Python开发之运维架构

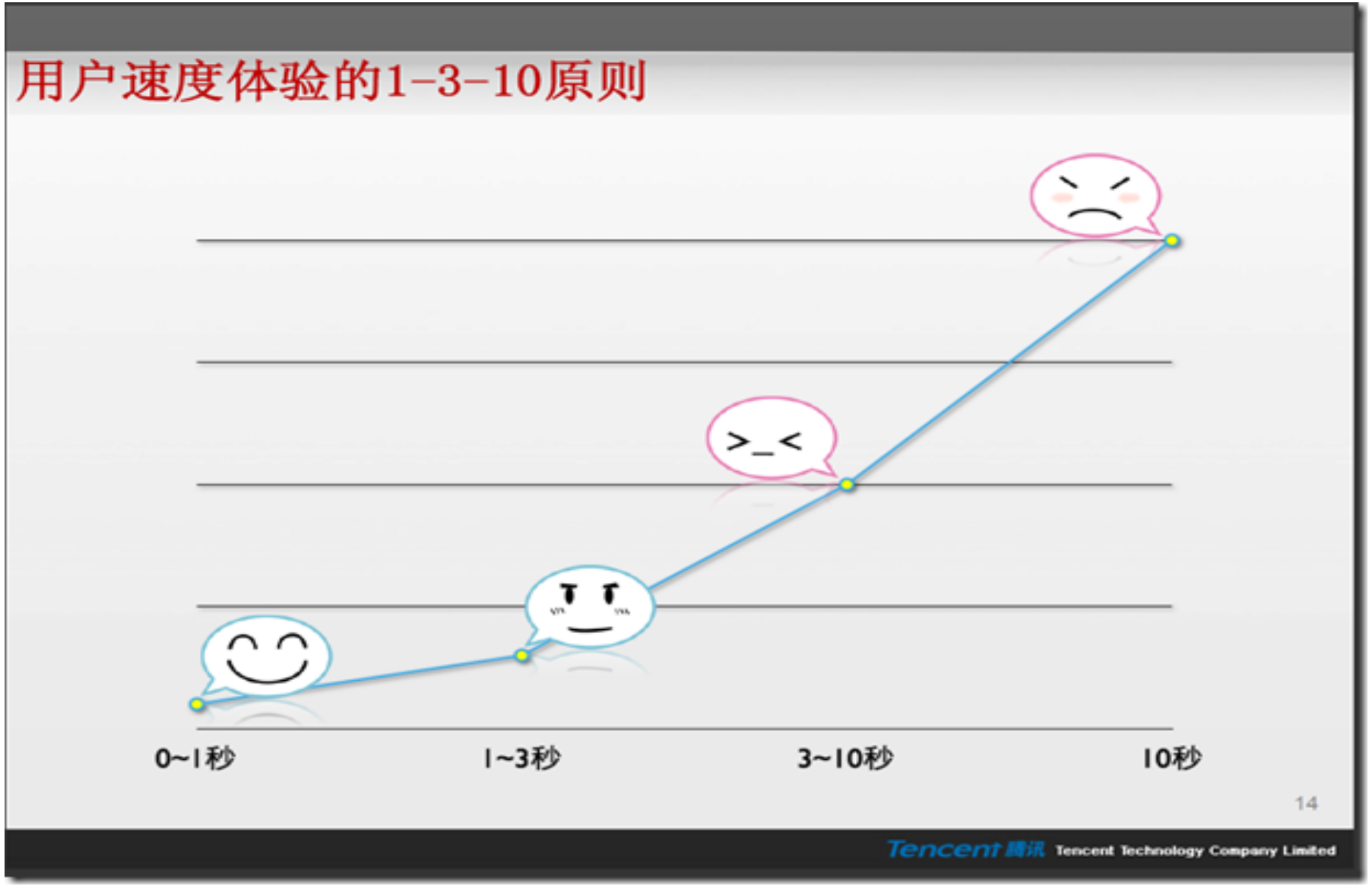
讲师：王晓春

高性能WEB服务器NGINX

本章内容

- I/O模型
- Nginx介绍
- Nginx安装
- Nginx各种模块

用户速度体验的1-3-10原则



性能影响

- 有很多研究都表明，性能对用户的行为有很大的影响：
- 79%的用户表示不太可能再次打开一个缓慢的网站
- 47%的用户期望网页能在2秒钟以内加载
- 40%的用户表示如果加载时间超过三秒钟，就会放弃这个网站
- 页面加载时间延迟一秒可能导致转换损失7%，页面浏览量减少11%
- 8秒定律：用户访问一个网站时，如果等待网页打开的时间超过8秒，会有超过30%的用户放弃等待

Httpd MPM

- httpd MPM：
  - prefork：进程模型，两级结构，主进程master负责生成子进程，每个子进程负责响应一个请求
  - worker：线程模型，三级结构，主进程master负责生成子进程，每个子进程负责生成多个线程，每个线程响应一个请求
  - event：线程模型，三级结构,主进程master负责生成子进程，每个子进程响应多个请求

I/O介绍

- I/O:
  - 网络IO：本质是socket读取
  - 磁盘IO：

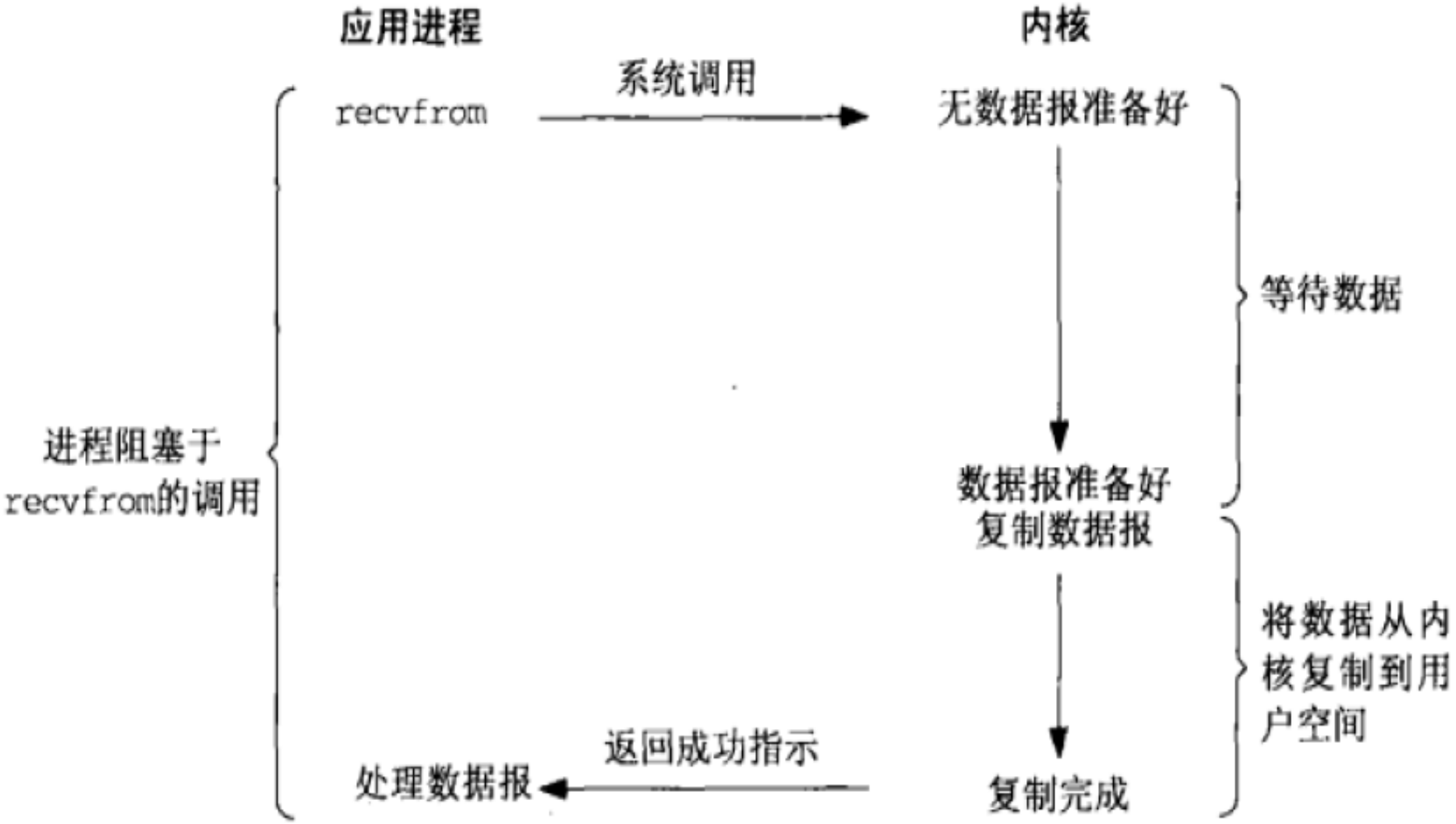
- 每次IO，都要经由两个阶段：
  - 第一步：将数据从磁盘文件先加载至内核内存空间（缓冲区），等待数据准备完成，时间较长
  - 第二步：将数据从内核缓冲区复制到用户空间的进程的内存中，时间较短

