链滴

说说 Golang 的使用心得

作者：Jazz
原文链接：https://ld246.com/article/1448438193153
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

13年上半年接触了Golang，对Golang十分喜爱。现在是2015年，离春节还有几天，从开始学到现在的一年半时间里，前前后后也用Golang写了些代码，其中包括业余时间的，也有产品项目中。一直有想法写点Golang相关的总结或者感想，决定还是在年前总结下吧。注明下：我只是Golang喜好者，不是脑残粉，也无意去挑起什么语言之争。
特性少，语法简单。GO是崇尚极简主义的，提倡少即是多。这点在它的Spec上尤其凸显，一下的时间绝对可以看完。GO的特性很少，很多GO的使用者都反馈，GO的关键字至少完全可以记在大里。同时它的语法极为简单，而且语义清晰。
部署方便。GO是一个强类型静态语言，可以把代码编译为本地机器指令。它的RUNTIME是会在译时一起链接到执行文件中，这也就意味着我们不需要像JAVA那样装一个JVM。而且编译出的执行件本身不依赖于其他动态库，完全可以做到轻松的发布。当然，如果你用GO编写了调用一些动态库口的代码，那么还是需要根据实际情况来部署这个动态库的。这点在很多从python/java转到go的朋来说，非常喜欢。
有较完善的标准库并且较为健壮。GO自身带的标准库是比较全面的，从文件归档、压缩、加密数据库到数据序列化，字符格式化、校验和以及网络库、同步库等应有尽有。基本上能够满足很多基的需求了。更好的是，这些标准库的质量都非常高，都很健壮。接口也较为简单，有清晰的文档说明同时随着这两年的发展，GO的第三方库也多了起来，虽然可能没有像python那么多，但是较其发展间来说，还是非常不错的。
集成测框架。在之前用C++写代码时，写单元测试不是个容易的工作。需要一些技巧和努力才可做起来。但是在GO中集成了单元测试框架，只要源码文件以_test.go结尾，就可以直接通过go test行单元测试。同时还提供了代码测试覆盖率工具，可以很容易实施自动化测试。除此之外，还集成了准测试框架功能，可以很容易的测量自己写的函数的运行效率。另外，还有性能剖析器，可以在运行，测试时剖析程序的瓶颈点，进而可以进行优化。
健全的代码风格与检查工具。当初学GO的时候，很多文章和书都会提到go fmt这个命令，统一代码风格。我觉着这点实在是解决了风格之争了。带来的影响就是别人写的代码感觉也是自己写的一。还有golint，可以按照go team的风格和要求来写代码。还有go vet可以用来检查一些在GO中很蔽的坑。
简单却强大的包管理。GO的包管理可能在很多其他语言的包管理看来太弱了，但是在我看来，解决了我需要的两个问题，一个是循环依赖问题,GO是拒绝有循环依赖关系的包；二是包的初始化，个包的文件都可以实现一个init函数，用来在导入时执行。这点在分工合作时非常有用。
容易编写跨平台代码。如果你用纯GO，不牵扯到CGO的话，你可以非常容易的做到跨平台。只要在文件后缀.go前，引入_linux,_windows,_x86,_x64等字符为文件名前缀的结尾就可以做到只在对的平台中编译。GO还有build constraints控制代码在什么条件下编译。如果你用到了CGO，就牵扯了C的跨平台问题，所以稍微麻烦那么一些，但是问题也不是太大。
垃圾回收。GC的存在极大的降低了并发代码的编写，而且还提供了程序的健壮性。做为一个从C/++做起，有过驱动开发经验的程序员来说，GC这个东西是我一直没有涉猎过的。对我来说GC就等于梦。但是当我开始试着接受GC时发现，GC真的是解决了程序员的生产力，极大的提高了效率。虽然前来说GO已经1.4版本了，但是GC还算不得上优秀，按照GO的路线图，后面会有更优秀的GC实现添进来。
接口与struct。在第一次学习GO的interface的时候，我第一反应是这就是我想要的。虽然很多也在说GO的这个interface的不好，而且说的很有道理，比如老赵的《为什么我不喜欢GO语言式的接》。interface可以通过组合扩展为新的interface，struct也可以通过组合扩展为新的struct。没有继，只有组合。可以通过匿名组合达到类似继承的效果。可以对interface进行查询，有点类似COM的道，但是语法层面上更为简单。struct到interface的映射是隐式的，不需要声明某个struct实现了某个nterface。虽然可能会出现名字上的冲突，但是可以通过wrapper进行解决。这种interface的另外一好处是单元测试时很容易实现MOCK，这点非常喜欢。也可以看这篇文章《Go interfaces make test tubbing easy》
统一的工作布局。GO定义了项目的目录结构，比如bin目录，pkg目录以及src目录。这个和我日的项目布局是一致的。之前用C++开发时我们也是如此安排布局，所以就这点来说，我觉着容易过渡
内建的并发原语。提到GO就不得不提到goroutine和channel。廉价的goroutine可以让我们欢的处理异步任务，channel可以用来交换数据。借助goroutine，可以很容易的实现高性能的服务端。oroutine及其调度器可以很容易和EPOLL,IOCP等系统机制结合起来，再通过Half Sync/Half Async式，很容易在语法层面上达到同步形式，却不失性能。
2013年初的时候还在做一个客户端，当时使用了C++0x，其中印象最深的事情是lambad，std::f

