V 1.0

模块规格说明书

撰写：Berzeck
翻译：Ray(Lambda)
NULS.IO

NULS 2.0 Platform - 模块规格说明书

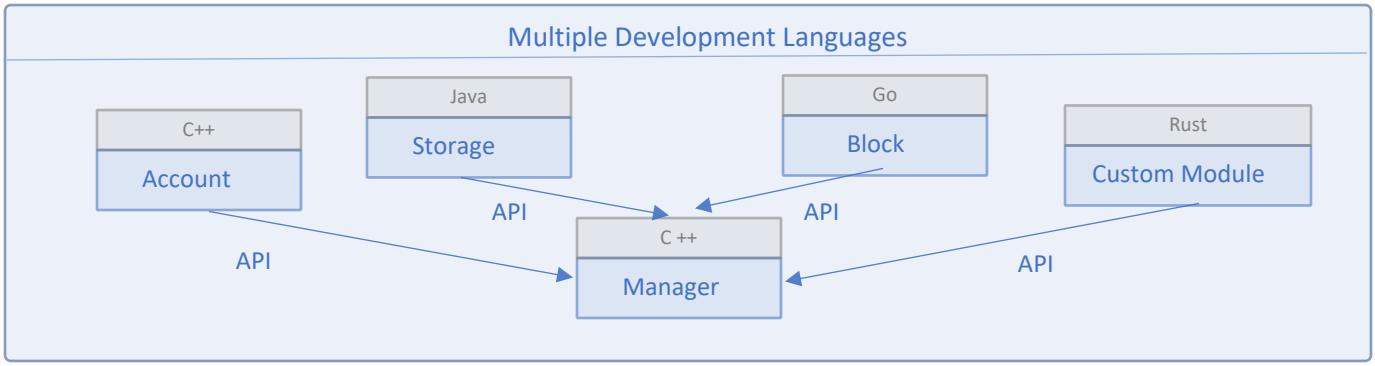
目录

1]设计目标

1.1] 开发灵活

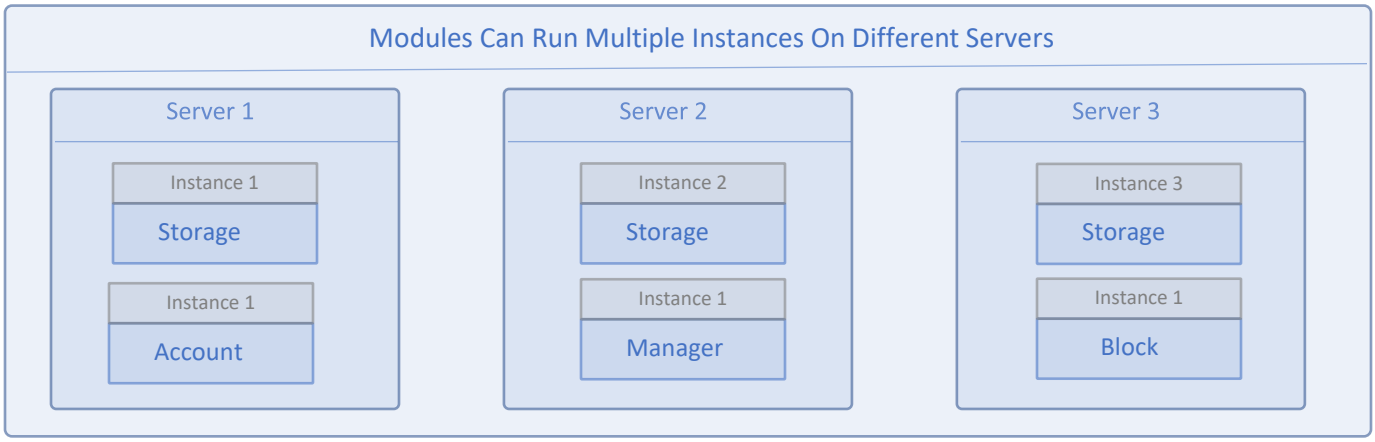
该平台与语言无关，因此可以使用不同的编程语言开发模块。 具体使用何种语言开发由开发人员自己决定，唯一的要求是模块使用相同的传输机制（WebSockets / JSon），并将 API 发送给 Manager 模块。

这样开发人员专注于单个模块，而不需要学习整个基础架构。 因为开发只需要专注于其业务领域所需的特定功能，所以学习在该平台上开发的学习曲线将会低很多，这样大大提高可业务模块的开发实现效率。



