SocketChannel

请求回应模式是和外部服务交互时所用到的最常用模式之一。通常的协议设计方式有两种。

1、**每个请求包对应一个回应包，由 TCP 协议保证时序。redis** 的协议就是一个典型。每个 redis 请求都必须有一个回应，但不必收到回应才可以发送下一个请求。

2、发起每个请求时带一个唯一 **session 标识**，在发送回应时，带上这个标识。这样设计可以不要求每个请求都一定要有回应，且不必遵循先提出的请求先回应的时序。**MongoDB** 的通讯协议就是这样设计的。

skynet 提供了一个更高层的封装：socket channel 用来实现上述两种模式。

```
local sc = require "skynet.socketchannel"
local channel = sc.channel {
  host = "127.0.0.1",
  port = 3271,
}
```

这样就可以创建一个 channel 对象出来，其中 host 可以是 ip 地址或者域名，port 是端口号。

接下来，我们就可以工作在模式 1 下了：

```
local resp = channel:request(req [, response[, padding]])
```

这里，req 是一个字符串，即请求包。response 是一个 function ，用来收取回应包。返回值是一个字符串，是由 response 函数返回的回应包的内容（可以是任意类型）。response 函数需要定义成这个样子：

```
function response(sock)

  return true, sock:readline()end
```

sock 是由 request 方法传入的一个对象，sock 有两个方法：read(self, sz) 和 **readline**(self, sep) 。**read** 可以读指定字节数；**readline** 可以读以 sep 分割（默认为 \n）的一个字符串（不包含分割符）。

response 函数的第一个返回值需要是一个 boolean ，如果为 true 表示协议解析正常；如果为 false 表示协议出错，这会导致连接断开且让 request 的调用者也获得一个 error 。

在 response 函数内的任何异常以及 sock:read 或 sock:readline 读取出错，都会以 error 的形式抛给 request 的调用者。

这里 dispatch 是一个解析回应包的函数，和上面提到的模式 1 中的解析函数类似。但其返回值需要有三个。第一个是这个回应包的 session，第二个是包是否解析正确（同模式 1 ），第三个是回应内容。

socket channel 就是依靠创建时是否提供 response 函数来决定工作在模式 1 还是模式 2 下的。

在模式 2 下,request 的参数有所变化。第 2 个参数不再是 response 函数(它已经在创建时给出),而是一个 session 。这个 session 可以是任意类型,但需要和 response 函数返回的类型一致。socket channel 会帮你匹配 session 而让 request 返回正确的值。

channel:close() 可以关闭一个 channel ,通常你可以不必主动关闭它,gc 会回收 channel 占用的资源。

---

socket channel 在创建时,并不会立即建立连接。如果你什么都不做,那么连接建立会推迟到第一次 request 请求时。这种被动建立连接的过程会不断的尝试,即使第一次没有连接上,也会重试。

你也可以主动调用 channel:connect(true) 尝试连接一次。如果失败,抛出 error 。这里参数 true 表示只尝试一次,如果不填这个参数,则一直重试下去。

由于连接可能发生在任何 request 之前(只要前一次操作检测到连接是断开状态就会重新发起连接),所以 socket channel 支持认证流程,允**许在建立连接后,立刻做一些交互**。如果开启这个功能,需要在创**建 channel 时,填写一个 auth 函数**。和 response 函数一样,会给它传入一个 channel 对象。auth 函数不需要返回值,如果认证失败,在 auth 函数中抛出 error 即可。

由于对端有可能在任何时候断开连接,所以任何一次 request 都有可能抛出 error 而失败,socket channel 将在下一次 request 时重新建立连接。注意:重连并不会重发过去发生 error 的请求,连接断开并不是隐藏在内部的。所以,如果有必要,你应该在**请求失败时,业务层重新提出请求**。

---

socket channel 也可用于仅发包而不接收回应。只需要在 request 调用时不填写 response 即可。

channel:response 则可以用来单向接收一个包。

```
channel:request(req)local resp = channel:response(dispatch)

-- 等价于

local resp = channel:request(req, dispatch)
```

request 还有第三个参数 **padding** ，这是用来**将体积巨大的消息拆分成多个包**发出用的（如果你的协议支持）。padding 需求是一个 table ，里面有若干字符串。如果提供了 padding 参数，socket channel 将连同 req 以及 padding 数组里的字符串，利用 Socket 的低优先级通道发出（使用 socket.lwrite）。

这种用法下的 response 函数，应该多返回一个 padding 值。即，对于模式一返回 succ:boolean data:string padding:boolean 三个值；对于模式二，返回 session:number succ:boolean data:string padding:boolean 四个值。

padding 表明了后续是否还有该长消息的后续部分。

如果回应消息是由多个短小的消息合成。channl:request 将返回一个 table ，里面有所有短消息的内容，由调用者来连接这些短消息。

关于 socket channel 的具体用法除了阅读 lualib/socketchannel.lua （同时这也是理解 socket 模块的好材料）的实现外，也可以阅读 lualib/redis.lua 和 lualib/mongo.lua 这两个为 skynet 编写的数据库 driver 。