Jsonrpc

cita 微服务每个模块负责不同的功能，而且每个模块都可以再分为多个子功能。每个模块会有自己的侧重点，即会有一个最根本的要求，必须达到这个要求之后，这个模块才成立。总的来说就是：每个模块都会有一个最核心的子功能。
jsonrpc 是接口模块，最核心的任务就是处理用户的请求。这样就对软件的使用性、体验性、响应时延等会有很高的要求，这就是 jsonrpc 模块区别与其他模块的最根本的要求，其他模块不会直接面向用户，这方面的压力就没有这么大（当然对性能的要求时刻都在）。
auth 模块最核心的子功能是　　xxxxx 交易池的维护。
consensus 模块最核心的子功能是　xxxxx 达成共识。
chain 模块最核心的子功能是　　xxxxx 存储数据块。
executor 模块最核心的子功能是　xxxxx 执行交易。
network 模块最核心的子功能是　xxxxx 维持与其他节点的通信。

jsonrpc 是接口模块。
接口模块的要求有：
　　响应延迟、
　　并发
TODO: ***这个再多看书研究。***

【主线程流程】
流程跟代码有些不一致，按照我的理解做了些调整。
1. 设置资源限制
接口模块需要设置打开文件最大个数等这些限制。后端程序需要明确这些限制，防止资源不够用的问题。这也是 jsonrpc 这个模块特有的。
参见下面对 fdlimit.rs 的分析。

2. 读取配置文件
```
struct Config {
    backlog_capacity: usize,
    profile_config: ProfileConfig,
    http_config: HttpConfig,
    ws_config: WsConfig,
    new_tx_flow_config: NewTxFlowConfig,
}
```
这里有 5 项配置：backlog_capacity 是衡量 Response 池子大小的，http_config 是与 http 服务相关的配置，ws_config 是与 websocket 服务相关的配置，new_tx_flow_config 是批量发送交易给 auth 的配置。
TODO: Profile 相关的内容都先暂时不考虑。

并检查一下配置：如果 ws 和 http 都不使能，就退出。

3. 新建 ws 线程

启动 websockert 服务，用来接收用户发来的消息。
参见下面对 ws_handler.rs 的分析。

4. 新建 http 线程组
启动 http server 服务，用来接收用户发来的 http 请求。
按照 cpu 个数生成线程，每个线程启动服务来监听 ip 地址和端口。
参见下面 http_server.rs 的分析。

5. 建立 Response 池
建了一个 response 池来缓存信息。缓冲池是个 HashMap，最大个数限制来自于配置文件。

Response 池是做什么的呢？目的是用来跟踪每条消息的处理。要记住一点：Jsonrpc 接收到用户请求之后，由于本模块并没有保存什么内容，需要向其他模块发送请求，等收到 Response 之后分析结果并返回给用户。
http 在收到用户请求之后，会为每一条消息建立一个通道，然后在 Response 中添加一条记录，记录里包含 sender 和一些信息，然后 http 内部持有 receiver。Mq handler 在收到其他模块的 Response 之后，通过 Response 记录里的 sender 发送消息，http 内部的 receiver 接收到消息进行处理。 ws 的处理流程也是如此。

6. 新建 Dispatch 线程
Dispatch 线程用来分发 Jsonrpc 向其他模块的请求消息。即 Jsonrpc 向其他模块发送的消息都要经过 Dispatch 来转发。
处理机制：
* 有全局的 req buffer 和时间戳。
* 如果收到 http 或者 ws 发过来的消息，进入下列处理：
    如果非 RequestNewTx 类型，说明是查询类消息 Request 或 RequestNet，立刻转发出去；
    如果是，说明是新交易，将消息存入 buffer，然后判断 batch 是否够大或时间够长，满足条件就转发出去。
* 如果收到错误消息，就检查一下 buffer 中是否有数据，有就转发出去。

转发步骤是：
    取出 buffer 中所有消息，加上 request_id，发送出去。并清理 buffer，更新时间戳。
TODO: request_id 是个随机数，用在 jsonrpc 和 auth 中，不知道有什么特殊的作用。