I/O模型

- 同步/异步：关注的是消息通信机制
  - 同步：synchronous，调用者自己主动等待被调用者返回消息，才能继续执行
  - 异步：asynchronous，被调用者通过状态、通知或回调机制主动通知调用者被调用者的运行状态
- 阻塞/非阻塞：关注调用者在等待结果返回之前所处的状态
  - 阻塞：blocking，指IO操作需要彻底完成后才返回到用户空间，调用结果返回之前，调用者被挂起
  - 非阻塞：nonblocking，指IO操作被调用后立即返回给用户一个状态值，无需等到IO操作彻底完成，最终的调用结果返回之前，调用者不会被挂起
- I/O模型：

  阻塞型、非阻塞型、复用型、信号驱动型、异步

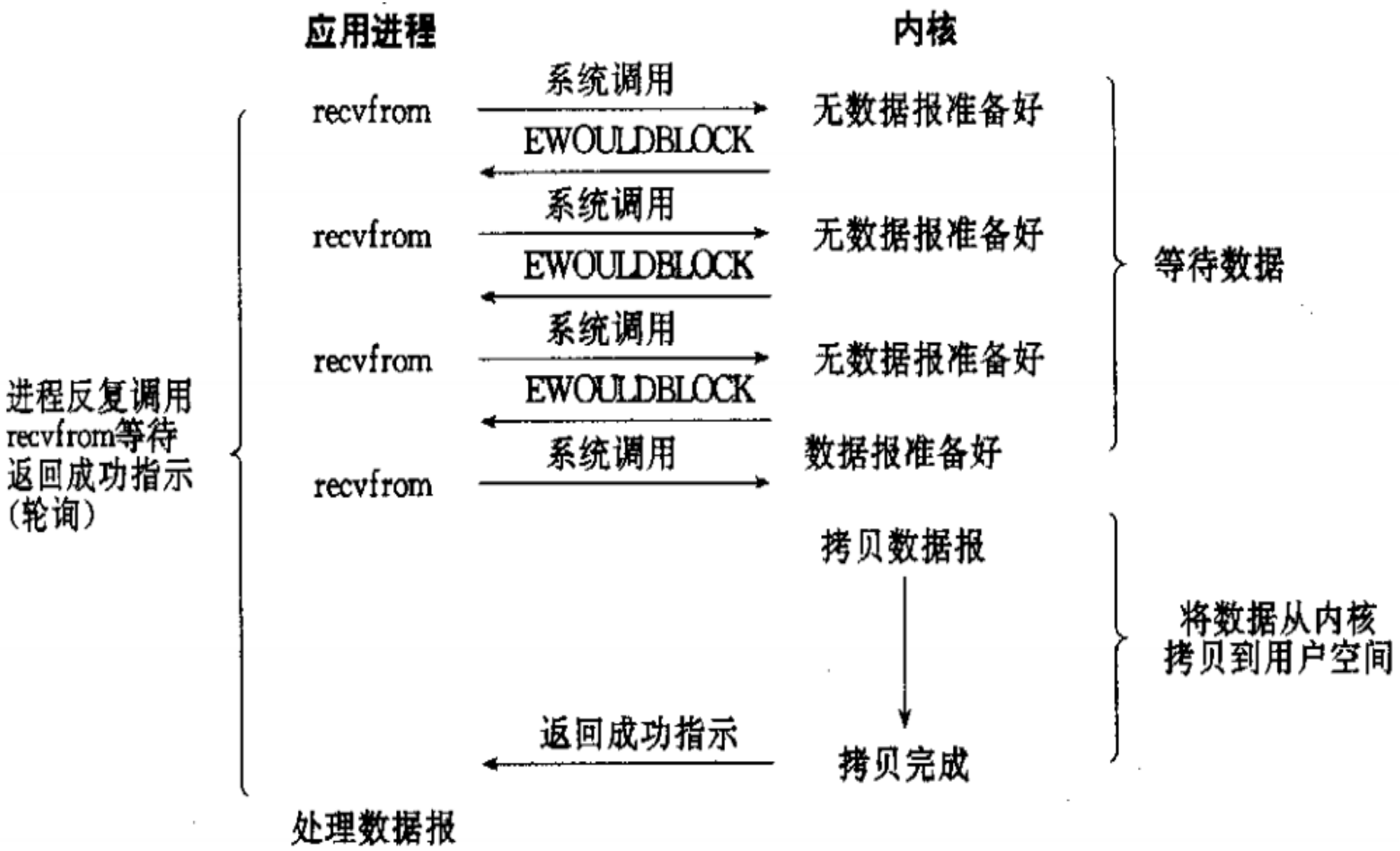
同步阻塞IO模型



同步阻塞IO模型

- 同步阻塞IO模型是最简单的IO模型，用户线程在内核进行IO操作时被阻塞
- 用户线程通过系统调用read发起IO读操作，由用户空间转到内核空间。内核等到数据包到达后，然后将接收的数据拷贝到用户空间，完成read操作
- 用户需要等待read将数据读取到buffer后，才继续处理接收的数据。整个IO请求的过程中，用户线程是被阻塞的，这导致用户在发起IO请求时，不能做任何事情，对CPU的资源利用率不够

同步非阻塞IO模型


应用进程
内核
系统调用
recvfrom
无数据报准备好
EWOULDBLOCK
系统调用
recvfrom
无数据报准备好
EWOULDBLOCK
系统调用
recvfrom
无数据报准备好
EWOULDBLOCK
系统调用
recvfrom
数据报准备好
等待数据
进程反复调用
recvfrom等待
返回成功指示
(轮询)
拷贝数据报
将数据从内核
拷贝到用户空间
返回成功指示
拷贝完成
处理数据报

同步非阻塞IO模型

- 用户线程发起IO请求时立即返回。但并未读取到任何数据，用户线程需要不断地发起IO请求，直到数据到达后，才真正读取到数据，继续执行。即 “轮询”机制
- 整个IO请求的过程中，虽然用户线程每次发起IO请求后可以立即返回，但是为了等到数据，仍需要不断地轮询、重复请求，消耗了大量的CPU的资源
- 是比较浪费CPU的方式，一般很少直接使用这种模型，而是在其他IO模型中使用非阻塞IO这一特性