1.2] 部署灵活

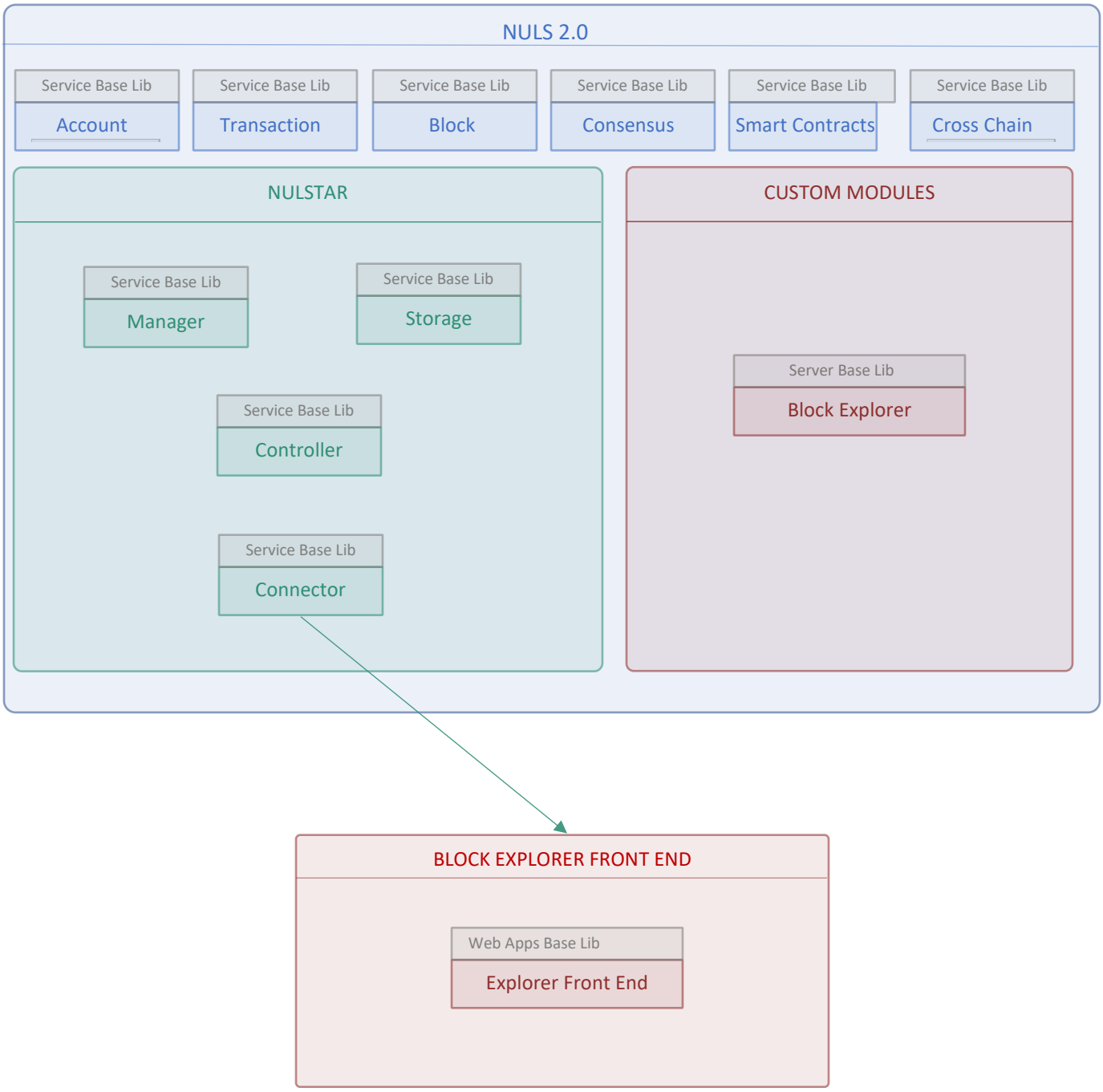
每个模块实际上都是一个单独的应用程序，因此可以根据每个公司的特定需求将有些模块放在单独的环境中。 例如，如果特定应用程序中的模块性能密集，则可以将其部署在多个并发运行的服务器中。



1.3] 构建复杂系统的标准框架

各个组件将作为构建 NULS 软件系统的基础，如交易所，社交平台，拍卖网站，博彩网站，浏览器等。NULS 不再只是一个区块链实现，而成为一个开发平台，可以在平台上构建各种复杂的应用程序。

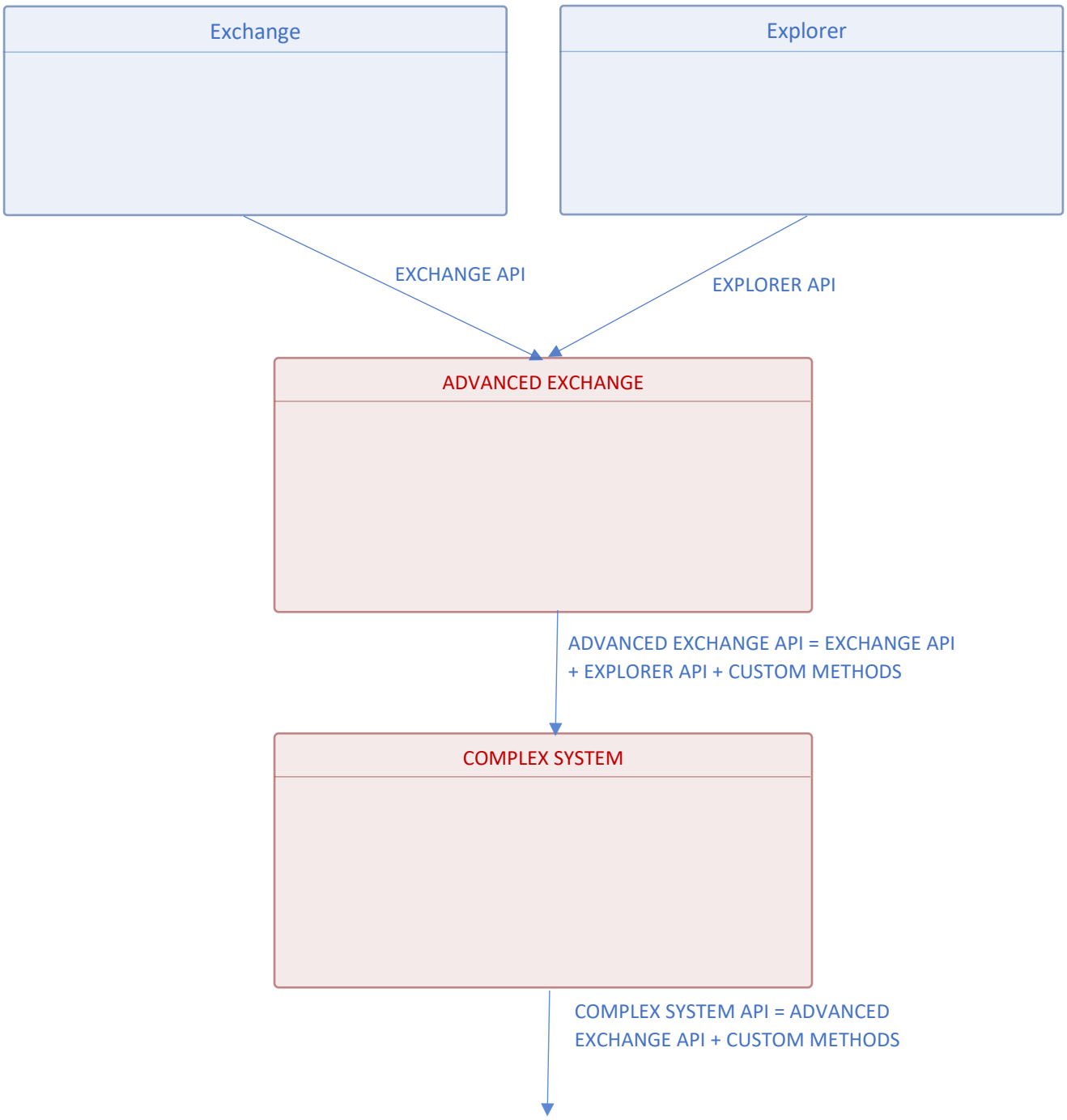
例如，要构建块浏览器，只需要将一个模块连接到标准模块（在第 2 节中描述），该标准模块将执行特定功能并将对应的 API 公开给外部应用程序使用。 如果没有需要定制的 API 功能，那么只需要开发块浏览器前端功能。


