« 对 skynet 的 gate 服务的重构 | 返回首页 | Skynet 发布第一个正式版 »

skynet 的 snax 框架及热更新方案

skynet 目前的 api 提供的偏底层，由于一些历史原因，某些 api 的设计也比较奇怪。（比如 skynet.ret 是不对返回数据打包的）

我想针对一些最常见的应用环境重新给出一套更简单的 api ，如果按固定模式来编写 skynet 的内部服务会简单的多。

这就是这两天实现的 snax 模块。今天我已经将其提交到 github 的 snax 分支上，如果没有明显的问题，将合并入主干。

snax 仅解决一个简单的需求：编写一个 skynet 内部服务，处理发送给它的消息。snax 并不会取代 skynet 原有的 api ，只是方便实现这类简单需求而已。

通常，我们需要编写一个 skynet 服务，它可以响应消息。用 skynet api 实现通常是这样的：

```
skynet.start(function()
  skynet.dispatch("lua", function(session, address, ...)
    dispatch(...)
  end)
```

我们需要先用 skynet.start 注册一个 skynet 服务的启动函数。然后确定这个服务用什么协议来解析消息。一般会选择 lua 协议，因为这种协议为 lua 设计，可以最高效的把 lua 原生数据序列化。

然后，我们会规定消息的第一个数据是字符串，表示消息类型。然后我们可以利用这个类型做不同的响应。

```
local command = {}

function command.foobar(...)
end

local function dispatch(cmd, ...)
  command[cmd](...)
end
```

skynet 内部消息可以是单向推送的，也可以是发起一个请求，然后等待回应。发送消息的人必须知晓这一点，分别用 skynet.send 或 skynet.call 来发送消息。

在服务实现的代码中，我们使用 skynet.ret 来回应请求。由于某些历史原因，这个 api 并不负责数据打包的工作。如果采用 lua 协议， 通常要写成 skynet.ret(skynet.pack(...)) 的形式。

snax 框架对这套流程做了一些简化， 把通用的代码放在了一个简单的框架里。编写一个服务就变成了这样：

```
-- foobar service
local i = 0
local hello = "hello"

function response.echo(data)
    return data, i
end

function subscribe.touch()
    i = i + 1
end

function init( ... )
    print ("service start:", ...)
```

```
end

function exit(...)
    print ("service exit:", ...)
end
```

这是一个简单的封装，只是把上面提到的 skynet.start skynet.dispatch 等调用藏起来了。init() 是启动会执行的代码，exit() 是退出要执行的代码。

response. 开头的函数表示返回值会调用 skynet.ret 返回，而 subscribe 函数则会扔掉返回值。

由于这段代码必须工作在 snax 框架下，所以，启动服务以及调用这个服务得使用 snax 的 api 。

```
local p = snax.newservice ("foobar", "hello world")
print(p.req.echo("foobar"))
print(p.pub.touch())
snax.exit(p)
```

snax.newservice 会在 config 文件中配置的 snax 路径上找到 "foobar" 模块并加载。由于 skynet 的核心是 C 编写的，所以服务的启动参数只能是一个字符串。为了更加灵活，snax 在实现时规定了启动服务的消息，newserice 的启动参数是用 lua 协议打包，通过启动消息传递的，也就不受限制了。

newservice 会产生一个对象方便使用。如果你从别的渠道拿到一个服务地址，那么可以用 snax.bind(handle, type) 生成同样的对象。这里的 type 是必须的，它用来分析协议组。

实际上 snax 的发送方和接收方都可以通过服务实现的源文件分析出这个服务支持哪些命令。这样可以正确感知发送的一条消息是否会收到回应。同时，也可以把命令类型从字符串转换为一个内部数字编号（减少数据传输量，以及提高编码和解码的性能）。

这里保留了 req 和 pub 分别对应 response 和 subscribe 。还是需要让调用者心里明确自己在做什么。因为在 skynet 中，发送一条消息是不会使 coroutine 挂起的，而等待回应则有可能中间插入其它 coroutine 的运行而导致状态改变。