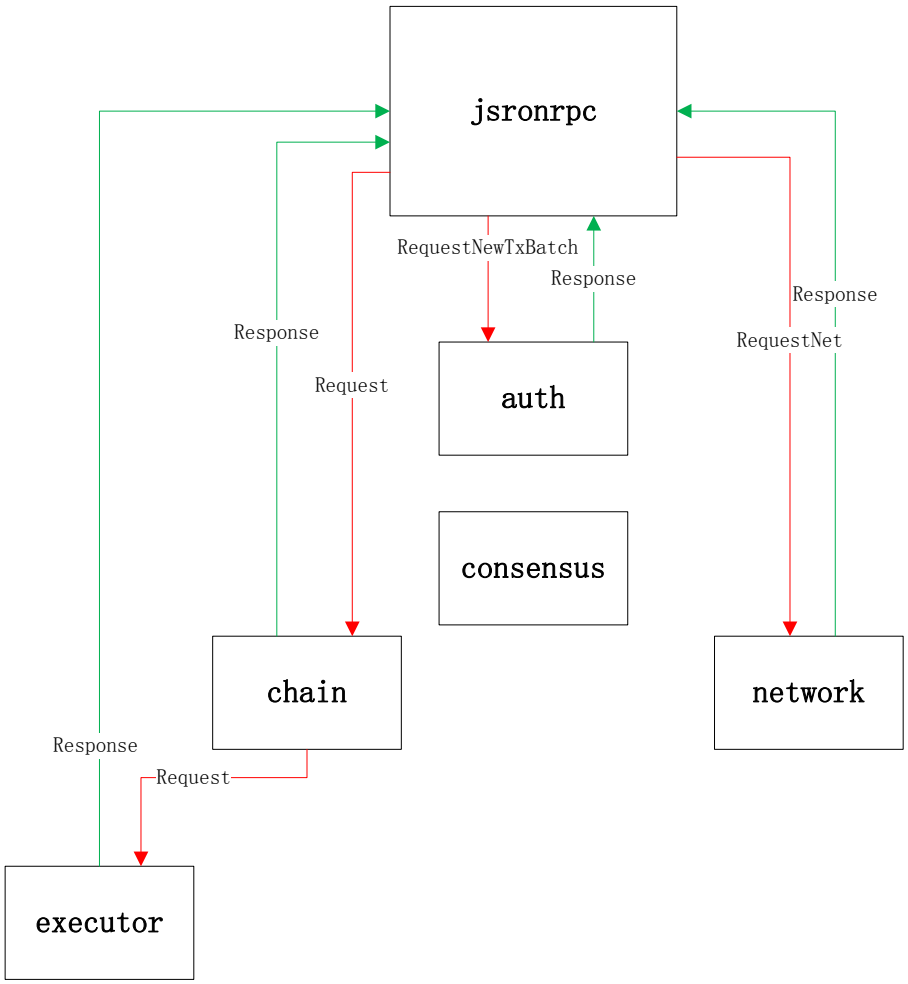
为什么需要 Dispatch？目的就是为了完成批量转发交易。

7. 订阅消息
接收的消息有：
    Auth >> Response
    Chain >> Response
    Executor >> Response
    Net >> Response

可以看出 Jsonrpc 明显区别于其他模块的一个地方：接收的消息全是其他模块的 Response。

这是因为真正的用户输入是通过 http 请求或者 web socket 进来的，由 http server 或 ws server 来接收的，而不是通过 MQ 进来。

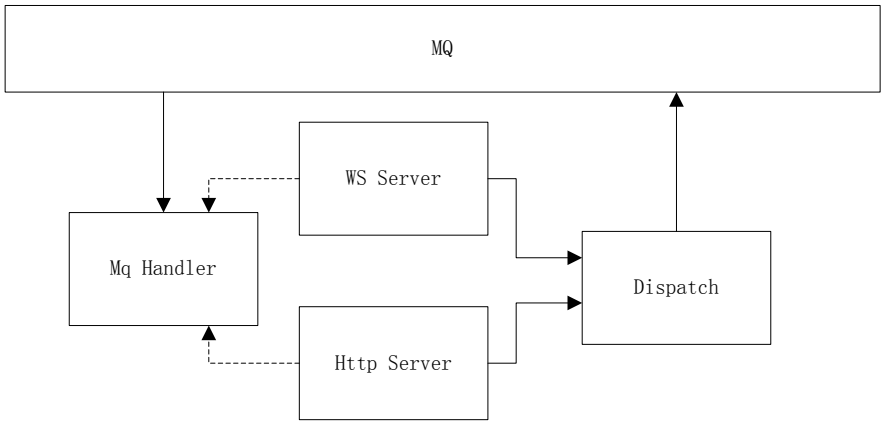
有一个地方需要注意：
Jsonrpc 发出去的信息，都是要有响应的，因为需要返回给用户一个结果。
那么为什么 Executor 没有收到 Jsonrpc 发过来的 Request 信息，还要给它会一个 Response 呢？
这是因为 Chain 和 Executor 的分离，有些信息 Chain 里面没有，只能在 Executor 中查，所以 chain 会将这些消息转给 Executor（Chain >> Request），然后 Executor 恢复 Response 给 Jsonrpc。
参见下面的模块关系图。



9. 新建 Mq handler
循环接收 mq 消息，在接收到其他模块发来的 Response 之后，需要解析并返回给用户。
参见 mq_handler.rs 的分析。