NULS 2.0
Service Base Lib
Account
Service Base Lib
Transaction
Service Base Lib
Block
Service Base Lib
Consensus
Service Base Lib
Smart Contracts
Service Base Lib
Cross Chain
NULSTAR
Service Base Lib
Manager
Service Base Lib
Storage
Service Base Lib
Controller
Service Base Lib
Connector
CUSTOM MODULES
Server Base Lib
Block Explorer
BLOCK EXPLORER FRONT END
Web Apps Base Lib
Explorer Front End

1.4]互联生态系统

围绕该平台构建的系统将能够彼此连接，促使其生态系统以比其他加密货币系统更快且更低成本的方式进行改进。 这可以归功于系统间使用相同的消息协议和简约的 API 约定。

例如，如果已经有了浏览器和交易平台系统，那么构建一个提供这两个系统功能的应用程序会很容易，只需订阅相关功能函数并将结果呈现在前端页面上；此系统还可以自动继承两个基本系统的 API，并添加自己的功能，使其可以用作其他更复杂系统的基础功能。



1.5] 模块间完全解耦

该体架构保证模块之间没有硬编码依赖关系，因此在系统运行时很容易向平台添加新的服务。 如果模块找不到任何可以提供所需内容的服务，它可以优雅地向其他组件报告此情况，而不会导致整个应用程序崩溃。

1.6] 接近零停机时间

该架构可以很容易提供持续运行的服务。如果某个模块崩溃，则可以使用标准的云部署工具自动优雅地将该模块再次部署。如果要发送消息给该模块，则需要等待几秒钟，直到部署了该模块一个新的实例。 如果模块是关键模块，则可以部署多个并行工作单元，Manager 将充当负载均衡器。

即使是升级也可以在不重新启动整个基础架构的情况下完成，因为每个模块都可以单独更换，并且升级可以分步执行。 此外，可以部署相同模块的两个版本，企业可以创建规则，规则要求新版本只能占用 5%的时间来测试真实运行情况。这样做不会冒接收处理所有命令会出现异常的风险，当然如果升级失败，可以优雅自动地恢复以前的版本。

1.7] 提高安全性和质量保证

由于模块是高内聚松耦合的软件，因此可以高效完成代码审查和测试。每次更改只会影响正在开发的模块，而基础架构不会受到影响。 它还允许灵活地为每个模块提供不同级别的测试

例如，外部只需要检视一些关键模块，而无需检视整个平台。 由于这些服务可能在不同的服务器中运行，因此更不会破坏整个平台。

1.8] 发布-订阅（Publish - Subscription）开发模式

加密货币系统中的大多数应用程序都是使用传统技术设计和开发，所以当内部业务功能状态不断变化需要相应地更新应用程序时，这些技术并不适用。 例如，当节点正在下载交易事务时，有一个函数返回已处理的事务数，这个函数需要不断轮询以更新应用程序，但轮询是获取信息的最低效方法之一。

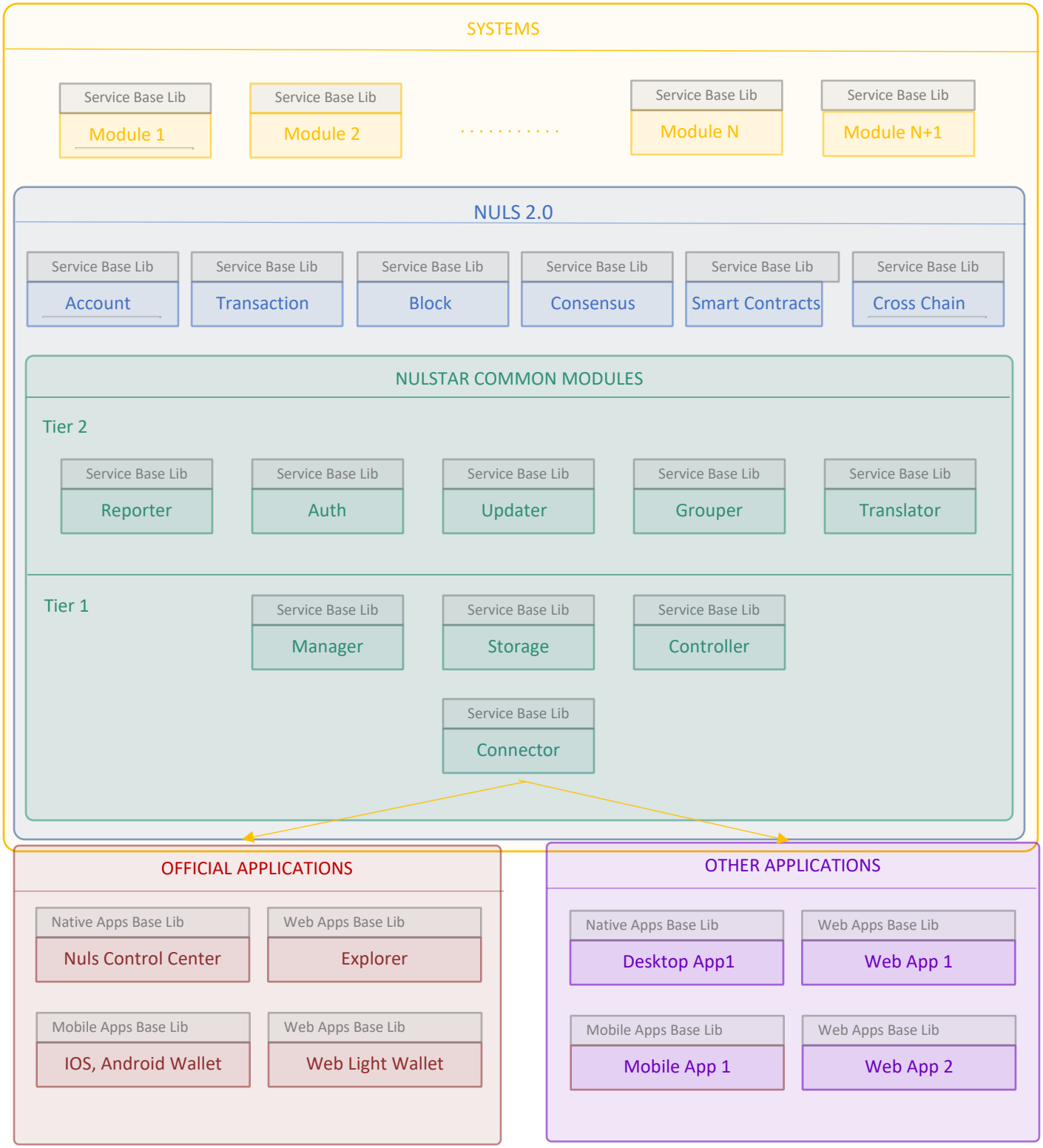
使用发布-订阅（Publish - Subscription）开发模式，可以完全消除此问题。因为该函数将在特定时段或由用户定义的事件触发执行而更新数据，所以 当应用程序不再需要该信息时，它只是“取消订阅”该功能。

