47 | 基于HTTP协议的网络服务

2018-11-28 郝林

我们在上一篇文章中简单地讨论了网络编程和socket，并由此提及了Go语言标准库中的syscall代码包和net代码包。

我还重点讲述了net.Dial函数和syscall.Socket函数的参数含义。前者间接地调用了后者，所以正确理解后者，会对用好前者有很大裨益。

之后，我们把视线转移到了net.DialTimeout函数以及它对操作超时的处理上，这又涉及了net.Dialer类型。实际上，这个类型正是net包中这两个“拨号”函数的底层实现。

我们像上一篇文章的示例代码那样用net.Dial或net.DialTimeout函数来访问基于HTTP协议的网络服务是完全没有问题的。HTTP协议是基于TCP/IP协议栈的，并且它也是一个面向普通文本的协议。

原则上，我们使用任何一个文本编辑器，都可以轻易地写出一个完整的HTTP请求报文。只要你搞清楚了请求报文的头部（header）和主体（body）应该包含的内容，这样做就会很容易。所以，在这种情况下，即便直接使用net.Dial函数，你应该也不会感觉到困难。

不过，不困难并不意味着很方便。如果我们只是访问基于HTTP协议的网络服务的话，那么使用net/http代码包中的程序实体来做，显然会更加便捷。

其中，最便捷的是使用http.Get函数。我们在调用它的时候只需要传给它一个URL就可以了，

比如像下面这样：

```
url1 := "http://google.cn"
fmt.Printf("Send request to %q with method GET ...\n", url1)
resp1, err := http.Get(url1)
if err != nil {
 fmt.Printf("request sending error: %v\n", err)
}
defer resp1.Body.Close()
line1 := resp1.Proto + " " + resp1.Status
fmt.Printf("The first line of response:\n%s\n", line1)
```

http.Get函数会返回两个结果值。第一个结果值的类型是*http.Response，它是网络服务给我们传回来的响应内容的结构化表示。

第二个结果值是error类型的，它代表了在创建和发送HTTP请求，以及接收和解析HTTP响应的过程中可能发生的错误。

http.Get函数会在内部使用缺省的HTTP客户端，并且调用它的Get方法以完成功能。这个缺省的HTTP客户端是由net/http包中的公开变量DefaultClient代表的，其类型是*http.Client。它的基本类型也是可以被拿来使用的，甚至它还是开箱即用的。下面的这两行代码：

```
var httpClient1 http.Client
resp2, err := httpClient1.Get(url1)
```

与前面的这一行代码

```
resp1, err := http.Get(url1)
```

是等价的。

http.Client是一个结构体类型，并且它包含的字段都是公开的。之所以该类型的零值仍然可用，是因为它的这些字段要么存在着相应的缺省值，要么其零值直接就可以使用，且代表着特定的含义。

现在，我问你一个问题，是关于这个类型中的最重要的一个字段的。

今天的问题是：http.Client类型中的Transport字段代表着什么？

这道题的**典型回答**是这样的。

http.Client类型中的Transport字段代表着：向网络服务发送HTTP请求，并从网络服务接收HTTP响应的操作过程。也就是说，该字段的方法RoundTrip应该实现单次HTTP事务（或者说基于HTTP协议的单次交互）需要的所有步骤。

这个字段是http.RoundTripper接口类型的，它有一个由http.DefaultTransport变量代表的缺省值（以下简称DefaultTransport）。当我们在初始化一个http.Client类型的值（以下简称Client值）的时候，如果没有显式地为该字段赋值，那么这个Client值就会直接使用DefaultTransport。

顺便说一下，http.Client类型的Timeout字段，代表的正是前面所说的单次HTTP事务的超时时间，它是time.Duration类型的。它的零值是可用的，用于表示没有设置超时时间。

问题解析

下面，我们再通过该字段的缺省值DefaultTransport，来深入地了解一下这个Transport字段。

DefaultTransport的实际类型是*http.Transport，后者即为http.RoundTripper接口的默认实现。这个类型是可以被复用的，也推荐被复用，同时，它也是并发安全的。正因为如此，http.Client类型也拥有着同样的特质。

http.Transport类型，会在内部使用一个net.Dialer类型的值（以下简称Dialer值），并且，它会把该值的Timeout字段的值，设定为30秒。

也就是说，这个Dialer值如果在30秒内还没有建立好网络连接，那么就会被判定为操作超时。在DefaultTransport的值被初始化的时候，这样的Dialer值的DialContext方法会被赋给前者的DialContext字段。

http.Transport类型还包含了很多其他的字段，其中有一些字段是关于操作超时的。

- IdleConnTimeout：含义是空闲的连接在多久之后就应该被关闭。
- DefaultTransport会把该字段的值设定为90秒。如果该值为0，那么就表示不关闭空闲的连接。注意，这样很可能会造成资源的泄露。
- ResponseHeaderTimeout：含义是，从客户端把请求完全递交给操作系统到从操作系统那里接收到响应报文头的最大时长。DefaultTransport并没有设定该字段的值。
- ExpectContinueTimeout：含义是，在客户端递交了请求报文头之后，等待接收第一个响应报文头的最长时间。在客户端想要使用HTTP的“POST”方法把一个很大的报文体发送给