IO多路复用模型

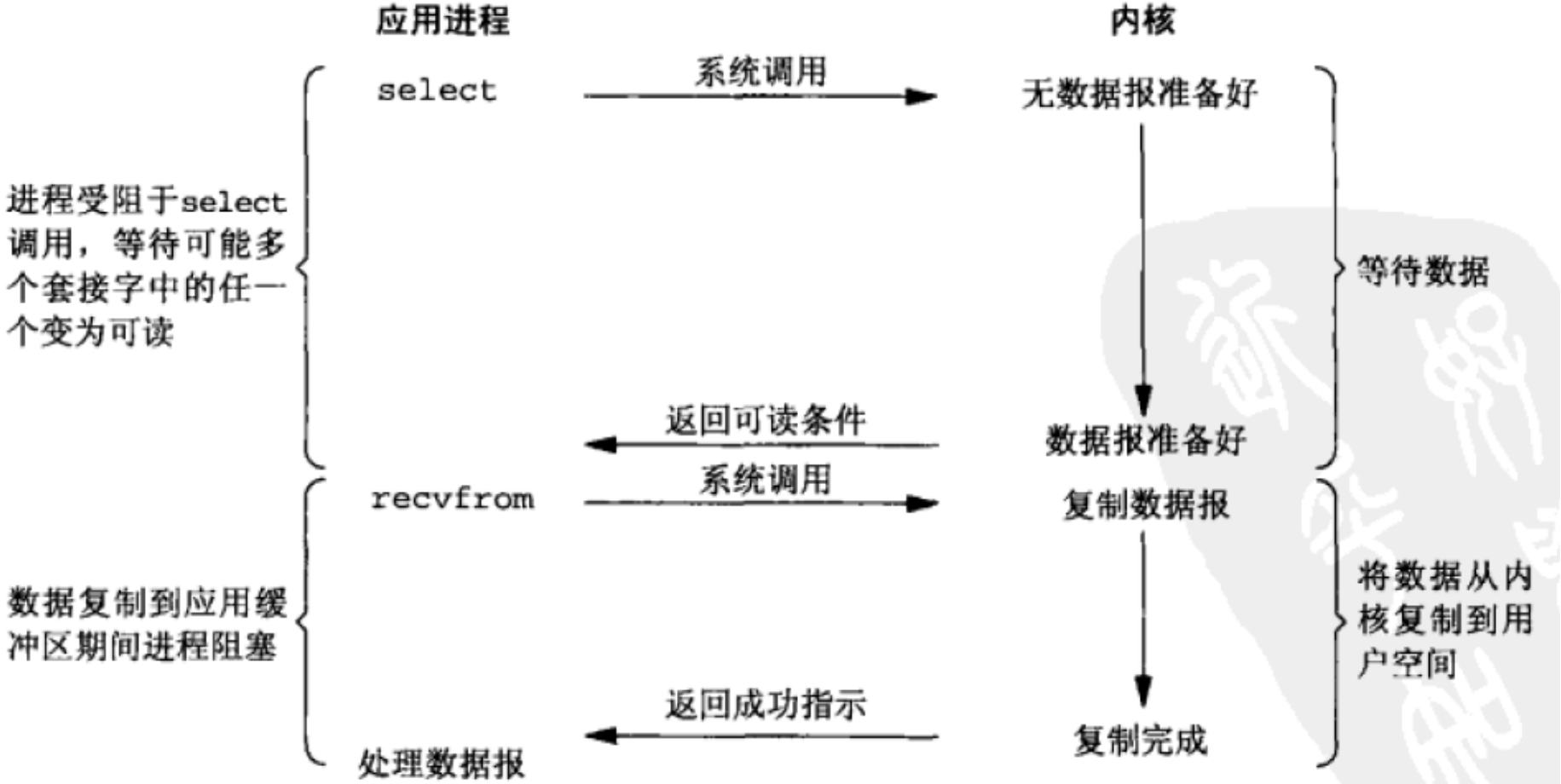


图6-3　I/O复用模型

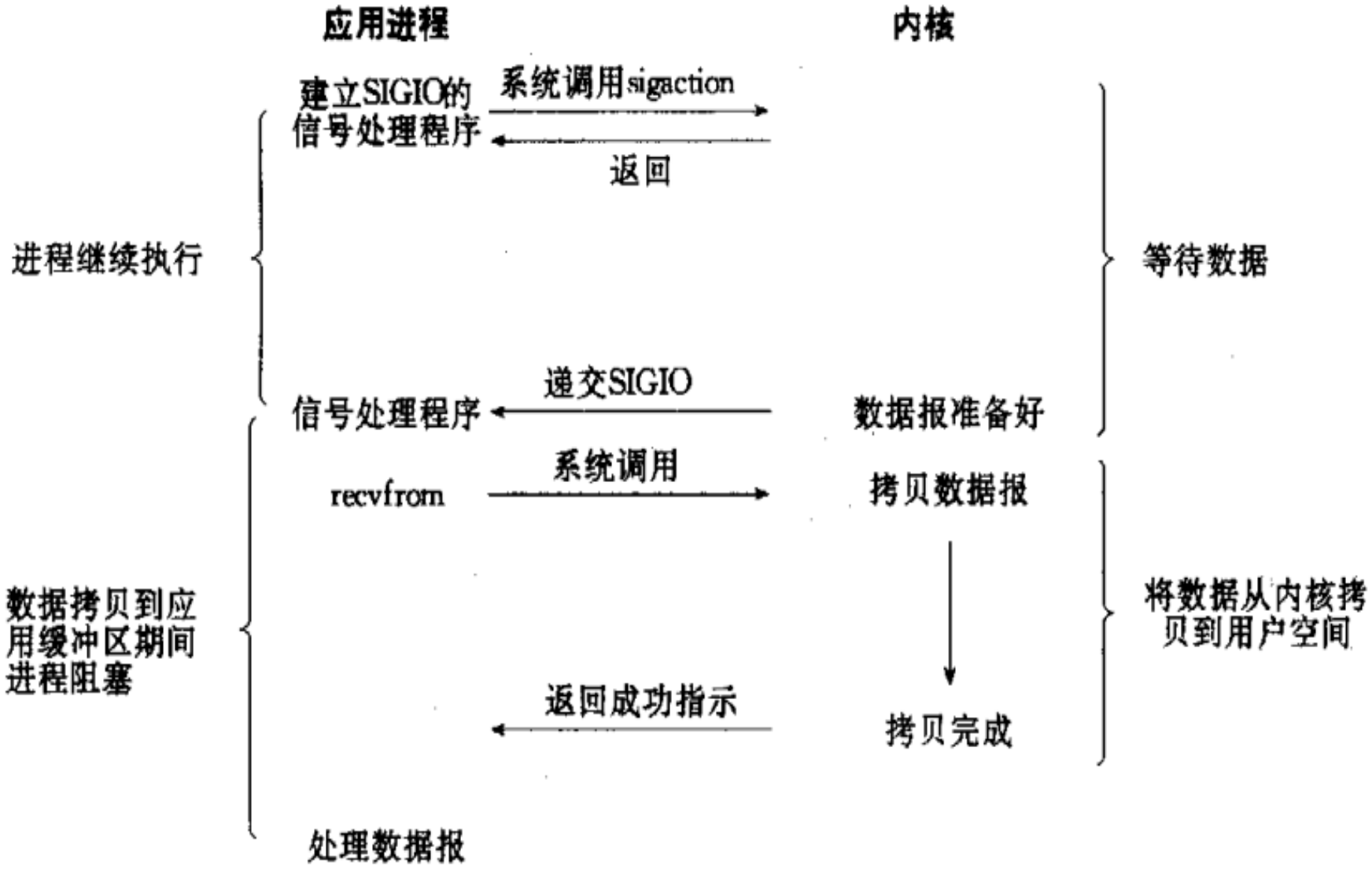
I/O多路复用模型

- 多个连接共用一个等待机制，本模型会阻塞进程，但是进程是阻塞在select或者poll这两个系统调用上，而不是阻塞在真正的IO操作上
- 用户首先将需要进行IO操作添加到select中，继续执行做其他的工作（异步），同时等待select系统调用返回。当数据到达时，IO被激活，select函数返回。用户线程正式发起read请求，读取数据并继续执行。
- 从流程上来看，使用select函数进行IO请求和同步阻塞模型没有太大的区别，甚至还多了添加监视IO，以及调用select函数的额外操作，效率更差。并且阻塞了两次，但是第一次阻塞在select上时，select可以监控多个IO上是否已有IO操作准备就绪，即可达到在同一个线程内同时处理多个IO请求的目的。而不像阻塞IO那种，一次只能监控一个IO
- 虽然上述方式允许单线程内处理多个IO请求，但是每个IO请求的过程还是阻塞的（在select函数上阻塞），平均时间甚至比同步阻塞IO模型还要长。如果用户线程只是注册自己需要的IO请求，然后去做自己的事情，等到数据到来时再进行处理，则可以提高CPU的利用率
- IO多路复用是最常使用的IO模型，但是其异步程度还不够“彻底”，因为它使用了会阻塞线程的select系统调用。因此IO多路复用只能称为异步阻塞IO模型，而非真正的异步IO