1.9] 专用的用户界面

用户界面是完全与核心架构解耦的应用程序，它们之间通过传输协议和 API 交互。 这非常重要，因为用户界面的开发周期非常短，而核心的开发周期很长；多个开发人员和设计人员可以并行工作，无需彼此协调，甚至无需与核心开发团队协调。 该属性将允许开发多个接口以满足不同的需求，因为有些公司可能需要功能强大且复杂的接口，而其他公司需要简洁易用的接口，有些可能更倾向 Web 应用程序而有些更愿意开发主机应用程序。

2] 架构的顶层概念设计

FIG 2.1



2.1] 综述

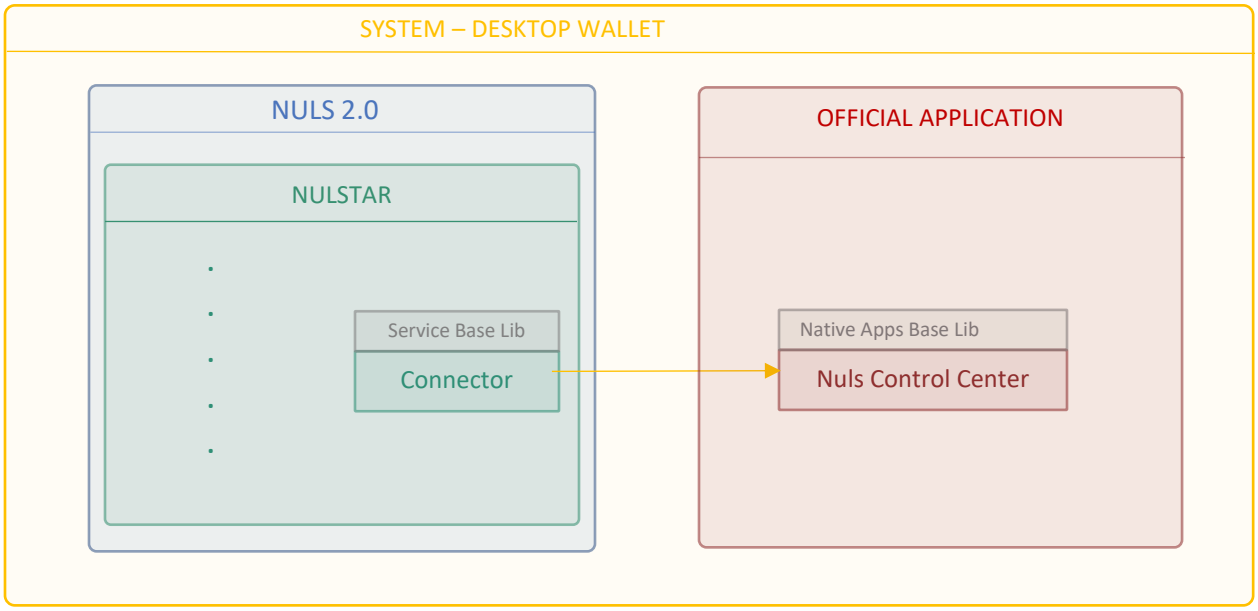
如图 2.1 所示，可以使用 NULS 2.0 组件构建所有类型的复杂系统。所有这些系统都将通过 Connector 模块公开其 API，该模块通过 WebSockets 使用 Json 数据包交换消息。

命名为 Modules.cfg 的配置文件应随模块一起部署到每个服务器实例中，它包含特定参数和模块可能需要的其他指令。

反过来，模块可以由作为插件开发的较小组件组成；例如，共识（Consensus）模块可以有多个组件，每个组件对应一个共识机制，如 PoC，PoS，PoW；根据 Modules.cfg 配置文件指定的组件，在启动应用程序时激活这些组件。

像 GUI，资源管理器，轻型钱包和其他应用程序等外部应用程序应建立与 Connector 模块的连接，以请求正常运行所需的任何信息。每一个应用程序可以和 NULS 2.0 模块一起组合在一个包中，以方便最终用户使用。

例如，桌面钱包只是 NULS 2.0 组件，它可以和 NULS Control Center 组合在一个的软件包中发布使用：



2.2] 基础库

基础库（在图 2.1 中用灰色框表示）提供了基本的标准功能，允许模块和外部应用程序简单无缝地集成到平台。 这些库支持多种语言，不同语言背景的开发者可以开发自己的模块和应用程序。

基础库有：

- 服务基础库：所有模块都应继承此库，该库负责提供与其他基础模块无缝交互的通用工具，如连接例程，发送/接收 Json 包等。
- 原生应用基础库：该库允许开发桌面应用程序，包括 GUI 界面，平台实现细节抽象。
- 移动应用基础库：支持移动和平板电脑平台应用开发
- Web 应用基础库：提供必要的方法来为应用程序构建 Web 前端

2.2] Nulstar

Nulstar 由一组模块组成，可用于开发任何类型的复杂系统，包括 NULS 2.0。 所有这些系统都将通过 Connector 模块公开其 API，该模块通过 WebSocket 建立连接，使用 Json 格式交换消息。

第 1 层（Tier 1）包含运行底层基础架构所需的最小模块集，包括:

Manager, Storage, Controller, Connector

第 2 层（Tier 2）有一些可选的模块，它们以系统标准的方式提供大多系统所需的常用功能，包括:

Reporter, Auth, Updater, Grouper, Translator

2.3] NULS 2.0

NULS 2.0 由 Nulstar 第 1 层提供的模块和实现 NULS 区块链协议的特定模块组成，在后期阶段还将提供第 2 层模块。

NULS 2.0 特定模块有:

