
2. 字节码生成

是编译过程的最后一个阶段

Java语法糖的味道

泛型和类型擦除

泛型本质是参数化类型的应用，也就是说所操作的数据类型被指定为一个参数，这种参数可以用在类、接口和方法的创建中，分别称为泛型类、泛型接口和泛型方法。

所以对于运行期的java语言来说，ArrayList<integer>与ArrayList<String>就是同一个类。所以泛型技术实际上是java语言的一颗语法糖。

类型擦除

伪泛型

自动装箱、拆箱与遍历循环

自动装箱和拆箱在编译之后被转换成了对应的包装和还原方法

而遍历循环则把代码还原成了迭代器的实现，这也是为何遍历循环需要被遍历的类实现Iterable接口的原因。

变长参数在调用的时候变成了一个数组类型的参数

条件编译

类似其他语言中的预处理器指示符来完成条件编译。

Java语言当然也可以进行条件编译，方法就是使用条件为常量的if语句。

Java还有很多其他的语法糖，如内部类、枚举类、断言语句、对枚举和字符串的switch支持、try语句中定义和关闭资源等。