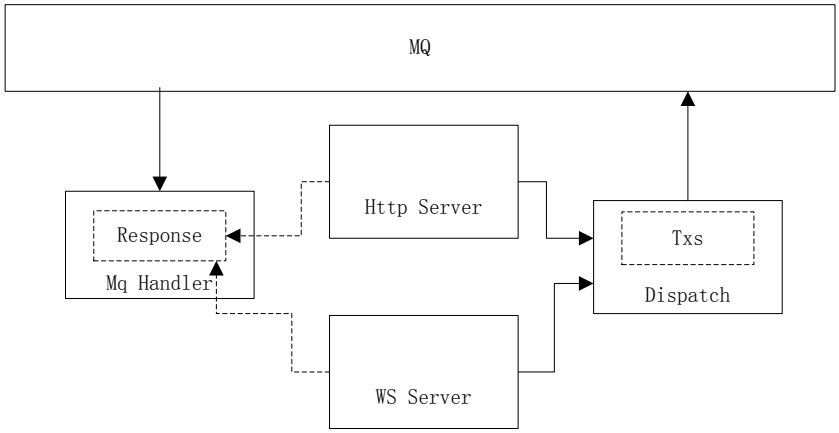
总结：

下面是 Jsonrpc 线程结构图。实线是通道；虚线是 Response 引用，Mq handler、http server、ws server 共享 Response 池。



Jsonrpc 内部数据结构如下图所示。Response 是 Jsonrpc 的数据核心，模块内部全局性的缓存池就这一个。http server 或者 ws server 在收到消息后会发给 Dispatch，并在 Response 池中添加一条记录，Mq handler 在收到其他模块的 Response 之后，会从 Response 中取出内容，发送给 http server 或者 ws server，然后由它们返回给用户。

Dispatch 内部有 Tx buffer，Http server 或者 ws server 在收到交易后，会发送 Dispatch，由 Dispatch 来进行批量转发。



【文件结构】

Jsonrpc 模块有 8 个文件：fdlimit.rs、config.rs、helper.rs、http_server.rs、ws_handler.rs、mq_handler.rs、main.rs、response.rs。

main.rs 已经在上面描述过了，config.rs 解析配置文件，下面看其他的文件，先从简单的开始。

1. fdlimit.rs

内核对每个用户进行了资源限制，有很多项资源分类，包含最大打开文件的个数，一般是 1024，由于我们是后端程序，所以这里讲文件打开个数设置为最大。

可以从网上找些资料看下 linux 对用户使用资源的控制。

2. helper.rs

内容较少，定义了 TransferType、RpcMap、ReqSender、select_topic。

RpcMap 是 Response 池的类型，

```
type RpcMap = Arc<Mutex<HashMap<Vec<u8>, TransferType>>>
enum TransferType {
        HTTP((RequestInfo, oneshot::Sender<OutPut>)),
        WEBSOCKET((RequestInfo, ws::Sender)),
}
```

可以从 RpcMap 中看出 Response 池中都存了哪些内容，HashMap 的 key 是 request id，用来标记每个请求；value 是 TransferType，里面是 RequestInfo 和 sender，前者是请求的消息，后者就是一个 sender，能够将信息发送给 http 或者 ws。

ReqSender 是一个 sender，

```
type ReqSender = Mutex<mpsc::Sender<(String, ProtoRequest)>>
```

select_topic 根据请求中的 method 来决定 MQ 消息的类型：

* 如果是 peerCount，就向 network 发送 RequestNet 消息；
* 如果是 senRawTransaction 或者 sendTransaction，就设置为 RequestNewTx 消息，然后在 Dispatch 线程中 forward_service 中处理；
* 其他的都是信息查询请求，向 Chain 发送 Request 消息。

这里有一个巧妙的地方（坑）：

发送交易的时候会形成 RequestNewTx 类型的消息，但是后面会打包成 RequestNewTxBatch 再发送给 auth。

从 select_topic 也可以看出，jsonrpc 发出的消息就 3 种：Request、RequestNet、RequestNewTxBatch。

3.ws_handler.rs

TODO: *web socket 之前没用过，就先不看了，以后再学习。*

4. http_server.rs

启动 http server 需要一些背景知识，而且使用起来是有固定的 trait 等。

我们分析一下这个文件几个关键的方法：

start：

启动 http service。

有几个关键：访问权限控制、json 类型、sender（发送给 Dispatch）、Response 池、timeout。

handle_preflighted：

被 call 调用，来处理非 POST 的请求

call：

是回调，当有请求进来时会被调用。下面主要分析 Post 方法的请求。

根据 Json 请求的类型分为 single 和 batch，分别处理，这里有 4 个比较重要的方法：read_single、handle_single、read_batch、handle_batch，都是供 call 调用。

single：

read_single 读取 json 请求

handle_single 将消息插入 Response 池，然后发送给 Dispatch
select 等待处理

batch：

跟 single 类似，也是调用 read_batch、handle_batch，将 batch 里的消息逐个插入到 Response 池，并发送给 Dispatch，然后 select 等待。

总结一下，这里的主要工作就是接收到请求之后，创建通道，将请求信息和 sender 放入到 Response 池中，并发送给 Dispatch 来分发消息。

5. response.rs
负责返回给用户结果。

6. mq_handler.rs
负责从 mq 收到 Response 消息之后的一些处理。
处理流程：

收到 Response 消息之后，根据 request_id 为 key 将其在 Response 池中删除
使用 Response 中记录下的 sender 发送给 http 或者 ws。

原子功能：

接收用户请求：http、ws
维护 Response 池，包括添加、删除：http、ws、mq handler
转发消息：Dispatch
批量发送交易：Dispatch
返回结果给用户：response

TODO: 对 Request 的分析
分析用户都有哪些 Request？