服务端的时候，它可以先通过发送一个包含了“Expect: 100-continue”的请求报文头，来询问服务端是否愿意接收这个大报文体。这个字段就是用于设定在这种情况下的超时时间的。注意，如果该字段的值不大于0，那么无论多大的请求报文体都将会被立即发送出去。这样可能会造成网络资源的浪费。DefaultTransport把该字段的值设定为了1秒。

- TLSHandshakeTimeout：TLS是Transport Layer Security的缩写，可以被翻译为传输层安全。这个字段代表了基于TLS协议的连接在被建立时的握手阶段的超时时间。若该值为0，则表示对这个时间不设限。DefaultTransport把该字段的值设定为了10秒。

此外，还有一些与IdleConnTimeout相关的字段值得我们关注，
即：MaxIdleConns、MaxIdleConnsPerHost以及MaxConnsPerHost。

无论当前的http.Transport类型的值（以下简称Transport值）访问了多少个网络服务，MaxIdleConns字段都只会对空闲连接的总数做出限定。而MaxIdleConnsPerHost字段限定的则是，该Transport值访问的每一个网络服务的最大空闲连接数。

每一个网络服务都会有自己的网络地址，可能会使用不同的网络协议，对于一些HTTP请求也可能会用到代理。Transport值正是通过这三个方面的具体情况，来鉴别不同的网络服务的。

MaxIdleConnsPerHost字段的缺省值，由http.DefaultMaxIdleConnsPerHost变量代表，值为2。也就是说，在默认情况下，对于某一个Transport值访问的每一个网络服务，它的空闲连接数都最多只能有两个。

与MaxIdleConnsPerHost字段的含义相似的，是MaxConnsPerHost字段。不过，后者限制的是，针对某一个Transport值访问的每一个网络服务的最大连接数，不论这些连接是否是空闲的。并且，该字段没有相应的缺省值，它的零值表示不对此设限。

DefaultTransport并没有显式地为MaxIdleConnsPerHost和MaxConnsPerHost这两个字段赋值，但是它却把MaxIdleConns字段的值设定为了100。

换句话说，在默认情况下，空闲连接的总数最大为100，而针对每个网络服务的最大空闲连接数为2。注意，上述两个与空闲连接数有关的字段的值应该是联动的，所以，你有时候需要根据实际情况来定制它们。

当然了，这首先需要我们在初始化Client值的时候，定制它的Transport字段的值。定制这个值的方式，可以参看DefaultTransport变量的声明。

最后，我简单说一下为什么会出现空闲的连接。我们都知道，HTTP协议有一个请求报文头叫做“Connection”。在HTTP协议的1.1版本中，这个报文头的值默认是“keep-alive”。

在这种情况下的网络连接都是持久连接，它们会在当前的HTTP事务完成后仍然保持着连通性，因此是可以被复用的。

既然连接可以被复用，那么就会有两种可能。一种可能是，针对于同一个网络服务，有新的HTTP请求被递交，该连接被再次使用。另一种可能是，不再有对该网络服务的HTTP请求，该连接被闲置。

显然，后一种可能就产生了空闲的连接。另外，如果分配给某一个网络服务的连接过多的话，也可能会导致空闲连接的产生，因为每一个新递交的HTTP请求，都只会征用一个空闲的连接。所以，为空闲连接设定限制，在大多数情况下都是很有必要的，也是需要斟酌的。

如果我们想彻底地杜绝空闲连接的产生，那么可以在初始化Transport值的时候把它的DisableKeepAlives字段的值设定为true。这时，HTTP请求的“Connection”报文头的值就会被设置为“close”。这会告诉网络服务，这个网络连接不必保持，当前的HTTP事务完成后就可以断开它了。

如此一来，每当一个HTTP请求被递交时，就都会产生一个新的网络连接。这样做会明显地加重网络服务以及客户端的负载，并会让每个HTTP事务都耗费更多的时间。所以，在一般情况下，我们都不要去设置这个DisableKeepAlives字段。

顺便说一句，在net.Dialer类型中，也有一个看起来很相似的字段KeepAlive。不过，它与前面所说的HTTP持久连接并不是一个概念，KeepAlive是直接作用在底层的socket上的。

它的背后是一种针对网络连接（更确切地说，是TCP连接）的存活探测机制。它的值用于表示每间隔多长时间发送一次探测包。当该值不大于0时，则表示不开启这种机制。DefaultTransport会把这个字段的值设定为30秒。

好了，以上这些内容阐述的就是，http.Client类型中的Transport字段的含义，以及它的值的定制方式。这涉及了http.RoundTripper接口、http.DefaultTransport变量、http.Transport类型，以及net.Dialer类型。