nction/bind，结合着线程+队列的方式，可以很容易的实现类似于Chromuim的线程模型。在处理U
与慢任务比如读文件，请求网络数据等交互时，为了保证UI的体验，使这些任务异步化是非常有必要
。如果有对Golang了解或熟悉的朋友就会明白，这是类似与Golang的goroutine+channel。但是在
用这个模型时，觉着还是繁琐了些，我要注意对象的生命周期，这个在C++虽然有各种智能指针的帮
，但是难免还会挂一漏万。而且遇到稍微的复杂的问题，比如一个慢任务接着一个慢任务时，就会涉
到一个任务链，在没有future/promise(Facebook开源的folly库中有个不错的实现futures，后来还
现WINDOWS有个基于actor model的并发库Asynchronous Agents Library非常不错，只可惜目前
在WINDOWS上使用。)机制的帮助下很容易进入Callback Hell。这样就会导致代码相对来说比较难
护，而且容易滋生BUG。好在当时这个部分的代码不是太多，而且也不是太过于复杂，很容易通过自
稳定下来。
上面虽然提到future/promise, AAL可以解决部分Callback Hell的问题，但是像future还是要用到
allback。所以我在想，如果可以做到代码层面上是同步式的，背后却是异步的就爽了。GO就满足了
这个需求。
GO标准库中还提供了sync包，其中有基本的mutex说，还有RMutex这样的读写锁，还有Once
WaiterGroup等东西。基本满足日常中对锁的需求了。
GO为了帮助程序员解决在并发时经常遇到的race condition问题，还提供了相应的race conditio
工具。还有相应的死锁检测工具。
虽然GO社区有个slogan:”do not communicate by sharing memory; instead, share
emory by communicating.“，但是每个goroutine之间并不是完全独立的，一样可以做到通
内存共享数据。这个时候只能依靠程序员自己去遵守了。而且因为goroutine不是完全独立，panic这
东西就可能会导致整个程序挂掉。这点和Erlang比起来确实不是很好。
蛋疼的defer。用习惯了C++的RAII后，十分反感GO的defer机制，但是有的时候又不得不用。
因就是这个defer不是block级别的，而是函数级别的，需要在函数返回前才得到执行。所以这就会导
在处理一些类似于文件打开，操作再关闭的逻辑时非常蛋疼，回到了C的年代，必须手动去Close。</
>
蛋疼的panic。虽然我在C++下不怎么用异常，但是对于panic这个设计我表示非常的不满意啊
因为它会影响全局。而要捕捉panic就需要用defer。如果panic只是让当前goroutine挂掉我觉着就
皮坏了。
没有泛型。GO没有泛型带来的蛋疼地方是，要么就用interface{}来做运行时泛型，要么就自己
动写代码生成器。比如我自己为了生成网络协议序列化代码就撸了一个生成器。而且因为没有泛型，
实现类似C++ STL的容器与算法基本没太可能，当然方法还是有的，继续使用代码生成器。而且GO1.
干脆引入了一个叫go generate的命令。
总结
GO里面其他一些内建的数据结构，比如slice,map等，但这些也是诟病，因为它又没给予程序员
以享用range关键字的福利。
在GO的所有特性里，最喜欢就是GC，goroutine，channel以及interface。而其余的特性（比
上面我列举的很多特性）我觉着都不是太重要，其中很多都可以在工程中实践，和语言本身没有太大
系。
总结下来，这东西就是一个工程工具，各种好用，但是从设计角度讲各种粗糙，没必要过度高估
它算的上工程实践中的好朋友。在写服务端时，它是把利器，至少在写服务端程序时，我自己感觉如
。
有朋友说一个语言好不好就看它有没有开拓你的眼界，给予你新的思想，我想至少GO在这点上
足了。