Account, Transaction, Block, Consensus, Smart Contracts, Cross Chain

3] 基础库详细设计

3.1] 服务基础库（Service Base Library）

每个需要集成到该平台的模块都应当使用服务基础库。服务基础库提供基础通用的功能集合，提供给不同语言开发的模块使用。开发者需要自己选择调用该库完成模块功能。

该库的功能有:

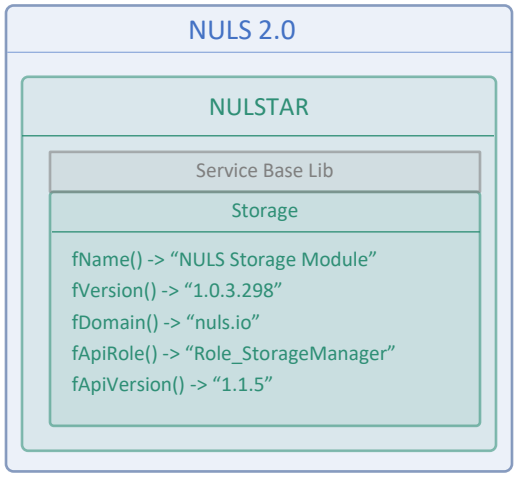
- 提供模块的基本信息
- 启动 WebSocket 服务端创建连接通讯
- 提供按需添加 WebSocket 服务端功能
- 按 JSon 规范格式编码发送的消息报文
- 按 JSon 规范格式解码收到的消息报文
- 管理等待应答的请求队列
- 严重事件日志硬盘归档
- 收集模块 API 的方法和方法参数
- 给每个 API 方法添加元信息（Meta），包括方法描述，方法参数验证和事件/周期限制
- 连接 Storage 模块存储日志
- 发送非严重事件日志信息给 Storage 模块
- 使用 Manager 模块返回的连接信息，根据需要协商建立与其他模块的 WebSocket 连接

3.1.1] 提供模块的基本信息

该功能由五个返回字符串类型的函数来完成:

- fName() .- 模块的名称
- fVersion() .- 模块的版本号。由四部分数字组成：Major, Minor, Bug, Build。Major 数字用于标识模块功能大规模修改或重构。Minor 标识模块新增功能。Bug 标识模块新增小功能或修复模块的问题。Build 标识模块被重新构建。如果 Major 增加代表模块升级向前不兼容，要特比留心此版本模块的升级。其他版本号增加都是兼容性升级。
- fDomain() .- 为了避免和第三方模块名字和角色的冲突，必须提供 domain 作为服务前缀，最好使用唯一的字符串，建议使用 web domain 名字
- fApiRole() .- 一个标识模块角色的字符串。习惯上字符串以“Role_”开头，相似角色的模块必须提供相似的 API。否则 Manager 模块使用 domain 来指定使用的模块。可以并行运行多个提供相同角色的模块，此时 Manager 相当于负载均衡器（load balancer）。
- fApiVersion. – API 的版本号。由四部分数字组成：Major, Minor, Bug。Major 数字用于标识 API 修改向前不兼容。Minor 标识 API 添加了新的方法或修改了已有实现，但修改是向前兼容的。Bug 标识 API 很小的变更，比如方法的描述或向前向后都兼容其他模块的变更，此时只是 Bug 版本号不同，Major 和 Minor 版本号是不变的。

  如图所示：

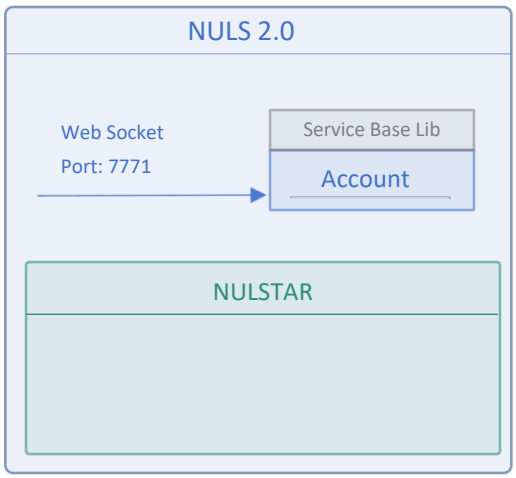


3.1.2] 启动 WebSocket 服务端

默认情况模块需要打开 WebSocket 服务端，来接收客户端的连接。日志级别和加密方式在启动模块命令行中指定。如果端口号不可用必须使用下一个端口号，并且日志记录到 Storage 模块，如果模块不可用必须记录严重日志到硬盘上。

WebSocket 端口号仅作为内部模块连接用，如果需要外部模块连接，需要 Connector 模块新建 WebSocket 服务端来接收外部的请求。

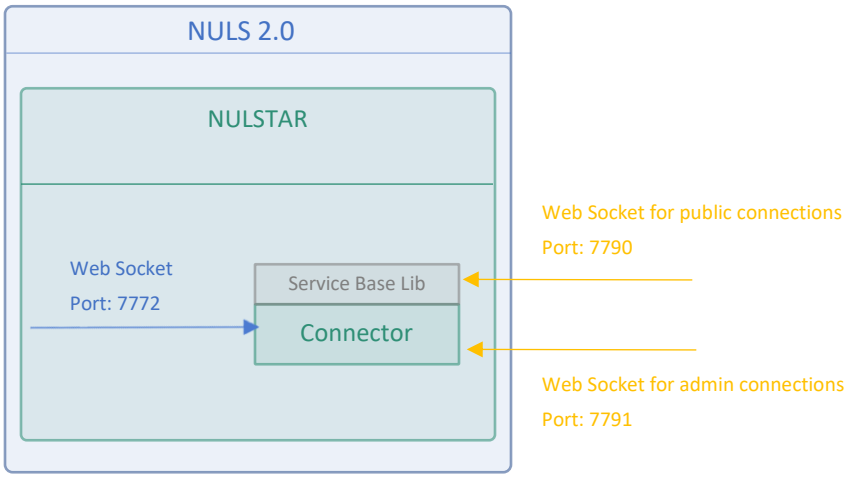
例如下图：Account 模块开启 WebSocket 服务端，在 7771 端口上接收连接。