多路I/O复用

- IO多路复用是指内核一旦发现进程指定的一个或者多个IO条件准备读取，它就通知该进程
- IO多路复用适用如下场合：
  - 当客户端处理多个描述符时（一般是交互式输入和网络套接口），必须使用I/O复用
  - 当一个客户端同时处理多个套接字时，此情况可能的但很少出现
  - 当一个TCP服务器既要处理监听套接口，又要处理已连接套接口，一般也要用到I/O复用
  - 当一个服务器即要处理TCP，又要处理UDP，一般要使用I/O复用
  - 当一个服务器要处理多个服务或多个协议，一般要使用I/O复用

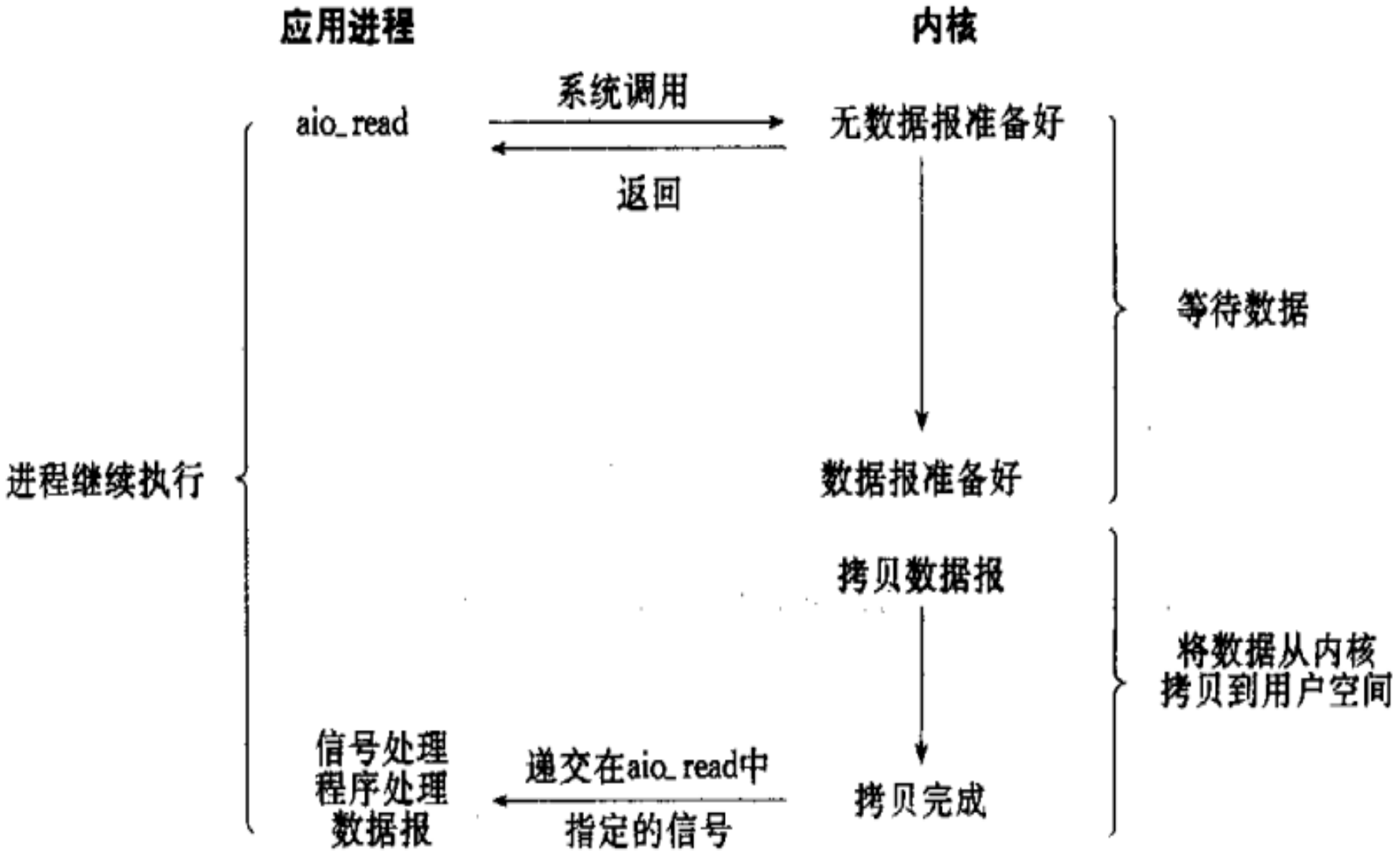
信号驱动IO模型


应用进程
内核
建立SIGIO的信号处理程序
系统调用sigaction
返回
进程继续执行
等待数据
递交SIGIO
信号处理程序
数据报准备好
系统调用
recvfrom
拷贝数据报
数据拷贝到应用缓冲区期间进程阻塞
将数据从内核拷贝到用户空间
返回成功指示
拷贝完成
处理数据报

信号驱动IO模型

- 信号驱动IO：signal-driven I/O
- 用户进程可以通过sigaction系统调用注册一个信号处理程序，然后主程序可以继续向下执行，当有IO操作准备就绪时，由内核通知触发一个SIGIO信号处理程序执行，然后将用户进程所需要的数据从内核空间拷贝到用户空间
- 此模型的优势在于等待数据报到达期间进程不被阻塞。用户主程序可以继续执行，只要等待来自信号处理函数的通知
- 该模型并不常用