热更新

早些年我们的项目是这样做热更新的：把修改过的 lua 文件重新加载一次，把消息处理函数换掉。但这种方法并非完备的。如果你理解 lua 如何工作，就能看出里面的坑来。

当服务本身有状态时，新旧版本的代码如何交接数据就是个难题。几乎无法给出一个使用者完全不用关心的方案。

snax 在给出热更新方法时，遵循了简单原则。我们把热更新作为一个产品上线后，紧急修复 bug 的非常手段。而并不打算用它做服务器的不停机升级。准确说，这是一个热修复接口。

基于这个定位，我并不打算让使用者更新他的源文件，命令 snax 重新加载它们。而是给了一个提交 patch 的接口，当发现有 bug 需要修复时，仅仅提交需要修改的那几个函数，并提供一个方法可以读写线上运行中服务的内部数据（通常是一些 local 变量）。

得益于 lua 5.2 提供的 upvalueid 和 upvaluejoin 一对新 api ，这些操作很容易实现。我们只需要把 patch 代码以字符串的形式传递进去。snax 独立编译这段 patch ，然后从正在运行中的环境里提取出 patch 可能涉及的 local 变量，用 upvaluejoin 绑定到 patch 中，最后将 patch 里提供的新函数替换掉前一个版本即可。

比如想换掉 subscribe.touch 的实现，只用

```
snax.hotfix(p, [[
local i   -- 必须有这个声明，用于标示引用的局部变量。

function subscribe.touch()
    i = i + 2
end
]])
```

这样就可以了。

如果在 patch 中实现一个hotfix() 函数，还可以在热更新的时候做一些额外的事情，或读取返回服务的内部状态。例如：

```
snax.hotfix(p, [[
local i

function hotfix(t)
  local tmp = i
  i = t
  return tmp
end
]],100)
```

这个 patch 可以把 i 设置为新的值，并返回之前的数值。

另外，这次 snax 集成了之前 mqueue 的特性。如果你需要让一个服务按次序处理消息（一个消息由于 io 操作挂起，也不会处理新到的请求），那么让 init 函数返回 "queue" 就可以开启这个模式。

云风 提交于 April 17, 2014 03:44 PM | 固定链接

COMMENTS

js 里面实现热更新，可以把prototype里面的函数替换下就实现了，当然这个替换过程如果要管理起来，则就需要有一个容器进行管理

Posted by: fantasyni | (7) August 1, 2014 03:39 PM

热更新某个函数的思路，带来了很大的隐患，你不能指望每个程序员像您一样严谨、优秀，不犯错误。某些函数本身也是有状态的，而且有些情况下更新某个函数并不能解决问题。
不如借鉴web开发的概念，直接模块重载，这样既简单也好理解。至于状态问题，一是可以通过内存数据库实现无状态化，二是在重载模块时，将模块状态持久化，载入新的模块后实现反持久化。

Posted by: gtfcugb | (6) July 30, 2014 02:16 PM

subscribe 改成了accept吧！

Posted by: acai | (5) May 15, 2014 11:05 AM

咩哈哈~ @thinka

Posted by: cupenoruler | (4) April 24, 2014 10:55 AM

咩哈哈~ @thinka

Posted by: cupenoruler | (3) April 24, 2014 10:55 AM

咩哈哈~ @thinka

Posted by: cupenoruler | (2) April 24, 2014 10:55 AM

刚吭哧吭哧啃完组播的代码，怎么就全删了。。。。

Posted by: thinka | (1) April 21, 2014 12:50 AM

POST A COMMENT

非这个主题相关的留言请到：留言本

名字：

Email 地址：

为了验证您是人类，请将六加一的结果（阿拉伯数字七）填写在下面：

URL:

记住我的信息?

留言:
（不欢迎在留言中粘贴程序代码）

提交