它的 def-use 链提供了简单方法。对于每个 Value 类，还可以通过 getName() 方法访问其名称。这符合任何 LLVM 变量都可以具有与其相关的明确标识这一情况。例如，%add1 可以定义一个加法指令的结果，BB1 可以定义一个基本块，myfunc 可以标识一个函数。Value 也有一个强大的方法，称为 replaceAllUsesWith(Value*)，它可以遍历该值的所有使用者，并用其他值取代它。这是一个很好的 SSA 表示形式的例子，使得你可以轻松地替换指令，并快速编写优化。该类的完整接口可以通过如下地址查看：

- http://llvm.org/docs/doxygen/html/classllvm_1_1Value.html

- User 类具有 op_begin() 和 op_end() 方法，它们允许快速访问它使用的所有 Value 接口。请注意，这个关系对应的是 use-def 链。用户也可以使用名为 replaceUsesOfWith(Value *From,Value *To) 的方法来替换它使用的任何值。该类的完整接口可以通过如下地址查看：
  - http://llvm.org/docs/doxygen/html/classllvm_1_1User.html

5.4 编写自定义的 LLVM IR 生成器

可以使用 LLVM IR 生成器 API 以编程方式为 sum.ll 构建 IR（在 -O0 优化级别创建，即不进行优化）。在这一节，我们将逐步阐述该过程。首先来看所需的头文件：

- #include <llvm/ADT/SmallVector.h>：该头文件定义模板类 SmallVector<>，当元素数量不大时，可以构建高效的向量数据结构。请参阅 http://llvm.org/docs/ProgrammersManual.html 以获取关于 LLVM 数据结构的帮助。
- #include <llvm/Analysis/Verifier.h>：它提供一个重要的程序验证分析，即检查当前的 LLVM 模块是否与 IR 语法规则相符。
- #include <llvm/IR/BasicBlock.h>：它是声明 BasicBlock 类的头文件，该类是我们已经介绍过的重要的 IR 实体。
- #include <llvm/IR/CallingConv.h>：这个头文件定义在函数调用中使用的一组 ABI 规则，比如函数参数的存储位置。
- #include <llvm/IR/Function.h>：这个头文件声明 Function 类，也是一个重要的 IR 实体。
- #include <llvm/IR/Instructions.h>：这个头文件声明 Instruction 类的所有子类，这是 IR 的基本数据结构。
- #include <llvm/IR/LLVMContext.h>：这个头文件存储 LLVM 库的所有全局作用范围的数据，它允许多线程版本在每个线程中使用不同的上下文。
- #include <llvm/IR/Module.h>：这个头文件声明 Module 类，它是 IR 层次结构中的顶层实体。
- #include <llvm/Bitcode/ReaderWriter.h>：这个头文件包含用于读取和写入 LLVM 位码文件的代码。