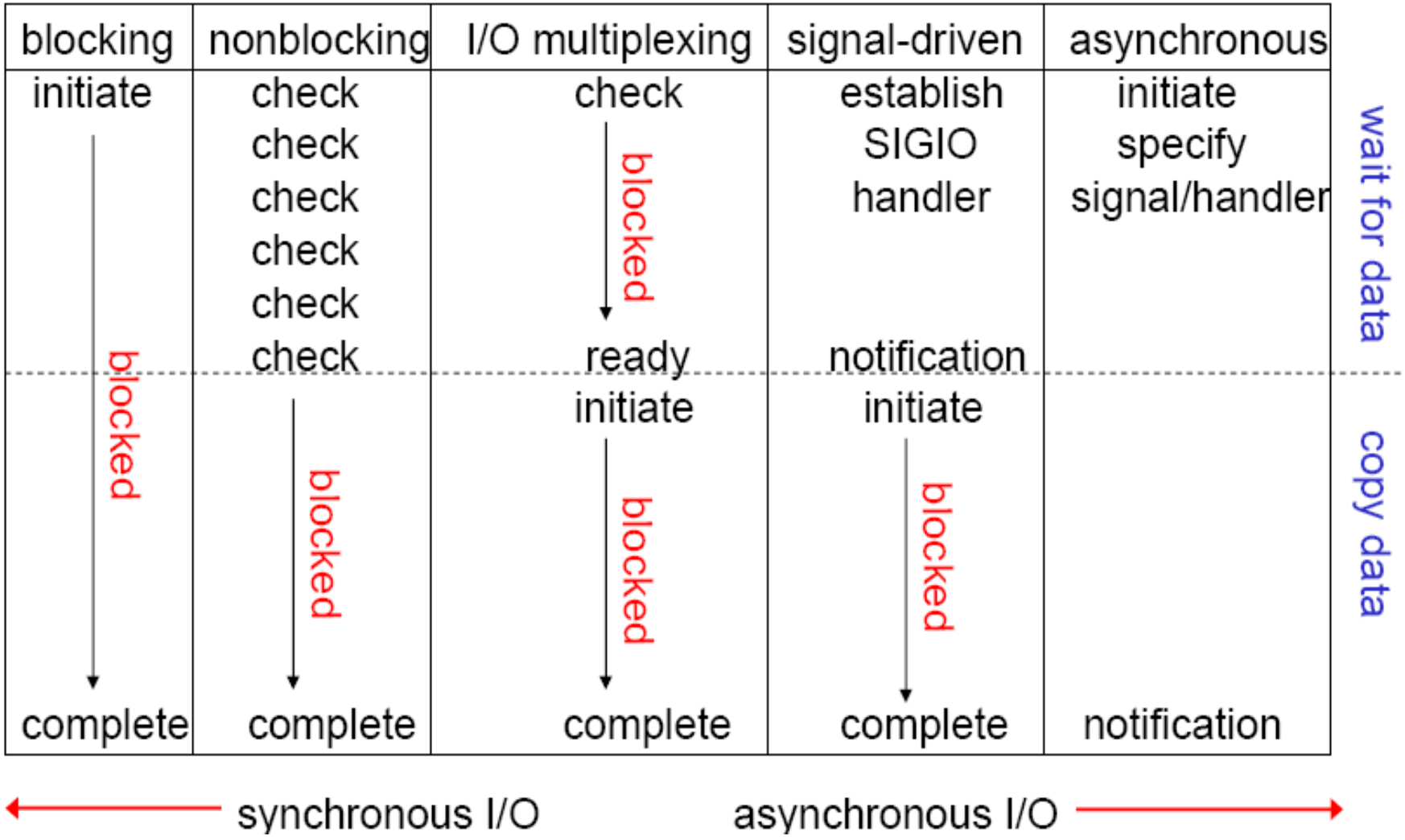
异步IO模型


应用进程
内核
系统调用
aio_read
无数据报准备好
返回
等待数据
进程继续执行
数据报准备好
拷贝数据报
将数据从内核
拷贝到用户空间
信号处理
程序处理
数据报
递交在aio_read中
指定的信号
拷贝完成

异步IO模型

- 异步IO与信号驱动IO最主要的区别是信号驱动IO是由内核通知何时可以进行IO操作，而异步IO则是由内核告诉我们IO操作何时完成了。具体来说就是，信号驱动IO当内核通知触发信号处理程序时，信号处理程序还需要阻塞在从内核空间缓冲区拷贝数据到用户空间缓冲区这个阶段，而异步IO直接是在第二个阶段完成后内核直接通知可以进行后续操作了
- 相比于IO多路复用模型，异步IO并不十分常用，不少高性能并发服务程序使用IO多路复用模型+多线程任务处理的架构基本可以满足需求。况且目前操作系统对异步IO的支持并非特别完善，更多的是采用IO多路复用模型模拟异步IO的方式（IO事件触发时不直接通知用户线程，而是将数据读写完毕后放到用户指定的缓冲区中）

五种I/O模型


blocking
nonblocking
I/O multiplexing
signal-driven
asynchronous
initiate
check
check
check
check
check
check
check
establish
SIGIO
handler
initiate
specify
signal/handler
blocked
blocked
blocked
ready
notification
wait for data
initiate
initiate
blocked
blocked
blocked
copy data
complete
complete
complete
complete
notification
synchronous I/O
asynchronous I/O

I/O模型的具体实现

- 主要实现方式有以下几种：
  - Select：Linux实现对应，I/O复用模型，BSD4.2最早实现
  - Poll：Linux实现，对应I/O复用模型，System V unix最早实现
  - Epoll：Linux实现，对应I/O复用模型，具有信号驱动I/O模型的某些特性
  - Kqueue：FreeBSD实现，对应I/O复用模型，具有信号驱动I/O模型的某些特性
  - /dev/poll：SUN的Solaris实现，对应I/O复用模型，具有信号驱动I/O模型的某些特性
  - Iocp　Windows实现，对应第5种（异步I/O）模型

select/poll/epoll

\	select	poll	epoll
操作方式	遍历	遍历	回调
底层实现	数组	链表	哈希表
IO效率	每次调用都进行线性遍历，时间复杂度为O(n)	每次调用都进行线性遍历，时间复杂度为O(n)	事件通知方式，每当fd就绪，系统注册的回调函数就会被调用，将就绪fd放到rdllist里面。时间复杂度O(1)
最大连接数	1024（x86）或 2048（x64）	无上限	无上限
fd拷贝	每次调用select，都需要把fd集合从用户态拷贝到内核态	每次调用poll，都需要把fd集合从用户态拷贝到内核态	调用epoll_ctl时拷贝进内核并保存，之后每次epoll_wait不拷贝

select/poll/epoll

- Select:POSIX所规定，目前几乎在所有的平台上支持，其良好跨平台支持也是它的一个优点，本质上是通过设置或者检查存放fd标志位的数据结构来进行下一步处理
- 缺点
  - 单个进程可监视的fd数量被限制，即能监听端口的数量有限

    cat /proc/sys/fs/file-max
  - 对socket是线性扫描，即采用轮询的方法，效率较低
  - select 采取了内存拷贝方法来实现内核将 FD 消息通知给用户空间，这样一个用来存放大量fd的数据结构，这样会使得用户空间和内核空间在传递该结构时复制开销大

select/poll/epoll

- poll
  - 本质上和select没有区别，它将用户传入的数组拷贝到内核空间，然后查询每个fd对应的设备状态
  - 其没有最大连接数的限制，原因是它是基于链表来存储的
  - 大量的fd的数组被整体复制于用户态和内核地址空间之间，而不管这样的复制是不是有意义
  - poll特点是“水平触发”，如果报告了fd后，没有被处理，那么下次poll时会再次报告该fd
  - 边缘触发：只通知一次

