## 选型结论

Go语言虽然在执行效率和内存消耗上略逊于C++（最差为C++的一半）， 但它具有更加简洁、安全的语法，并且在语言级别上支持并发编程。同时提供了丰富的工具库用于观察内存和CPU的使用状况，以及垃圾回收的执行频率和占用时间，大幅度提高了开发效率并降低了维护成本。

**所以建议使用Go语言替换C++进行IM Core的开发。**

## Go语言特性

自动垃圾回收，更丰富的内置类型（如map），函数多返回值，匿名函数和闭包，非侵入式的接口，并发编程，语言交互性（支持与C混合编程）

## Go语言特性与C++和Java的对比

|  |  |  |
| --- | --- | --- |
| **C++** | **Java** | **Go** |
| 除了一些比较少见的情况之外和[C语言](http://zh.wikipedia.org/wiki/Cèªè¨)兼容 | 没有对任何之前的语言向前兼容。但在语法上受 C/C++ 的影响很大 | 语法上受 C语言的影响很大，但又有许多新的语法特性（如函数多返回值） |
| 一次编写多处编译 | 一次编写多处运行 | 一次编写多处编译 |
| 允许[过程式程序设计](http://zh.wikipedia.org/wiki/ç¨åºå¼ç¨å¼è¨­è¨)和[面向对象程序设计](http://zh.wikipedia.org/wiki/é¢åå¯¹è±¡ç¨åºè®¾è®¡) | 鼓励(或者说必须)面向对象的程序设计方式 | 可以面向过程编程、面向对象编程、也可以函数式编程 |
| 允许直接调用本地的系统库 | 要通过[JNI](http://zh.wikipedia.org/wiki/Java_Native_Interface)调用, 或者 [JNA](http://zh.wikipedia.org/w/index.php?title=Java_Native_Access&action=edit&redlink=1) | 可以直接调用系统库 |
| 能直接使用底层系统接口 | 在一个保护模式下的虚拟机中运行 | 能直接使用底层系统接口 |
| 只提供对象的类型和类型名 | 是[反射的](http://zh.wikipedia.org/wiki/åå°_(è®¡ç®æºç§å­¦)), 允许[元程序设计](http://zh.wikipedia.org/w/index.php?title=%E5%85%83%E7%A8%8B%E5%BC%8F%E8%A8%AD%E8%A8%88&action=edit&redlink=1)和运行时的动态生成代码 | 使用语言自带的 reflect包实现反射 |
| 有多种二进制兼容标准 (例如:微软和Itanium/GNU) | 一种二进制兼容标准，允许运行时库的正确性检查 | 有多种二进制兼容标准(例如:微软和Itanium/GNU) |
| 可选的自动[边界检查](http://zh.wikipedia.org/wiki/è¾¹çæ£æ¥). (例如: vector 和 string 这两个容器的 at() 方法) | 一般都有做边界检查。[HotSpot (java)](http://zh.wikipedia.org/wiki/HotSpot_(java))(Sun 的虚拟机实现) 可以去掉边界检查 | 提供数组或数组切片的边界检查 |
| 支持本地的无符号数学运算 | 不支持本地的无符号数学运算 | 支持无符号数学运算 |
| 对所有的数字类型有标准的范围限制，但字节长度是跟实现相关的。标准化的类型可以用 typdef 定义 (uint8\_t, ..., uintptr\_t) | 在所有平台上对所有的基本类型都有标准的范围限制和字节长度 | int, uint, uintptr 等类型不能够对长度做任何假设，但其他基本类型如 uint32, int8, byte 都是固定字节长度和取值范围的 |
| 支持指针，引用，传值调用 | 基本类型总是使用传值调用。对象以可以为空的参考的方式传递（相当于在C++里使用指向 class 或者 struct参数的指针）。[[1]](http://zh.wikipedia.org/wiki/JavaåC++çå°ç§#cite_note-Java_is_Pass-By-Value-1) | 支持值语义和引用语义。大多数基本类型和复合类型都是基于值语义。但数组切片slice、map、channel以及interface是引用类型 |
| 显式的存储器管理，但有第三方的框架可以提供垃圾搜集的支持。支持析构函数。 | 自动垃圾搜集(可以手动触发)。没有析构函数的概念，对 finalize() 的使用是不推荐的 | 自动垃圾搜集(可以手动触发)。可以使用defer语句实现类似 finalize() 的功能，但语法更为简洁 |
| 支持类class，结构struct，联合union，可以在[堆栈](http://zh.wikipedia.org/wiki/å ç)或者[堆](http://zh.wikipedia.org/wiki/åæè¨æ¶é)里为它们动态分配存储器 | 只支持类，只在[堆](http://zh.wikipedia.org/wiki/åæè¨æ¶é)中为对象分配存储器。[Java SE 6](http://zh.wikipedia.org/wiki/J2SE)在栈为一些对象分配存储器的使用了[逃逸分析](http://zh.wikipedia.org/w/index.php?title=%E9%80%83%E9%80%B8%E5%88%86%E6%9E%90&action=edit&redlink=1)的优化方法 | 支持类和结构体，使用相同的关键字 struct实现 |
