1. 前端编译
2. 编译分类

Java的编译有三种，一种是前端编译器，将java文件转变为Class文件，如JDK的Javac；一种是Java虚拟机的即时编译器（JIT，Just In Time），在运行期将字节码转变为本地机器码，如HotSpot虚拟机的C1，C2编译器；一种是静态的提前编译器（AOT，Ahead Of Time Compiler），直接把程序编译为与目标机器指令集相关的二进制代码，如JDK的Jaotc。

前端编译器中几乎没有优化措施，Java虚拟机将性能的优化集中到运行期的即时编译器中，这样可以让不是由Javac产生的Class文件也同样能够享受到编译器优化措施带来的性能提升。前端编译器在编译期的优化只能让开发者的编码效率提高。

1. Javac编译器

编译过程分为1个准备过程和3个处理过程

1. 准备过程

初始化插入式注解处理器。

1. 解析与填充符号表过程
2. 词法分析，语法分析

词法分析是将源代码的字符流转变为标记集合的过程，单个字符是程序编写的最小元素，标记是编译时的最小元素。如“int a = b + 2”包含了6个标记，不可再拆分。

语法分析时根据标记序列构造抽象语法树的过程，抽象语法树（AST, Abstract Syntax Tree）是一种用来描述程序代码语法结构的树形表示方式，树中的每一个节点都代表着代码中的一个语法结构，如包，类型，修饰符等等。

经过词法，语法分析生成语法树后，编译器就不会再对源码字符流进行操作了，后续的操作都建立在抽象语法树上。

1. 填充符号表

符号表是一组符号地址和符号信息构成的数据结构，可以由哈希表实现，在该表中登记的信息在编译的不同阶段都要用到，如在语义分析阶段进行语义检查和产生中间代码。

1. 插入式注解处理器的注解处理过程

JDK5之后，Java提供了对注解的支持，JDK6又提出了“插入式注解处理器”的标准API，可以在编译期对代码中的特定注解进行处理，从而影响到前端编译器的工作过程。插入式注解处理器可以看做是编译器的插件，可以读取，修改，添加抽象语法树中的任意元素，如果这些插件修改了语法树，编译器将回到解析和填充符号表的过程重新处理，直到所有插入式注解处理器无法对语法树修改，每一次循环称为一个轮次。

开发者可以通过编译器注解处理这个API干涉编译器的行为，语法树中的任意元素甚至是代码注释都可以在插件中被访问到，因此插入式注解处理器可以有很大的发挥空间，如编码效率工具Lombok，通过注解来实现自动产生getter/setter方法。

1. 语义分析与字节码生成过程
2. 语义分析概述

抽象语法书能够表示一个结构正确的源程序，但是无法保证源程序的语义是符合逻辑的，通过语义分析可以对源程序进行上下文相关性质的检查，如类型检查，控制流检查，数据流检查等等。所谓的语义检查指的是是否符合某个具体语言的语言规范，因此语义检查是与具体的语言和具体的上下文环境紧密联系的，我们在编码时常看到的红线标注错误提示就是语义分析的检查结果。

语义分析过程又分为标注检查和数据及控制流分析。

1. 标注检查

标注检查要检查的内容包括变量使用前是否已经被声明了，变量与赋值之间的数据类型是否能够匹配等等，还有常量折叠的代码优化，如将“int a = 1 + 2”优化为字面量“3”。

1. 数据及控制流分析

对程序上下文逻辑的进一步验证，检查出如局部变量使用前是否赋值，方法的每条路径是否有返回值等等。

1. 解语法糖

语法糖是一种在语言中添加的某种语法，这种语法对于语言的编译结果和功能没有实际影响，但是能够方便开发者使用该语言，减少代码量。Java属于低糖语言，即比较啰嗦的语言，Java中的语言糖包括泛型，变长参数，自动装箱拆箱等。但是Java虚拟机运行时不支持这些语法的，它们在编译阶段被还原为原始的基础语法结构，这个过程称为解语法糖。

1. 字节码生成

字节码生成是Javac编译过程的最后一个阶段，将前面各个步骤生成的信息（语法树，符号表）转换为字节码指令写入到磁盘中，还进行少量的代码添加和转换工作。如实例构造器<init>()方法和类构造器<clinit>()方法就是在这个阶段被添加到语法树中的，完成对语法树的遍历和调整之后，就会把填充了所有所需信息的符号表交到ClassWriter类中，由这个类的writeClass()方法输出字节码，生成最终的Class文件。