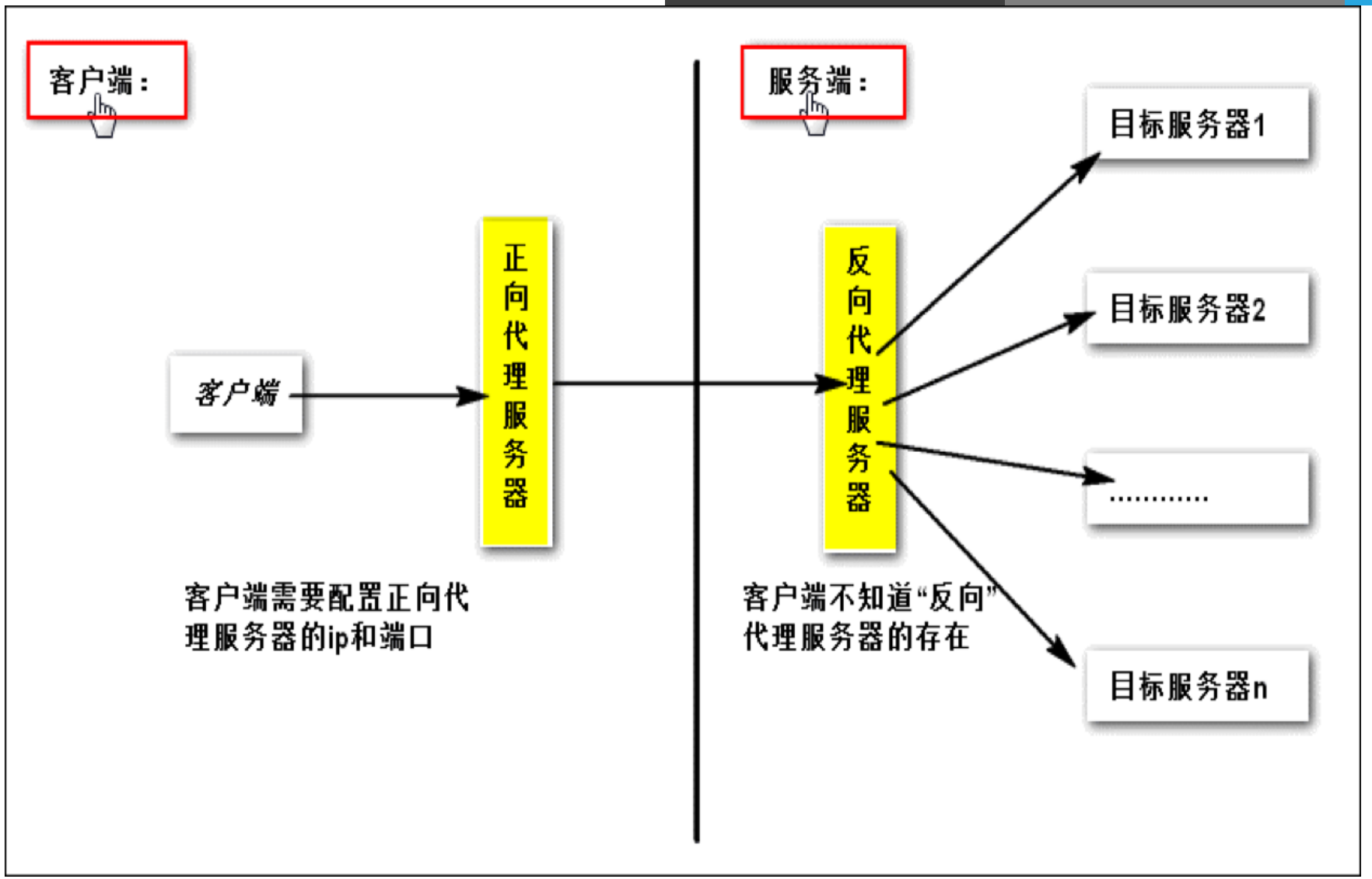
select/poll/epoll

- epoll：在Linux 2.6内核中提出的select和poll的增强版本
  - 支持水平触发LT和边缘触发ET，最大的特点在于边缘触发，它只告诉进程哪些fd刚刚变为就需态，并且只会通知一次
  - 使用“事件”的就绪通知方式，通过epoll_ctl注册fd，一旦该fd就绪，内核就会采用类似callback的回调机制来激活该fd，epoll_wait便可以收到通知
- 优点:
  - 没有最大并发连接的限制：能打开的FD的上限远大于1024(1G的内存能监听约10万个端口)
  - 效率提升：非轮询的方式，不会随着FD数目的增加而效率下降；只有活跃可用的FD才会调用callback函数，即epoll最大的优点就在于它只管理“活跃”的连接，而跟连接总数无关
  - 内存拷贝，利用mmap()文件映射内存加速与内核空间的消息传递；即epoll使用mmap减少复制开销

Nginx介绍

- Nginx：engine X ，2002年，开源，商业版
- http协议：web服务器（类似于httpd）、http reverse proxy（类似于httpd）、imap/pop3 reverse proxy,tcp
- NGINX is a free, open-source, high-performance an HTTP and reverse proxy server, a mail proxy server, and a generic TCP/UDP proxy server
- C10K（10K Connections）
- 二次开发版：Tengine, OpenResty
- 官网：http://nginx.org

正向代理和反向代理


客户端：
服务端：
目标服务器1
目标服务器2
.....
目标服务器n
客户端
正向代理服务器
反向代理服务器
客户端需要配置正向代理服务器的ip和端口
客户端不知道“反向”代理服务器的存在

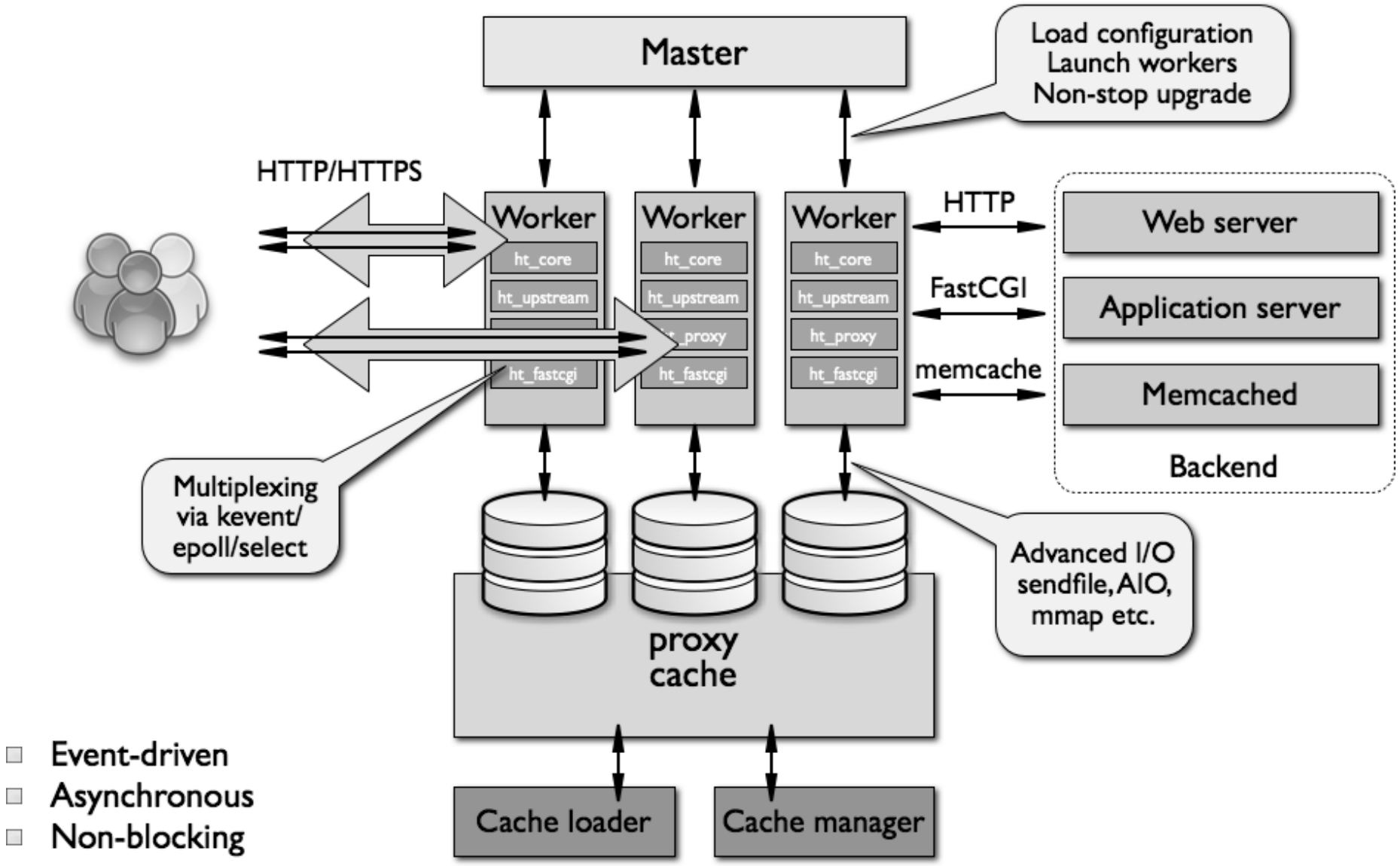
Nginx介绍

- 特性：
  - 模块化设计，较好的扩展性
  - 高可靠性
  - 支持热部署：不停机更新配置文件，升级版本，更换日志文件
  - 低内存消耗：10000个keep-alive连接模式下的非活动连接，仅需要2.5M内存
  - event-driven,aio,mmap，sendfile
- 基本功能：
  - 静态资源的web服务器
  - http协议反向代理服务器
  - pop3/imap4协议反向代理服务器
  - FastCGI(lnmp),uWSGI(python)等协议
  - 模块化（非DSO），如zip，SSL模块

nginx的程序架构

- web服务相关的功能：
  - 虚拟主机（server）
  - 支持 keep-alive 和管道连接
  - 访问日志（支持基于日志缓冲提高其性能）
  - url rewirte
  - 路径别名
  - 基于IP及用户的访问控制
  - 支持速率限制及并发数限制
  - 重新配置和在线升级而无须中断客户的工作进程
  - Memcached 的 GET 接口

nginx架构



- Event-driven
- Asynchronous
- Non-blocking

nginx的程序架构

- nginx的程序架构：

  master/worker结构

  - 一个master进程：

    负载加载和分析配置文件、管理worker进程、平滑升级

  - 一个或多个worker进程

    处理并响应用户请求

  - 缓存相关的进程：

    cache loader：载入缓存对象

    cache manager：管理缓存对象

nginx模块

- nginx高度模块化，但其模块早期不支持DSO机制；1.9.11版本支持动态装载和卸载
- 模块分类：
  - 核心模块：core module
  - 标准模块：
    - HTTP 模块： ngx_http_*

