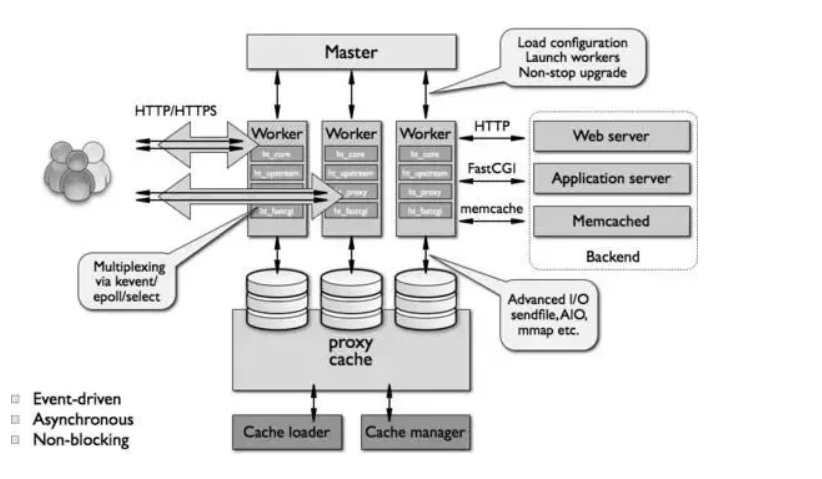
# Nginx整体架构



## 1.1 主进程

Nginx 启动时，会生成两种类型的进程，一个是主进程 （ master ）， 一个 （ windows版本的目前只有一个）或 多个工作进程 （ worker ）。

主进程并不处理网络请求，主要负责调度工作进程 ，也就是图示的 3 项：

1. 加载配置
2. 启动工作进程
3. 非停升级

因此，Nginx 启动以后，查看操作系统的进程列表，我们就能看到至少有两个Nginx 进程。

## 1.2工作进程

服务器实际处理网络请求及响应的是工作进程 （ worker ），在类 unix 系统上， Nginx可以配置多个worker ，而每个 worker 进程都可以同时处理数以千计的网络请求

## 1.3 模块化设计

Nginx的worker进程，包括核心和功能性模块，核心模块负责维持一个运行循环(run-loop)，执行网络请求处理的不同阶段的模块功能。

比如：网络读写，存储读写，内容传输，外出过滤，以及将请求发往上游服务器等。

而起模块化的设计，也使得我们可以根据不同的需求，进行特定的选择和修改，编程成具有特定功能的服务器

## 1.4. 事件驱动模型

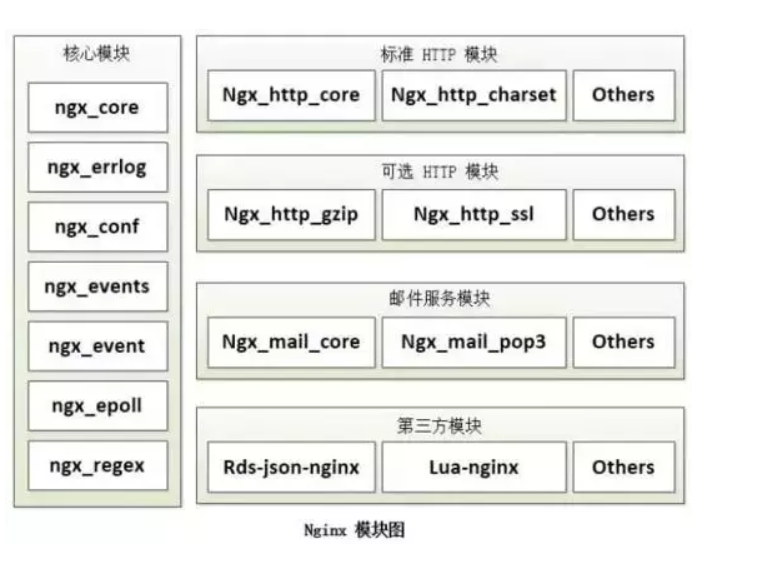
基于 异步及非阻塞的事件驱动模型 ，可以说是 Nginx 得以获得高并发、高性能的关键因素，同时也得益于对 Linux 、 Solaris 及类 BSD 等操作系统内核中 事件通知 及 I/O 性能增强功能 的采用，如 kqueue 、 epoll 及 event ports 。

## 1.5. 代理（proxy）设计

代理设计，可以说是 Nginx 深入骨髓的设计，无论是对于 HTTP ，还是对于 FastCGI 、 Memcache 、 Redis 等的网络请求或响应，本质上都采用了 代理机制 。所以， Nginx 天生就是高性能的 代理服务器 。

