为语言添加优化器支持，以及添加JIT编译器支持

1. 添加优化器支持

常量折叠是其中一种被很多现代编译器使用的编译器优化技术，直接的说常量折叠就是在编译时期用常量的具体取值代替所有用到常量的地方，但是编译器会为常量分配地址,只是在使用常量的时候,常量会被数替换。在AST中支持constant folding常量折叠的优化。在LLVM中不需要此支持。在LLVM IR的代码生成过程中，会通过LLVM IR构建器，构建器会在调用过程中检查是否存在constant folding。优化过程调用LLVM的Pass，所以说这些优化是通用的。因为这些优化都是与高级语言无关的，是对LLVM IR 的优化。

为了实现优化，设置一个新的FunctionPassManager来保存和组织想要运行的LLVM优化，因此编写InitializeModuleAndPassManager（）函数来创建和初始化模块和传递管理器。

1. 添加JIT编译器

JIT是即时编译编译器，而使用即时编译器技术，能够加速 Java 程序的执行速度。JIT的编译过程为当JIT编译启用时，JVM 读入.class 文件解释后，将其发给JIT编译器，JIT编译器将字节码编译成本机机器代码，在运行时JIT会把翻译过的机器码保存起来，以备下次使用

声明和初始化JIT：通过调用一些InitializeNativeTarget函数并添加一个全局变量TheJIT并在main函数中初始化它。还需要在InitializeModuleAndPassManager（）函数中为JIT设置data layout。JIT的API非常简单：addModule将一个LLVM IR模块添加到JIT；removeModule删除模块，释放与该模块中代码关联的任何内存。利用这些修改top-level expressions中的代码，若成功，则将包含top-level expressions的模块添加到JIT，addModule触发模块中的所有函数的代码生成，并返回一个句柄，以便删除。由于加入到JIT的模块的代码不能修改，因此调用InitializeModuleAndPassManager（）来保存后续代码。通过调用JIT的findSymbol方法并传递顶级表达式函数的名称来获得指向最终生成代码的指针。接下来调用getAddress()符号来获取函数在内存中的地址。为了解决删除的范围问题，使每个函数都存在于自己的模块中：首先添加一个新的全局变量FunctionProtos，它包含每个函数的最新原型。添加一个getFunction()来替换TheModule->getFunction()。还需要更新HandleDefinition和HandleExtern：将新定义的函数传递给JIT并打开一个新模块。在HandleExtern中，将原型添加到FunctionProtos中。并删除匿名函数的转储。