      HTTP Core modules 默认功能

      HTTP Optional modules 需编译时指定
    - Mail 模块 ngx_mail_*
    - Stream 模块 ngx_stream_*
  - 第三方模块

nginx的功用

- 静态的web资源服务器

  html，图片，js，css，txt等静态资源
- 结合FastCGI/uWSGI/SCGI等协议反向代理动态资源请求
- http/https协议的反向代理
- imap4/pop3协议的反向代理
- tcp/udp协议的请求转发（反向代理）

nginx的安装

- 官方：
  http://nginx.org/packages/centos/7/x86_64/RPMS
- Fedora-EPEL:
  https://mirrors.aliyun.com/epel/7/x86_64/
- 编译安装：

- yum install pcre-devel openssl-devel zlib-devel
- useradd -r nginx
- ./configure --prefix=/usr/local/nginx --conf-path=/etc/nginx/nginx.conf --error-log-path=/var/log/nginx/error.log --http-log-path=/var/log/nginx/access.log --pid-path=/var/run/nginx.pid --lock-path=/var/run/nginx.lock --user=nginx --group=nginx --with-http_ssl_module --with-http_v2_module --with-http_dav_module --with-http_stub_status_module --with-threads --with-file-aio
- make && make install

编译安装nginx选项

- 编译安装nginx选项：
  - --prefix=/etc/nginx 安装路径
  - --sbin-path=/usr/sbin/nginx 指明nginx程序文件安装路径
  - --conf-path=/etc/nginx/nginx.conf 主配置文件安装位置
  - --error-log-path=/var/log/nginx/error.log 错误日志文件安装位置
  - --http-log-path=/var/log/nginx/access.log 访问日志文件安装位置
  - --pid-path=/var/run/nginx.pid 指明pid文件安装位置
  - --lock-path=/var/run/nginx.lock 锁文件安装位置
  - --http-client-body-temp-path=/var/cache/nginx/client_temp 客户端body部分的临时文件存放路径，如果服务器允许客户端使用put方法提交大数据时，临时存放的磁盘路径

编译安装nginx选项

- --http-proxy-temp-path=/var/cache/nginx/proxy_temp 作为代理服务器，服务器响应报文的临时文件存放路径
- --http-fastcgi-temp-path=/var/cache/nginx/fastcgi_temp 作为fastcgi代理服务器，服务器响应报文的临时文件存放路径
- --http-uwsgi-temp-path=/var/cache/nginx/uwsgi_temp 作为uwsgi代理服务器，服务器响应报文的临时文件存放路径
- --http-scgi-temp-path=/var/cache/nginx/scgi_temp 作为scgi反代服务器，服务器响应报文的临时文件存放路径
- --user=nginx 指明以那个身份运行worker进程，主控master进程一般由root运行
- --group=nginx
- --with-http_ssl_module 表示把指定模块编译进来

nginx目录结构和命令

- ls /usr/local/nginx/
  html是测试页，sbin是主程序
- ls /usr/local/nginx/sbin/
  nginx 只有一个程序文件
- ls /usr/local/nginx/html/
  50x.html index.html 测试网页
- Nginx：默认为启动nginx
  -h 查看帮助选项
  -V 查看版本和配置选项
  -t 测试nginx语法错误
  -c filename 指定配置文件(default: /etc/nginx/nginx.conf)
  -s signal 发送信号给master进程，signal可为：stop, quit, reopen, reload 示例：-s stop 停止nginx -s reload 加载配置文件
  -g directives 在命令行中指明全局指令

nginx配置

- 配置文件的组成部分：
  - 主配置文件：nginx.conf

    子配置文件 include conf.d/*.conf
  - fastcgi， uwsgi，scgi等协议相关的配置文件
  - mime.types：支持的mime类型
- 主配置文件的配置指令：
  - directive value [value2 ...];
- 注意：

  (1) 指令必须以分号结尾

  (2) 支持使用配置变量

  内建变量：由Nginx模块引入，可直接引用

  自定义变量：由用户使用set命令定义

  set variable_name value;

  引用变量：$variable_name

nginx配置文件

- 主配置文件结构：四部
  - main block：主配置段，即全局配置段，对http,mail都有效

```
event {
        ...
    }  事件驱动相关的配置
```

  - http {

```
        ...
    }       http/https 协议相关配置段
```

  - mail {

```
        ...
    }       mail 协议相关配置段
```

  - stream {

```
        ...
    }          stream 服务器相关配置段
```

http协议相关的配置结构


```
http {
        ...
        ... 各server的公共配置
        server {    每个server用于定义一个虚拟主机
                ...
                }
        server {
                ...
                server_name   虚拟主机名
                root     主目录
                alias     路径别名
                location [OPERATOR] URL {     指定URL的特性
                        ...
                        if CONDITION {
...
                        }
                }
        }
}
```

nginx配置

- Main 全局配置段常见的配置指令分类
  - 正常运行必备的配置
  - 优化性能相关的配置
  - 用于调试及定位问题相关的配置
  - 事件驱动相关的配置
- 帮助文档

  http://nginx.org/en/docs/

nginx配置

- 正常运行必备的配置：
- 帮助文档：http://nginx.org/en/docs/ngx_core_module.html
  - 1、user

    Syntax: user user [group];

    Default:user nobody nobody;

    Context: main

    指定worker进程的运行身份，如组不指定，默认和用户名同名
  - 2、pid /PATH/TO/PID_FILE

    指定存储nginx主进程PID的文件路径
  - 3、include file | mask

    指明包含进来的其它配置文件片断
  - 4、load_module file

    模块加载配置文件：/usr/share/nginx/modules/*.conf

    指明要装载的动态模块路径: /usr/lib64/nginx/modules

nginx配置

- 性能优化相关的配置：
  - 1、worker_processes number | auto
    worker进程的数量；通常应该为当前主机的cpu的物理核心数
  - 2、worker_cpu_affinity cpumask ...
    worker_cpu_affinity auto [cpumask] 提高缓存命中率
    CPU MASK：　00000001：0号CPU
    00000010：1号CPU
    10000000：8号CPU
    worker_cpu_affinity 0001 0010 0100 1000;
    worker_cpu_affinity 0101 1010;
  - 3、worker_priority number
    指定worker进程的nice值，设定worker进程优先级：[-20,20]
  - 4、worker_rlimit_nofile number
    worker进程所能够打开的文件数量上限,如65535

nginx配置

- 事件驱动相关的配置:
  - events {
    ...
    }
  - 1、worker_connections number
    每个worker进程所能够打开的最大并发连接数数量，如10240
    总最大并发数：worker_processes * worker_connections
  - 2、use method
    指明并发连接请求的处理方法,默认自动选择最优方法
    use epoll;
  - 3、accept_mutex on | off 互斥
    处理新的连接请求的方法；on指由各个worker轮流处理新请求，Off指每个新请求的到达都会通知(唤醒)所有的worker进程，但只有一个进程可获得连接，造成“惊群”，影响性能，默认on

nginx配置

- 调试和定位问题：
  - 1、daemon on|off

    是否以守护进程方式运行nignx，默认是守护进程方式
  - 2、master_process on|off

    是否以master/worker模型运行nginx；默认为on

    off 将不启动worker
  - 3、error_log file [level]