# 2. Nginx的模块化设计

高度模块化的设计是 Nginx 的架构基础。 Nginx 服务器被分解为多个模块 ，每个模块就是一个功能模块 ，只负责自身的功能，模块之间严格遵循 “高内聚，低耦合” 的原则。



## 2.1. 核心模块

核心模块是 Nginx 服务器正常运行 必不可少的模块，提供错误日志记录 、 配置文件解析 、 事件驱动机制 、 进程管理 等核心功能。

## 2.2. 标准HTTP模块

标准 HTTP 模块提供 HTTP 协议解析相关的功能，比如： 端口配置 、 网页编码设置 、 HTTP响应头设置 等等。

## 2.3. 可选HTTP模块

可选 HTTP 模块主要用于 扩展 标准的 HTTP 功能，让 Nginx 能处理一些特殊的服务，比如： Flash 多媒体传输 、解析 GeoIP 请求、 网络传输压缩 、 安全协议 SSL 支持等。

## 2.4. 邮件服务模块

邮件服务模块主要用于支持 Nginx 的 邮件服务 ，包括对 POP3 协议、 IMAP 协议和 SMTP协议的支持。

## 2.5. 第三方模块

第三方模块是为了扩展 Nginx 服务器应用，完成开发者自定义功能，比如： Json 支持、 Lua 支持等。

# 3. Nginx的请求方式处理

Nginx 是一个 高性能 的 Web 服务器，能够同时处理大量的并发请求 。它结合多进程机制和 异步机制 ，异步机制使用的是 异步非阻塞方式 ，接下来就给大家介绍一下 Nginx 的 多线程机制 和 异步非阻塞机制 。

## 3.1. 多进程机制

服务器每当收到一个客户端时，就有 服务器主进程 （ master process ）生成一个 子进程（ worker process ）出来和客户端建立连接进行交互，直到连接断开，该子进程就结束了。

使用 进程 的好处是 各个进程之间相互独立 ， 不需要加锁 ，减少了使用锁对性能造成影响，同时降低编程的复杂度，降低开发成本。

其次，采用独立的进程，可以让 进程互相之间不会影响 ，如果一个进程发生异常退出时，其它进程正常工作， master 进程则很快启动新的 worker 进程，确保服务不会中断，从而将风险降到最低。

缺点是操作系统生成一个 子进程 需要进行 内存复制 等操作，在 资源 和 时间 上会产生一定的开销。当有 大量请求 时，会导致 系统性能下降 。

## 3.2. 异步非阻塞机制

每个 工作进程 使用 异步非阻塞方式 ，可以处理多个客户端请求 。

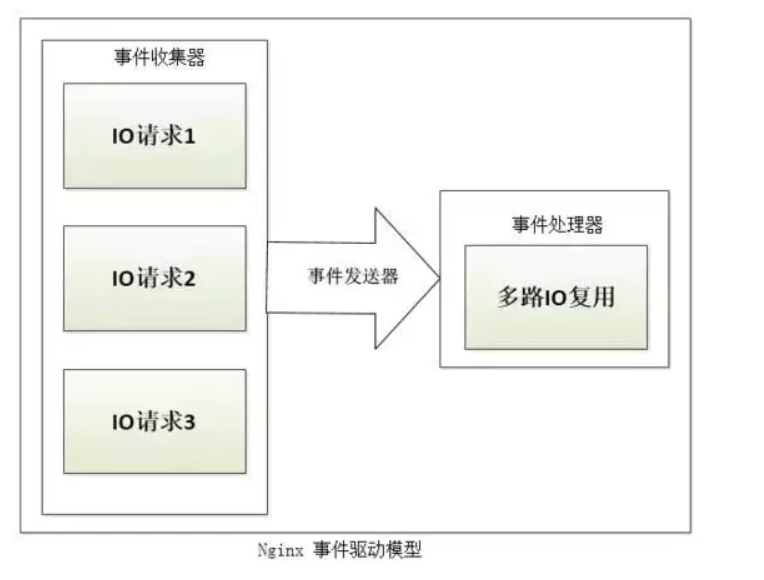
当某个 工作进程 接收到客户端的请求以后，调用 IO 进行处理，如果不能立即得到结果，就去处理其他请求 （即为非阻塞 ），而客户端在此期间也无需等待响应 ，可以去处理其他事情（即为异步 ）