知识扩展

问题：http.Server类型的ListenAndServe方法都做了哪些事情？

http.Server类型与http.Client是相对应的。http.Server代表的是基于HTTP协议的服务端，或者说网络服务。

http.Server类型的ListenAndServe方法的功能是：监听一个基于TCP协议的网络地址，并对接收到的HTTP请求进行处理。这个方法会默认开启针对网络连接的存活探测机制，以保证连接是持久的。同时，该方法会一直执行，直到有严重的错误发生或者被外界关掉。当被外界关掉时，它会返回一个由http.ErrServerClosed变量代表的错误值。

对于本问题，典型回答可以像下面这样。

这个ListenAndServe方法主要会做下面这几件事情。

1. 检查当前的http.Server类型的值（以下简称当前值）的Addr字段。该字段的值代表了当前的网络服务需要使用的网络地址，即：IP地址和端口号.如果这个字段的值为空字符串，那么就用":http"代替。也就是说，使用任何可以代表本机的域名和IP地址，并且端口号为80。
2. 通过调用net.Listen函数在已确定的网络地址上启动基于TCP协议的监听。
3. 检查net.Listen函数返回的错误值。如果该错误值不为nil，那么就直接返回该值。否则，通过调用当前值的Serve方法准备接受和处理将要到来的HTTP请求。

可以从当前问题直接衍生出的问题一般有两个，一个是“net.Listen函数都做了哪些事情”，另一个是“http.Server类型的Serve方法是怎样接受和处理HTTP请求的”。

对于第一个直接的衍生问题，如果概括地说，回答可以是：

1. 解析参数值中包含的网络地址隐含的IP地址和端口号；
2. 根据给定的网络协议，确定监听的方法，并开始进行监听。

从这里的第二个步骤出发，我们还可以继续提出一些间接的衍生问题。这往往会涉及net.socket函数以及相关的socket知识。

对于第二个直接的衍生问题，我们可以这样回答：

在一个for循环中，网络监听器的Accept方法会被不断地调用，该方法会返回两个结果值；第一个结果值是net.Conn类型的，它会代表包含了新到来的HTTP请求的网络连接；第二个结果值是代表了可能发生的错误的error类型值。

如果这个错误值不为nil，除非它代表了一个暂时性的错误，否则循环都会被终止。如果是暂时性的错误，那么循环的下一次迭代将会在一段时间之后开始执行。

如果这里的Accept方法没有返回非nil的错误值，那么这里的程序将会先把它的第一个结果值包装成一个*http.conn类型的值（以下简称conn值），然后通过在新的goroutine中调用这个conn值的serve方法，来对当前的HTTP请求进行处理。

这个处理的细节还是很多的，所以我们依然可以找出不少的间接的衍生问题。比如，这个conn值的状态有几种，分别代表着处理的哪个阶段？又比如，处理过程中会用到哪些读取器和写入器，它们的作用分别是什么？再比如，这里的程序是怎样调用我们自定义的处理函数的，等等。

诸如此类的问题很多，我就不在这里一一列举和说明了。你只需要记住一句话：“源码之前了无秘密”。上面这些问题的答案都可以在Go语言标准库的源码中找到。如果你想对本问题进行深入

的探索，那么一定要去看net/http代码包的源码。

总结

今天，我们主要讲的是基于HTTP协议的网络服务，侧重点仍然在客户端。

我们在讨论了http.Get函数和http.Client类型的简单使用方式之后，把目光聚焦在了后者的Transport字段。

这个字段代表着单次HTTP事务的操作过程。它是http.RoundTripper接口类型的。它的缺省值由http.DefaultTransport变量代表，其实际类型是*http.Transport。

http.Transport包含的字段非常多。我们先讲了DefaultTransport中的DialContext字段会被赋予什么样的值，又详细说明了一些关于操作超时的字段。

比如IdleConnTimeout和ExpectContinueTimeout，以及相关的MaxIdleConns和MaxIdleConnsPerHost等等。之后，我又简单地解释了出现空闲连接的原因，以及相关的定制方式。

最后，作为扩展，我还为你简要地梳理了http.Server类型的ListenAndServe方法，执行的主要流程。不过，由于篇幅原因，我没有做深入讲述。但是，这并不意味着没有必要深入下去。相反，这个方法很重要，值得我们认真地去探索一番。

在你需要或者有兴趣的时候，我希望你能去好好地看一看net/http包中的相关源码。一切秘密都在其中。

思考题

我今天留给你的思考题比较简单，即：怎样优雅地停止基于HTTP协议的网络服务程序？

戳此查看Go语言专栏文章配套详细代码。

极客时间

GO语言核心36讲

3个月带你通关GO语言

郝林

《Go 并发编程实战》作者
GoHackers 技术社群发起人
前轻松筹大数据负责人