Noodle Engine V2

简介

Noodle 的设计目标是实现一个通用的分布式服务端程序框架，将网络、业务逻辑、数据存储等部分分离，提供一个基于消息传递的异步处理系统，动态加载且分布式远程消息传递，可进行热部署，最终实现为 7×24 小时不间断运行的系统提供框架级支持。

Noodle 可大体划分为以下几部分:

1. 引擎核心
   a) 引擎
   b) 消息队列
   c) 线程池
   d) 内存池
   e) 组件管理器
   f) 代理管理器
   g) 消息源管理器
   h) 全局对象管理器
   i) 日志
   j) 配置管理器
   k) 提供给组件的运行环境调用接口
2. 控制终端
3. 组件

   组件从功能上划分为：消息接入组件和消息处理组件。

   a) 核心组件（已实现）
      i. Plugin FastCGI

         利用 FastCGI 协议与 Web Server 互联。

      ii. Plugin Console

         Noodle Engine 终端组件，开启后可利用 Noodle Console 操作和获取系统信息。

      iii. Plugin Stub

         远程消息传递存根（桩）组件。

      iv. Plugin Proxy

         远程消息传递代理组件。

   b) 用户组件

所有用户的逻辑都必须在组件中实现，通过控制终端，任何组件都可在不停机的情况下被替换/更新/禁用/开启，替换/更新/禁用/开启的过程对最终用户是透明的。在系统的运行过程中，也可以动态加载/卸载新的组件，即，动态扩展/裁剪系统的功能。

互相通信的组件使用自定义消息体格式，运行在不同机器上的组件，可以通过 stub 和 proxy 实现消息的互通，只需要通过配置文件，即可实现赋予无网络功能的组件网络消息传递的能力，传递的过程对组件是透明的，使单机系统平滑的过度为分布式系统。