| 允许显式的覆盖(也叫重写)类型 | 严格的[类型安全](http://zh.wikipedia.org/wiki/ç±)，除了变宽的类型转换。Java 1.5 开始支持自动类型包装和解包装(Autoboxing/Unboxing) | 类型安全的，但也允许使用类似C语言的Unsafe 做一些不安全的类型转换工作 |
| [C++库](http://zh.wikipedia.org/w/index.php?title=C%2B%2B%E5%87%BD%E5%BC%8F%E5%BA%AB&action=edit&redlink=1)包括：语言支持，诊断工具，常用工具，字符串，本地化，容器，算法，迭代器，数值，输入/输出，C库。Boost库提供了更多的功能，包括线程和网络I/O。用户必须在一大堆（大部分互相不兼容）第三方GUI或者其他功能库中进行选择 | 库在每次 Java 发布新版本的时候都会更新并增强功能。1.6版本支持：本地化，日志系统，容器和迭代器，算法，GUI 程序设计（但没有用到系统的GUI），图形，多线程，网络，平台安全，自省机制，动态类加载，阻塞和非阻塞的I/O，对于[XML](http://zh.wikipedia.org/wiki/XML)、[XSLT](http://zh.wikipedia.org/wiki/XSLT)、[MIDI](http://zh.wikipedia.org/wiki/MIDI)也提供了相关接口或者支持类，数据库，命名服务(例如 [LDAP](http://zh.wikipedia.org/wiki/LDAP))，密码学，安全服务(例如 [Kerberos](http://zh.wikipedia.org/wiki/Kerberos))，打印服务，WEB 服务。SWT 提供了一个系统相关的GUI的抽象 | Go标准库包括：打包与压缩库、I/O库、字节库、容器库（heap, list, ring）、加解密、sql库、调试库、字符编码库、错误库、命令行标示解析库、格式化输入输出库、hash库、网络库（IP/ICMP/TCP/UDP/HTTP）、图像库、日志库、数学库、反射库、排序库、同步库（锁、原子操作）、时间库、文本处理库。GUI相关的库没有官方支持，也基本没有可靠的第三方支持。 |
| 大部分运算符可以[运算符重载](http://zh.wikipedia.org/wiki/è¿ç®ç¬¦éè½½) | 运算符的意义一般来说是不可变的，例外是 + 和 += 运算符被字符串重载了 | 运算符不可重载 |
| 完全的多重继承，包括虚拟继承 | 类只允许单继承，需要多继承的情况要使用接口 | 没有继承，只能通过struct 封装实现类似继承的效果 |
| 支持编译期模板 | [泛型](http://zh.wikipedia.org/w/index.php?title=Java%E4%B8%AD%E7%9A%84%E6%B3%9B%E5%9E%8B&action=edit&redlink=1)被用来达到和C++模板类似的效果，但由于[类型消除](http://zh.wikipedia.org/w/index.php?title=%E7%B1%BB%E5%9E%8B%E6%B6%88%E9%99%A4&action=edit&redlink=1)它们不能在编译期间从代码被编译成字节码 | 没有模板，但可以通过interface{} 类型实现类似的功能 |
| 支持函数指针，函数对象，lambda（C++11）和接口 | 没有函数指针机制。替代的概念是接口，Adapter 和 Listener也是被广泛使用的 | 支持函数指针，匿名函数，闭包 |
| 没有标准的代码内嵌文档机制。不过有第三方的软件(例如 [Doxygen](http://zh.wikipedia.org/wiki/Doxygen)) | [Javadoc](http://zh.wikipedia.org/w/index.php?title=Javadoc&action=edit&redlink=1) 标准文档生成系统 | Go doc 标准文档生成系统 |
| const 关键字用来定义不可改变的常量和成员函数 | final 提供了一个限制版本的 const，等价于 type\* const 的对象指针或者 const的基本类型数据。没有const 成员函数，也没有const type\* 指针的等价物 | const 关键字用来定义不可改变的常量 |
| 支持 goto 语句 | 支持循环标签(label)和语句块 | 支持 goto 语句，支持循环标签(label) |
| 源代码可以写成平台无关的（可以被 [Windows](http://zh.wikipedia.org/wiki/Windows)、[BSD](http://zh.wikipedia.org/wiki/BSD)、[Linux](http://zh.wikipedia.org/wiki/Linux)、[Mac OS X](http://zh.wikipedia.org/wiki/Mac_OS_X)、[Solaris](http://zh.wikipedia.org/wiki/Solaris_(æä½ç³) 等编译，不用修改），也可以写成利用平台特有的特性。通常被编译成本地的机器码 | 被编译成[Java虚拟机](http://zh.wikipedia.org/wiki/Javaèæ¬æ©)的字节码。和Java平台相关，但是源代码一般来说是不依赖[操作系统](http://zh.wikipedia.org/wiki/æä½ç³)特有的特性的 | Go代码是平台无关的（除非内嵌平台相关的C代码或汇编代码） |
| 没有语言级别的“轻量级线程” | 没有语言级别的“轻量级线程” | 原生地提供的“轻量级线程” goroutine，更良好的并发编程 |