3.1.3] 提供按需增加 WebSocket 服务端功能

该库需要提供创建多个 WebSocket 服务端的功能，满足模块的特殊需要。

比如下图：Connector 模块需要开启两个以上的 WebSocket 服务端接收外部应用的连接，一个用于特权用户连接，其他用于一般用户连接。



3.1.4] 按 JSon 规范格式编码发送的消息报文

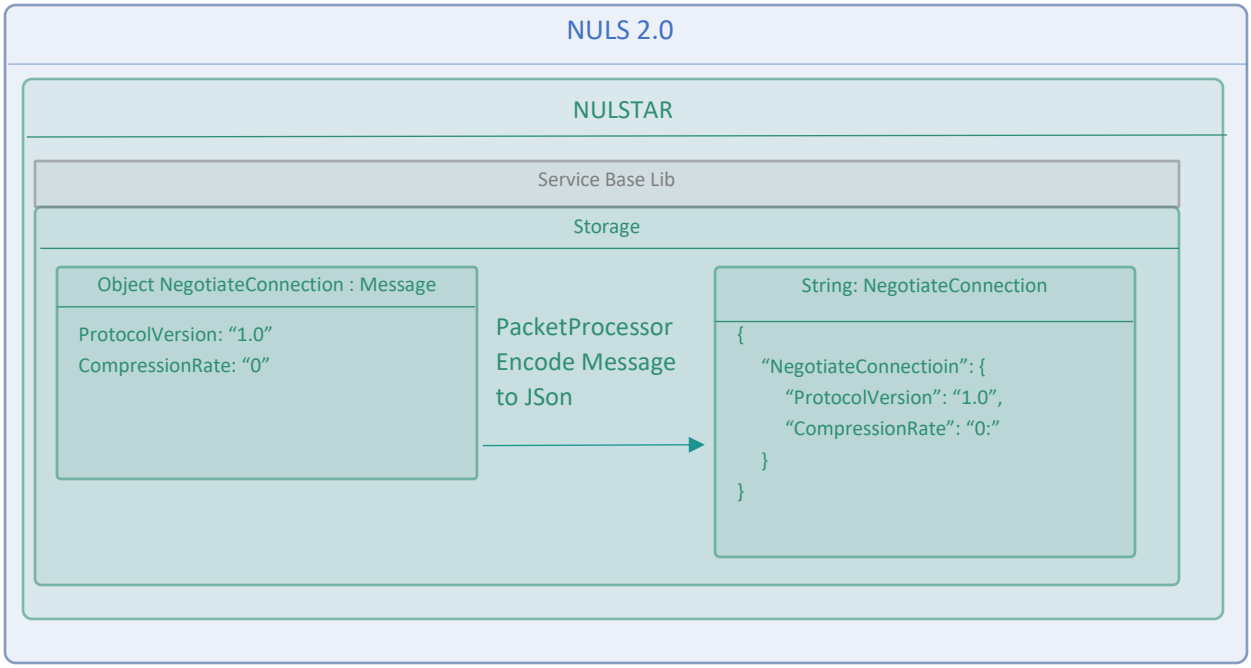
目前定义了 8 种必须要实现的消息类型：

- NegotiateConnection
- NegotiateConnectionResponse
- Request
- Unsubscribe
- Response
- Ack

- RegisterCompoundMethod
- UnregisterCompoundMethod.

请参考 Nulstar - Documentation - Message Protocol 文档获取详细内容.
当一个模块需要发送消息时，它可以实例化对应消息类型的类或结构体，填充对应的值，然后用填充好的额对象作为参数调用 PacketProcessor 完成选用报文格式编码，PacketProcessor 获取对象中数值，按照选用的格式（这里是 JSon）来编码。将来添加像 XML 或二进制格式的支持。

如下图：NegotiateConnection 的数据对象传给 PacketProccesor 完成 Json 格式的编码



3.1.5] 按 JSon 规范格式解码和处理收到的消息报文

依据消息类型解码和处理收到的消息报文:

- NegotiateConnection: 模块需要应答 NegotiateConnectionResponse 类型消息，如果和本模块协议兼容并且支持发送用 zlib 算法按指定压缩级别的消息，那么应答消息中的 NegotiationStatus 设置为 1，否则设置为 0。将来需要在这个消息中扩展支持用户鉴权机制。
- NegotiateConnectionResponse: 当模块收到这个类型消息，它应当检查 NegotiateConnection，并发送 Log Request 给 Storage 模块（详见 3.1.12）。如果协商成功，这个链接必须标记成可以处理 requests/responses，否则每隔 10 秒尝试重试 2 次，如果都尝试协商失败上报 Manager 模块对端模块无法工作。
- Request: 模块收到该消息，必须解码和处理消息中的 RequestMethods 数组和对应的参数列表。回复 Response 消息，消息中携带请求的信息。如果有一个方法执行失败，则认为 Response 是对请求的方法执行结果失败的应答。如果请求中设置了 Event 或 Period，请求方法需要再次处理，并把结果作为新的 Response 消息返回。
- Response: 各模块定义如何处理该消息。如果没有再要处理的 Response，应该清除 PndingMessages 队列中相应的请求条目（详见 3.1.6 节）

- Notification: 各模块定义如何处理该消息。
- RegisterCompoundMethod: 模块应该能够创建具有指定名称和参数别名的新虚方法，然后发送 API Registration Request 给 Manager 模块以添加虚函数（3.1.10 描述 API Registration Request）
- UnregisterRegisterCompoundMethod:模块应该能够移除具有指定名称和参数别名的虚方法，然后发送 API Registration Request 给 Manager 模块以移除该虚函数（3.1.10 描述 API Registration Request）

3.1.6] 管理等待应答的请求队列

创建请求等待队列记录哪些请求还未应答完毕，队里中记录了请求的 ID，对调用者对象的引用和上次响应的时间（使用场景：如果发送的请求中的 Event 或 Period 值不为 0，那么需要等待多个 Response 消息）。如果没有再要处理的 Response，应该清除队列中相应的请求条目。

3.1.7] 严重事件日志硬盘归档

当模块执行异常并且无法发送日志请求给 Storage 模块，它需要记录严重错误日志文件 Error.log 到硬盘，日志内容格式如下：

Date|Time|Time Zone|EventLevel|Id|Critical error description

3.1.8] 收集模块 API 的方法和方法参数

基类应该能够收集遵循使用 introspection 技术 API 的模块方法，每个方法应该用下面三种指定类型之一标记：

- Public:该方法内部和外部应用都可以使用
- Admin:该方法内部模块可以直接使用，外部应用必须通过 Connector 模块的 Admin Web Socket 服务端连接后使用
- Protected:该方法只内部模块使用