系统架构

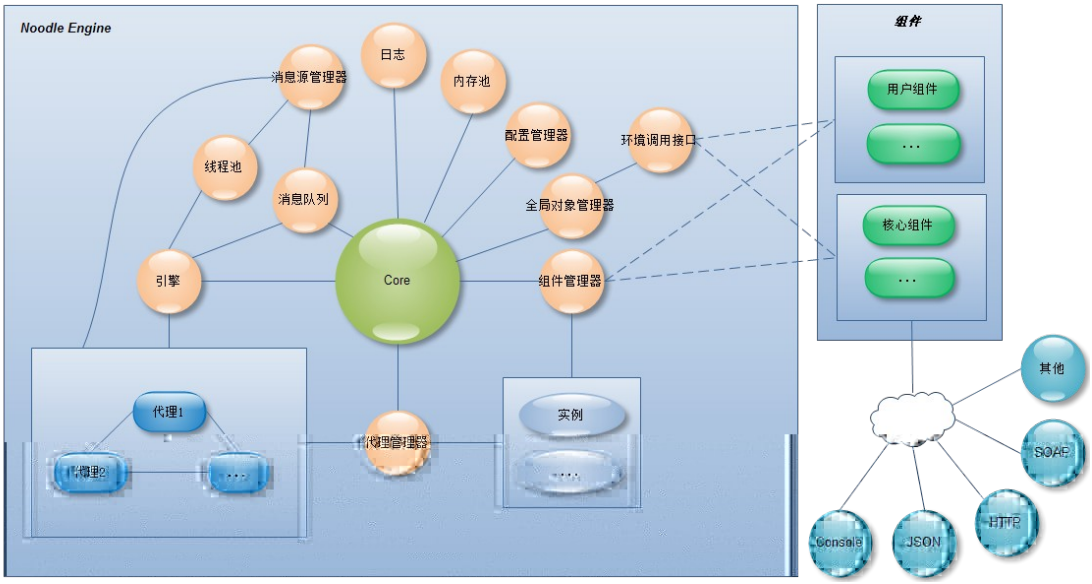


Illustration 1: 架构图

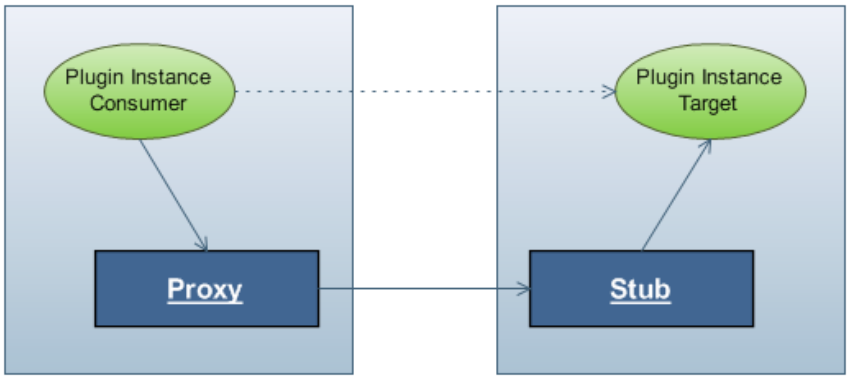
§

Core 作为整个框架的起始点，在启动时，Core 会启动所有的全局对象，并会将这些全局对象注册到全局对象管理器，全局对象管理器在系统中有唯一的实例，任何全局对象都可以通过全局对象管理器访问其他全局对象。

框架通过环境调用接口为组件提供与框架交互的能力，比如：内存池内存分配、消息传递、配置文件读取等功能，每个组件在加载后会产生一到多个实例，每个实例会对每个对它的引用产生一到多个代理，代理是访问组件实例的唯一方法，也是系统热部署功能的基础。

组件必须实现一个预定义的接口，这个接口会在组件的不同生命周期被框架调用，比如：组件加载、卸载等。组件从功能上分为两类：消息源组件和用户逻辑组件。消息源组件是外部消息接入的唯一方法，负责将外部消息规格化和格式转换。用户逻辑组件负责处理由消息源产生的消息。

为方便开发人员使用，框架预定义提供了一些核心组件，这些组件也可以被开发人员所开发的同样功能组件所替换。有两个特殊功能的组件（Proxy、Stub），为实现框架的分布式功能提供了功能性的支撑。



如上图所示，某个做为消费者组件实例，想发送一条消息给目标组件实例，由于目标组件实例与消费者组件不在同一物理机器上，通过系统配置，将一个 Proxy 组件置于消费者同一台物理计算机，将 Stub 组件置于目标同一台物理计算机。消费者的消息被 Proxy 路由到 Stub，再由 Stub 发送给目标组件，整个过程对于消费者和目标是完全透明的。理论上，可以通过配置形成所有组件（不论是否在同一台计算机上）的全连接网，即，任何组件都可以和任何其他组件用消息产生联系。对于小型的集群（几台到几十台），可以做到通过手工配置配置文件的方法来布置整个系统，但是对于大型的集群（几百到几千台），手工去配置每台机器是不现实的，需要配置文件自动生成及分发系统，这将在今后的版本中引入。

运行环境接口

运行环境接口是组件与 Noodle 交互的纽带，运行环境接口提供的方法分类如下（详见附录）：

1. 消息管理
    - 消息的产生
    - 消息的销毁
    - 设置和获取消息属性
    - 获取某个组件的实例 ID
    - 向 Noodle Engine 发送消息
2. 内存池的内存分配与释放
    - 从内存池分配指定大小内存
    - 释放内存到内存池
3. 配置文件读取
    - 读取整数
    - 读取双精度浮点数
    - 读取字符串
    - 读取布尔值
    - 读取数组
    - 读取字典
4. 写日志
    - 写日志

系统安装

系统依赖 BOOST，请先下载并安装 BOOST，按如下流程进行编译安装。

```
$ autoreconf -if
$ ./configure --prefix=YourInstallDIR
$ make
$ make install
```

编译安装完成后，在安装目录的 conf 目录下找到名为 noodle.conf 的配置文件，配置文件

的细节详见附录。

系统启动与终止

- 将当前目录转移至安装目录下的 bin 目录，用下面命令启动系统

  ```
  $ ./ne -c ../conf/noodle.conf
  ```

  此时，系统默认在安装目录下的 pid 目录下，会产生主进程的 PID 文件。

- 关闭系统，假定 PID 文件名为 noodle.pid，此文件可在配置文件中配置

  ```
  $ ./ne -c ./conf/noodle.conf -p ../pid/noodle.pid -s stop
  ```