## 与C++的执行性能的对比

## 执行速度

|  |  |  |
| --- | --- | --- |
| Bechmark | Time[sec] | Factor |
| C++ | 24 | 1.0x |
| Go | 49 | 2.04x |
| Go-Pro | 33 | 1.38x |

## 内存消耗

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Bechmark | Virt | Real | Factor Virt | Factor Real |
| C++ | 185m | 174m | 1.0x | 1.0x |
| Go | 429m | 427m | 2.32x | 2.45x |
| Go-Pro | 294m | 292m | 1.59x | 1.68x |

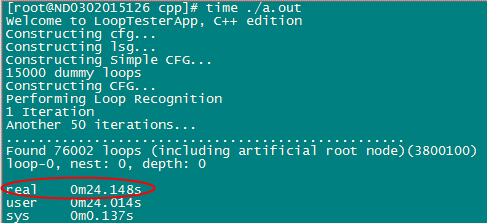
## 测试程序说明

**测试结果中 Go-Pro 是经过Google的工程师优化后的程序的测试结果。**

测试使用的是Google发布的[《循环识别的C++/Java/Go/Scala实现比较》](https://days2011.scala-lang.org/sites/days2011/files/ws3-1-Hundt.pdf) 文章中所使用的程序，源码位于(<http://code.google.com/p/multi-language-bench>)。 算法实现中使用实现语言的惯用的容器类，循环结构，以及内存/对象分配方案，并没有使用特定的语言和运行时功能，比如（线程、Go的goroutine等），使得对比近乎公平。

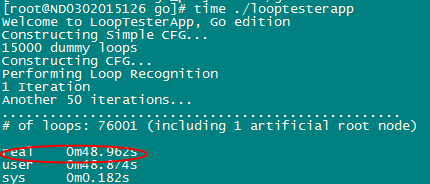
## 附：测试程序运行截图

## C++



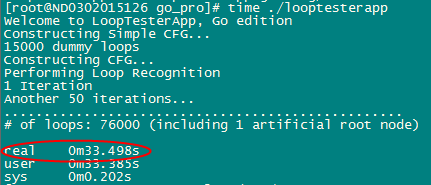


## Go





## Go-Pro





## 附：参考文献

[1] Robert Hundt, Loop Recognition in C++/Java/Go/Scala, 2011