3.1.9] 给每个 API 方法添加元信息（Meta），包括方法描述，方法参数验证和事件/周期限制

模块可以发送下面三种方法元信息：

- 方法描述：提供方法的描述或帮助信息
- 参数验证：可以将数值范围和正则表达式的字符串发送到 Manager 模块，以便在收到请求消息之前执行验证检查
- 事件/周期限制：方法应发回 Response 消息的最小周期（秒）和事件数量。

3.1.10] 连接到 Manager 模块并提供模块的 API，元信息和连接信息

启动模块时，应该向 Manger 模块发送一个调用 RegisterAPI 的 Request 消息，其中包含有关每个方法及其各自元数据的信息，还应附加一个包含所需角色及其最小 API 版本的字符串数组。

如下图:


```
{
  “Request”: {
    “RequestID”: “sdj8jcf8899ekffEFefee”,
    “RequestInternalID”: “348022847492”,
    “RequestDate”: “2018-11-05”,
    “RequestTime”: “03:00:00”,
    “RequestTimeZone”: “-4”,
    “SubscriptionEventCounter”: “0”,
    “SubscriptionPeriod”: “0”,
    “ResponseMaxSize: “0”,
    “RequestMethods”: [
      {

          "RegisterAPI": {
             "ServiceAPIVersion": "0.1.0",
             "ServiceDomain": "nuls.io",
             "ServiceName": "NULS Connector",
             "ServiceRole": "Role_ConnectionManager",
             "ServiceVersion": "0.0.1.1",
             “RoleDependencies”: [ { “Role_StorageManager”, “0.1.0” } ],
             “ServiceIP”: “130.34.32.44”,
             “ServicePort”: “7775”,
             "Methods": [
              {
               "MethodDescription": "Sets the maximum number of client connections that should be accepted.\n
                    Parameters:\n  maxconnections [0- ]: Maximum connections allowed. 0 means no limit.",
               "MethodMinEvent": "0",
               "MethodMinPeriod": "0",
               "MethodName": "setmaxconnections",
               "MethodScope": "admin",
               "Parameters": [
                {
                  "ParameterName": "IMaxConnections",
                  "ParameterType": "int",
                  “ParameterRange”: “1, 50”,
                  “ParameterRangeMinIncluded”: “1”,
                  “ParameterRangeMaxIncluded”: “1”,
                  “ParameterRegularExpValidator”: “”
                }
               ]
              },
             {
                ……
                ……
                   ……
                   ……
```

3.1.11]连接 Storage 模块记录日志

如果 RoleDependencies 包含 Storage 模块角色作为依赖项（“Role_StorageManager”），则应在注册 API 时自动执行此步骤

3.1.12]发送非严重级别的事件日志信息给 Storage 模块

当模块想要记录事件时，必须将类型为 LogEvent 的请求发送到 Storage 模块。

有如下五种日志事件级别:

- CriticalError = 1
- ImportantError = 2
- Warning = 3
- Info = 4
- Everything = 5

如下:

```
{
  “Request”: {
    “RequestID”: “AEsdj8jcf88d3fEFefee”,
    “RequestInternalID”: “348022847492”,
    “RequestDate”: “2018-11-05”,
    “RequestTime”: “03:00:00”,
    “RequestTimeZone”: “-4”,
    “SubscriptionEventCounter”: “0”,
    “SubscriptionPeriod”: “0”,
    “ResponseMaxSize: “0”,
    “RequestMethods”: [
      {
        "LogEvent": {
          "LogEventLevel": “4”,
            “LogSourceMessageType”: “Request”
          “LogSourceMessageID”: “”
          "LogSourceModule": "NULS Connector",
          "LogComments”: “Incoming connection accepted from IP: 190.190.1.1!”
        }
      }
    ]
  }
}
```

3.1.13]使用 Manager 模块返回的连接信息，根据需要协商建立与其他模块的 WebSocket 连接

注册 API 后将会收到提供所需 API 角色模块的连接信息 （更多细节见 4.2 节）。该模块应与这些其他模块建立连接。

3.2] 控制器（Controller）– 可执行程序 Nulstar

控制器（Controller）模块是系统的主执行程序，所以它依赖的库必须和其在同一目录下。由于在其他服务器上需要安装其他模块，所以在这些服务器上也需要部署控制器实例，这样控制器才能无缝管理所有模块。这些控制器实例必须和主控制器实例进行通信来保证所有模块都能被主控制器控制。主控制器实例必须和服务管理（Service Manager）模块部署在同一服务器上。

【译注：这里的控制器和 SDN 里的控制器不是同一概念】

可以把控制器模块注册为主机操作系统的一个服务，这样就可以随系统自动启动。

控制器的主要功能：

- 从配置文件 Nulstar.ncf 中加载控制器模块启动需要的参数
- 读取每个需要部署模块的配置文件
- 设置要部署模块的启动参数和依赖库的环境变量（路径等），然后启动这些要部署的模块
- 提供启动和停止模块的命令 API
- 在启动模块时，从官网库下载模块的可执行文件和依赖的库文件以及描述文件，依据描述文件来校验可执行程序文件和库文件的哈希值
- 非主控制器实例需要连接主控制器实例，接受处理主控制器实例下发的命令

3.2.1] 从 Nulsart.ncf 加载参数

控制器配置文件 Nulstar.ncf 定义控制器的相关参数。配置文件采用 Qt 标准的设置文件格式。如下：

```
[Network]
AllowedNetworks=127.0.0.1/32
CommPort=7770
MainControllerURL=127.0.0.1:7770

[Output]
LogLevel=4

[Security]
SslMode=0
HashFile=
HashFileAlgorithm=SHA256
```

AllowedNetworks:指定允许连接控制器模块的网络地址，无类别域间路由（CIDR:Classless Inter-Domain Routing）格式

CommPort:控制器监听端口号

MainControllerIP:当前部署的控制器如果不是主控是器，必须设置该参数为主控制器的地址；如果是主控制器，该参数设置为还回地址（loopback address），如: 127.0.0.1