- 重启系统

  ```
  $ ./ne -c ./conf/noodle.conf -p ../pid/noodle.pid -s restart
  ```

- 重新加载配置文件

  ```
  $ ./ne -c ./conf/noodle.conf -p ../pid/noodle.pid -s reload
  ```

- 以后台进程方式启动

  ```
  $ ./ne -c ../conf/noodle.conf -s start -d
  ```

- 指定日志文件路径和日志级别

  ```
  $ ./ne -c ../conf/noodle.conf -s start -l ../logs/noodle.log -e 5
  ```

  日志文件路径和级别也可在配置文件中配置。

- 启动多个系统实例
  为每个实例提供不同的配置文件和日志文件，如果用同一个配置文件启动多个实例，多个 console 组件会监听同一个服务端口，导致启动失败。

  ```
  $ ./ne -c ../conf/noodle0.conf -s start -l ../logs/noodle0.log
  $ ./ne -c ../conf/noodle1.conf -s start -l ../logs/noodle1.log
  ```

  以上两行命令，启动了两个不同的系统实例。

- 关闭所有系统实例

```
$ killall -15 ne
```

- 启动 Console
  在安装目录的 bin 目录下。

  ```
  $ ./console 127.0.0.1 12345
  ```

  连接成功后会看到欢迎信息，输入 help 可看到详细命令行语法。

编写组件

编写任何 Noodle 组件都必须实现一个预定义的接口，并同时声明一个宏。系统提供了一个简单包装过的消息源组件类，可以通过继承和扩展它的功能来实现消息源组件。

消息源组件

TBD.

逻辑组件

TBD.

Q&A

TBD.

附录

配置文件详解

```
ne {
  system {
    # PID(Process Identifier)文件路径
    pid = '../pid/noodle.pid'
    # 是否以守护进程方式启动
    daemon = true
    # 内存池相关设置
    pool {
      max = 1024000000  # 内存池所能占用操作系统内存的最大值
      begin = 4         # 内存池所能分配内存大小的最小值
      end = 1024000     # 内存池所能分配内存大小的最大值
```

```
        gap = 8                 # 内存池内，相临内存块大小之差
        factor = 0.2            # 内存池内，内存块预分配系数
        singleChunk = true      # 是否禁用预分配
      }
      engine {
        # 引擎工作线程数量，一般与CPU核心数量相同
        worker = 2
      }
      log {
        # 日志文件路径
        name = '../logs/noodle.log'
        /*
         ERG   = 1
         ALERT = 2
         ERR   = 3
         WARN  = 4
         NOTI  = 5
         INFO  = 6
         DEBUG = 7
        */
        level = 5     # 日志级别
        flush = true  # 是否为每条写入刷新输出缓冲区
      }
    }
    plugin {
      # 组件目录
      path = "../plugins"
      # 组件配置
      plugins =
      (
        # <组件名称> => [ <组件library名称>, <是否是消息源>, <所生成的实例> ]
        "plugin3"  => [ "libfcgitest.so",      false, [ 3, 10, 11, 12, 13 ] ],
        "console"  => [ "libpluginconsole.so", true,  [ 50  ] ],
        "fastcgi"  => [ "libfastcgi.so",       true,  [ 100 ] ]
      )
    }
}
console {
    ip   = "127.0.0.1" # 所监听的IP地址
    port = 12345       # 端口号
}
fastcgi {
    ip   = "127.0.0.1"          # 所监听的IP地址， Web Server的配置中与这个IP相同
    #domain = "whatever";    # 保留
```

```
    port = 30000                # 端口号
    concurrent = 4              # 并发线程数
    backLog = 1000              # 等待队列长度
    receiver = "plugin3"        # 接收 FastCGI 属性的组件名称
    maxBufferLength = 10240 # 每个线程的接收/发送缓冲区大小
}
```

Noodle Engine 环境接口

TBD.

组件接口

TBD.