Skynet Overview

- 我们编写好的c文件，在编译成so库以后，在某个时机，调用该**so**库**api**的句柄，会被加载到一个**modules**列表中，一般这样的模块会被要求定义**4**种接口**create**，**init**，**signal**和**release**。**c**文件编译成**so**文件作为**lua**服务动态链接库。
- 我们要创建一个新的，运行该业务逻辑的上下文环境时，则从**modules**列表中，找到对应的**so**库句柄，并且调用**create**接口，创建一个该类业务模块的数据实例，并且创建一个上下文环境（**context**），引用该类业务的接口和数据实例，该**context**会被存放在一个统一存放**context**的列表中，这种**context**被称之为服务
- 一个服务，默认不会执行任何逻辑，需要别人向它发出请求时，才会执行对应的逻辑（定时器也是通过消息队列，告诉指定服务，要执行定时事件），并在需要时返回结果给请求者。请求者往往也是其他服务。服务间的请求、响应和推送，并不是直接调用对方的**api**来执行，而是通过一个消息队列，也就是说，不论是请求、回应还是推送，都需要通过这个消息队列转发到另一个服务中。skynet的消息队列，分为两级，一个全局消息队列，他包含一个头尾指针，分别指向两个隶属于指定服务的次级消息队列。skynet中的每一个服务，都有一个唯一的、专属的次级消息队列。mon
- **skynet**一共有**4**种线程，**monitor**线程用于检测节点内的消息是否堵住，**timer**线程运行定时器，**socket**线程进行网络数据的收发，**worker**线程则负责对消息队列进行调度（worker线程的数量，可以通过配置表指定）。消息调度规则是，每条**worker**线程，每次从全局消息队列中**pop**出一个次级消息队列，并从次级消息队列中**pop**出一条消息，并找到该次级消息队列的所属服务，将消息传给该服务的**callback**函数，执行指定业务，当逻辑执行完毕时，再将次级消息队列**push**回全局消息队列中。因为每个服务只有一个次级消息队列，每当一条**worker**线程，从全局消息队列中**pop**出一个次级消息队列时，其他线程是拿不到同一个服务，并调用**callback**函数，因此不用担心一个服务同时在多条线程内消费不同的消息，一个服务执行，不存在并发，线程是安全的

- **socket**线程、**timer**线程甚至是**worker**线程，都有可能会往指定服务的次级消息队列中**push**消息，**push**函数内有加一个自旋锁，避免同时多条线程同时向一个次级消息队列**push**消息的惨局。 综上所述，我们可以将skynet的机制，用一张图概括

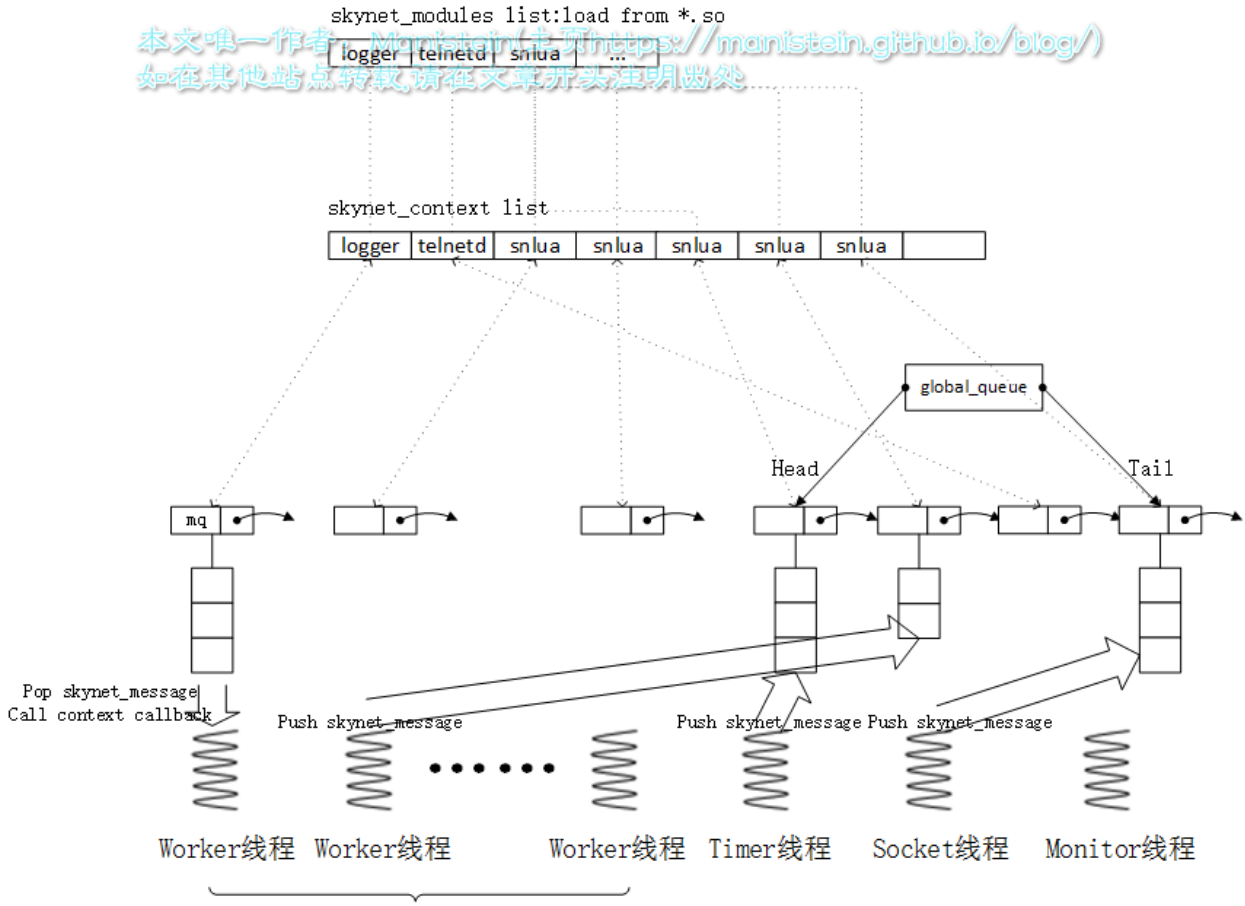

图1 从上面讨论可以得出如下结论，我们所写的不同的业务逻辑，可以运行在不同的独立的沙盒环境中，他们之间是通过消息队列来进行交互的。worker、timer和socket线程里运行的模块，都有机会向特定的服务push消息，他们是消息的生产者，而worker线程内的模块，同时也是消息的消费者（确切的说，应该是那些服务）**服务通过消息队列交互，其他线程是消息的生成者，worker线程是消息的消费者。** 注意：服务模块要将数据，通过socket发送给客户端时，并不是将数据写入消息队列，而是通过管道从worker线程，发送给socket线程，并交由socket转发。此外，设置定时器也不走消息队列，而是直接将在定时器模块，加入一个timer_node。其实这样也很好理解，因为timer和socket线程内运行的模块并不是这里的context，因此消息队列他们无

我们所有的lua服务，均是依附于一个叫snlua的c模块来运行的，lua服务每次收到一个消息，就会产生一个协程（事实上，skynet每个服务均有一个协程池，lua服务收到消息时，会优先去池子里取一个协程出来，这里为了理解方便，就视为收到一个消息，就创建一个协程吧），并通过协程执行注册函数，这些内容会在后面进行讨论。**服务依赖于snlua的c模块来运行的，lua服务每次收到一个消息，就会产生一个协程。**