当 IO 返回时，就会通知此工作进程，该进程得到通知，暂时挂起当前处理的事务去 响应客户端请求 。

# 4.Nginx事件驱动模型

在 Nginx 的 异步非阻塞机制 中， 工作进程在调用 IO 后，就去处理其他的请求，当 IO 调用返回后，会通知该工作进程 。

对于这样的系统调用，主要使用 Nginx 服务器的**事件驱动模型**来实现，如下图所示：



如上图所示， Nginx 的 事件驱动模型 由 事件收集器 、 事件发送器 和 事件处理器 三部分基本单元组成。

1. 事件收集器：负责收集 worker 进程的各种 IO 请求；
2. 事件发送器：负责将 IO 事件发送到 事件处理器 ；
3. 事件处理器：负责各种事件的 响应工作 。

事件发送器将每个请求放入一个 待处理事件列表 ，使用非阻塞 I/O 方式调用 事件处理器来处理该请求。

其处理方式称为 “多路 IO 复用方法” ，常见的包括以下三种： select 模型、 poll模型、 epoll 模型。

# 5.Nginx进程处理模型

Nginx 服务器使用 **master/worker** 多进程模式，多线程启动和执行的流程如下：

主程序Master process启动后，通过一个 for 循环来接收和处理外部信号

主进程通过 fork() 函数产生 worker 子进程 ，每个 子进程 执行一个 for 循环来实现 Nginx 服务器 对事件的接收 和 处理

一般推荐 worker 进程数 与 CPU 内核数 一致，这样一来不存在 大量的子进程 生成和管理任务，避免了进程之间 竞争 CPU 资源 和 进程切换 的开销。

而且 Nginx 为了更好的利用 多核特性 ，提供了 CPU 亲缘性 的绑定选项，我们可以将某 一个进程绑定在某一个核上，这样就不会因为 进程的切换 带来 Cache 的失效

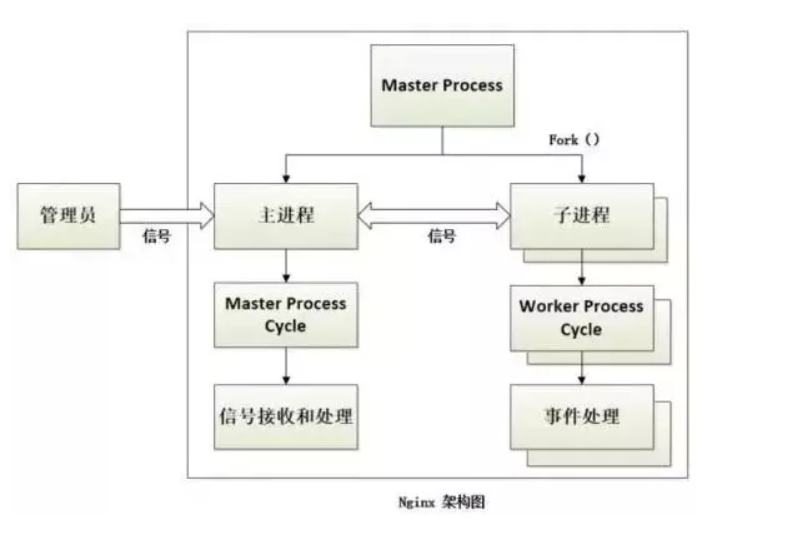
对于每个请求，有且只有一个 工作进程 对其处理。首先，每个 worker 进程都是从 master进程 fork 过来。在 master 进程里面，先建立好需要 listen 的 socket（listenfd） 之后，然后再 fork 出多个 worker 进程。

所有 worker 进程的 listenfd 会在 新连接 到来时变得 可读 ，为保证只有一个进程处理该连接，所有 worker 进程在注册 listenfd 读事件前抢占 accept\_mutex

抢到 互斥锁 的那个进程注册 listenfd 读事件 ，在读事件里调用 accept 接受该连接。

当一个 worker 进程在 accept 这个连接之后，就开始读取请求 ， 解析请求 ， 处理请求，产生数据后，再返回给客户端 ，最后才断开连接 ，一个完整的请求就是这样。

我们可以看到，一个请求，完全由 worker 进程来处理，而且只在一个 worker 进程中处理。



在 Nginx 服务器的运行过程中， 主进程 和 工作进程 需要进程交互。交互依赖于 Socket 实现的管道来实现。

## 5.1. 主进程与工作进程交互

这条管道与普通的管道不同，它是由 主进程 指向 工作进程 的单向管道 ，包含主进程向工作进程发出的指令工，作进程 ID 等。同时主进程与外界通过信号通信 ；每个子进程具备接收信号 ，并处理相应的事件的能力。

## 5.2. 工作进程与工作进程交互

这种交互和 主进程-工作进程 交互基本一致，但是会通过主进程间接完成，工作进程之间是相互隔离的。

所以当工作进程 W1 需要向工作进程 W2 发指令时，首先找到 W2 的 进程ID ，然后将正确的指令写入指向 W2 的 通道，W2 收到信号采取相应的措施。