LogLevel:需要日志记录的事件类型，有五个级别：1 严重，2 重要，3 告警，4 通知，5 所有事件

SslMode:如果值为 0，WebSocket 不使用加密通道通讯，否则每个模块都需要使用证书和控制器建立 WebSockets 加密通道进行通讯

HashFile:如果为空启动模块时不需要校验哈希值，否则需要对每个文件进行哈希计算后和文件哈希列表进行比对校验（详见 3.2.5 节）

HashFileAlgorithm:哈希校验使用的哈希算法，默认为 SHA256

3.2.2] 读取每个需要部署模块的配置文件

部署模块文件路径遵循下面约定：

Modules/<Namespace>/<Module Name>/<Version>

例如 Maganer 模块的部署文件目录为：

Modules/Nulstar/Manager/1.0.0

Nuls 2.0 特定模块应当部署文件目录为：

Modules/Nuls/<Module Name>/<Version>

在每个模块根目录下都有一个配置文件 Module.ncf，其定义模块的启动参数

比如下面文件内容展示了 Manager 模块的启动参数配置：

Module Configuration Example

```
[Core]
Language=CPP
Managed=1

[Libraries]
Nulstar=1.1.3
Qt=5.12.2

[Network]
AllowedNetworks=127.0.0.1/32
CommPort=7771

[Output]
LogLevel=4

[Security]
SslMode=0
```

Language:模块的开发语言，控制器使用该参数定位该模块对应的库文件。

Managed:如果该模块需要控制器管理需要设置为 true，如果设置为 false，模块需要手动命令完成启停

Nulstar:模块所使用的 Nulstar 库版本号

Qt:模块所使用的 Qt 库版本号

AllowedNetworks: 指定允许连接该模块的网络地址，无类别域间路由（CIDR:Classless Inter-Domain Routing）格式

CommPort:该模块监听端口号

LogLevel:需要日志记录的事件类型，有五个级别：1 严重，2 重要，3 告警，4 通知，5 所有事件

SslMode:如果值为 0，WebSocket 不使用加密通道进行通讯，否则每个模块都需要使用证书和该模块建立 WebSockets 加密通道进行通讯

3.2.3] 为模块启动设置参数和库路径环境变量

控制器需要启动的第一个模块是 Manager 模块，Manager 模块和控制器必须部署在相同服务器上。控制器模块必须存储 Manager 接受连接的 URL。

Manager 和其他的 Nulstar 模块的路径目录为 Modules/Nulstar/。

对应的库文件路径遵循 Libraries/<Language>/<Library Group Name>/<Version>路径格式

例如：

Libraries/CPP/Nulstar/1.0.2/

Libraries/CPP/Nulstar/1.3.1/

Libraries/CPP/Qt/5.12.1/

Libraries/Java/Lib1/1.0.2/

模块启动参数要通过‘getopt’函数统一转为小写格式（“-” 跟短参数名，“--”后跟长参数名）

模块启动参数分为三类：

- 环境变量（Environment variables）:设置和当前模块对应版本的库文件路径，这些变量必须由操作系统指定。配置文件 Module.ncf [Core] 和 [Libraries]段描述了怎么组织正确的 PATHs 内容
- Manager URL: 模块启动时必须指定，这样模块才能向 Manager 注册 API，并完成和其他依赖模块的连接
- 其他参数：Module.ncf 中的其他参数必须传递给启动模块

例如：Linux 上启动 Manager 模块的传入参数：

```
LD_LIBRARY_PATH=$(pwd)/Libraries/CPP/Qt/5.12.1/;$(pwd)/Libraries/CPP/Nulstar/1.0.2/
$(pwd)/Modules/Nulstar/Manager --managerurl 127.0.0.1:7771 --loglevel 4 --sslmode 0 --commport 7771 --allowednetworks 127.0.0.1/32
```

3.2.4] 提供模块启动和停止的命令 API

控制器必须像其他模块一样要注册自己的 API，必须提供下面四个管理员权限的基本函数：

- startallmodules
- startmodule <Module Name>
- stopallmodules
- stopmodule <Module Name>
- registermodules

3.2.5] 对要启动模块执行文件和库文件的下载和哈希校验

为了提升系统安全，如 3.2.1]描述对控制器要启动的模块文件和依赖的库文件可以进行哈希校验，下面是一个 HashFile 的例子：

HashFile Example

```
[Libraries]
NNetwork_CPP_1.1.0=2ccde07e9eaadc1b06a2fcd276a2364baf04772d5c0567d8a3d333cb3082b6d1
NNetwork_CPP_1.2.0=99196929374e067a05b4f97f3e61beb4d987e5c8f4546119a036f6d86e2e929a
NNCore_Java_1.3.1=36a7da47692860955aed337b0dd56e4a134f9a08711785d5fd8a67bf50da2092

[Modules]
Nulstar/Manager=b9f2c2bb05237a710856625483c51a712ced4fc10c0c13b3d432a815db2f6a26
Nulstar/Connector=391ff8a395bad8b267ab9604b2827507a140ab940b7f905a3bc8b6465be05135
Nuls/Block=ff6956640981b7a22bcb13beb3d37d7d9a70551657fd75c7a3a3c67b897197b5

[Security]
Hash=9d19be92a55355e5bcd2bb3fb760a7bc94404c45623e4ef2aca1a07453ab4177
```

对所有的模块和库文件求得的哈希值作为 HashFile 中 Hash 参数。Nulstar 中嵌入一个密钥（随机数），这样来保证这个 HashFile 不被人为修改。

控制器在执行模块文件前需要依据 HashFile 来校验模块文件和库文件的哈希值，如果校验失败该模块不能启动，并且记录严重级别的日志。

3.2.6] 连接主控制器实例以便于接收命令

在不同的服务器上部署模块时需要在这些服务器上部署控制器实例，这样控制器可以无缝管理所有模块。

这些拷贝部署的控制器实例不需要向 Manager 注册 API，主控制器实例负责注册。这些控制器实例必须将连接到它的模块列表和版本号通知给主控制器，必须通过发送携带相关信息的 Request 消息来完成。如下：

```
{
  ……
  “MessageType”: “Request”,
  “MessageData”: {
      “RequestAck”: “0”,
      “SubscriptionEventCounter”: “0”,
      “SubscriptionPeriod”: “0”,
      “SubscriptionRange”: “0”,
      “ResponseMaxSize: “0”,
      “RequestMethods”: {
          “RegisterModules”: {
              “Block”: “1.3.4”,
              “Transaction”: “2.3.4”,
          },
       }
   }
}
```