    错误日志文件及其级别；出于调试需要，可设定为debug；但debug仅在编译时使用了 “--with-debug”选项时才有效

    方式：file /path/logfile;

    stderr:发送到标准错误

    syslog:server-address[,parameter=values]:发送到syslog memory:size 内存

    level:debug|info|notice|warn|error|crit|alter|emerg

nginx配置

- http协议的相关配置：

```
http {
        ... ...
        server {
                ...
                server_name
                root
                location [OPERATOR] /uri/ {
                        ...
                }
        }
        server {
                ...
        }
}
```

ngx_http_core_module

- ngx_http_core_module
- 与套接字相关的配置：
  - 1、server { ... }

    配置一个虚拟主机

    server {

        listen address[:PORT]|PORT;

        server_name SERVER_NAME;

        root /PATH/TO/DOCUMENT_ROOT;

    }

ngx_http_core_module

- 2、listen PORT|address[:port]|unix:/PATH/TO/SOCKET_FILE

  listen address[:port] [default_server] [ssl] [http2 | spdy] [backlog=number] [rcvbuf=size] [sndbuf=size]

  default_server 设定为默认虚拟主机

  ssl 限制仅能够通过ssl连接提供服务

  backlog=number 超过并发连接数后，新请求进入后援队列的长度

  rcvbuf=size 接收缓冲区大小

  sndbuf=size 发送缓冲区大小

  - 注意：

(1) 基于port；

listen PORT; 指令监听在不同的端口

(2) 基于ip的虚拟主机

listen IP:PORT; IP 地址不同

(3) 基于hostname

server_name fqdn; 指令指向不同的主机名

ngx_http_core_module

- 3、server_name name ...;
  - 虚拟主机的主机名称后可跟多个由空白字符分隔的字符串
  - 支持*通配任意长度的任意字符

    server_name *.magedu.com www.magedu.*
  - 支持~起始的字符做正则表达式模式匹配，性能原因慎用

    server_name ~^www\d+\.magedu\.com$

    \d 表示 [0-9]
  - 匹配优先级机制从高到低：

(1) 首先是字符串精确匹配 如：www.magedu.com
(2) 左侧*通配符 如：*.magedu.com
(3) 右侧*通配符 如：www.magedu.*
(4) 正则表达式 如： ~^.*\.magedu\.com$
(5) default_server

ngx_http_core_module

- 4、tcp_nodelay on | off;
  在keepalived模式下的连接是否启用TCP_NODELAY选项
  当为off时，延迟发送，合并多个请求后再发送
  默认On时，不延迟发送
  可用于：http, server, location
- 5、sendfile on | off;
  是否启用sendfile功能，在内核中封装报文直接发送
  默认Off
- 6、server_tokens on | off | build | string
  是否在响应报文的Server首部显示nginx版本

ngx_http_core_module

- 定义路径相关的配置
- 7、root
- 设置web资源的路径映射；用于指明请求的URL所对应的文档的目录路径，可用于http, server, location, if in location

```
server {
        ...
        root  /data/www/vhost1;
}
```

  - 示例

    http://www.magedu.com/images/logo.jpg
    --> /data/www/vhosts/images/logo.jpg

ngx_http_core_module

- 8、location [= | ~ | ~* | ^~] uri { ... }
  location @name { ... }
  在一个server中location配置段可存在多个，用于实现从uri到文件系统的路径映射；ngnix会根据用户请求的URI来检查定义的所有location，并找出一个最佳匹配，而后应用其配置
- 示例：

```
server {...
        server_name  www.magedu.com;
        location /images/ {
                root  /data/imgs/;
                }
}
```

http://www.magedu.com/images/logo.jpg
--> /data/imgs/images/logo.jpg

ngx_http_core_module

- =：对URI做精确匹配；

```
location = / {
...
}
```

  http://www.magedu.com/ 匹配
  http://www.magedu.com/index.html 不匹配

- ^~：对URI的最左边部分做匹配检查，不区分字符大小写
- ~：对URI做正则表达式模式匹配，区分字符大小写
- ~*：对URI做正则表达式模式匹配，不区分字符大小写
- 不带符号：匹配起始于此uri的所有的uri
- 匹配优先级从高到低：

  =, ^~, ~/~*, 不带符号

ngx_http_core_module

- 示例：
  - root /vhosts/www/htdocs/

    http://www.magedu.com/index.html

    --> /vhosts/www/htdocs/index.html
  - server {

    root /vhosts/www/htdocs/

    location /admin/ {

    root /webapps/app1/data/

    }

    }

    http://www.magedu.com/admin/index.html

    --> /webapps/app1/data/admin/index.html

location示例

```
location = / {
        [ configuration A ]
}
location / {
        [ configuration B ]
}
location /documents/ {
        [ configuration C ]
}
location ^~ /images/ {
        [ configuration D ]
}
location ~* \.(gif|jpg|jpeg)$ {
        [ configuration E ]
}
```

http://www.mgadu.com

http://www.magedu.com/index.html

http://www.magedu.com/documents/log.jpg

http://www.magedu.com/documents/linux.txt

lhttp://www.magedu.com/images/log.jpeg

ngx_http_core_module

- 9、alias path;
  路径别名，文档映射的另一种机制；仅能用于location上下文
- 示例：

http://www.magedu.com/bbs/index.php

location /bbs/ {
  alias /web/forum/;
} --> /web/forum/index.html

location /bbs/ {
  root /web/forum/;
} --> /web/forum/bbs/index.html

  - 注意：location中使用root指令和alias指令的意义不同
    (a) root，给定的路径对应于location中的/uri/左侧的/
    (b) alias，给定的路径对应于location中的/uri/右侧的/

- 10、index file ...;
  指定默认网页资源，注意：ngx_http_index_module模块

ngx_http_core_module

- 11、error_page code ... [=[response]] uri;
  模块：ngx_http_core_module
  定义错误页，以指定的响应状态码进行响应
  可用位置：http, server, location, if in location
  error_page 404 /404.html
  error_page 404 =200 /404.html

ngx_http_core_module

- 12、try_files file ... uri;
  try_files file ... =code;

按顺序检查文件是否存在，返回第一个找到的文件或文件夹（结尾加斜线表示为文件夹），如果所有的文件或文件夹都找不到，会进行一个内部重定向到最后一个参数。只有最后一个参数可以引起一个内部重定向，之前的参数只设置内部URI的指向。最后一个参数是回退URI且必须存在，否则会出现内部500错误

```
location /images/ {
        try_files $uri /images/default.gif; 、
}
location / {
        try_files $uri $uri/index.html $uri.html =404;
}
```

ngx_http_core_module

- 定义客户端请求的相关配置
- 13、keepalive_timeout timeout [header_timeout];
  设定保持连接超时时长，0表示禁止长连接，默认为75s
- 14、keepalive_requests number;
  在一次长连接上所允许请求的资源的最大数量
  默认为100
- 15、keepalive_disable none | browser ...
  对哪种浏览器禁用长连接
- 16、send_timeout time;
  向客户端发送响应报文的超时时长，此处是指两次写操作之间的间隔时长，而非整个响应过程的传输时长

ngx_http_core_module

- 17、client_body_buffer_size size;

用于接收每个客户端请求报文的body部分的缓冲区大小；默认为16k；超出此大小时，其将被暂存到磁盘上的由client_body_temp_path指令所定义的位置

- 18、client_body_temp_path path [level1 [level2 [level3]]];

设定用于存储客户端请求报文的body部分的临时存储路径及子目录结构和数量

目录名为16进制的数字；

client_body_temp_path /var/tmp/client_body 1 2 2

1 1级目录占1位16进制，即2^4=16个目录 0-f

2 2级目录占2位16进制，即2^8=256个目录 00-ff

2 3级目录占2位16进制，即2^8=256个目录 00-ff

ngx_http_core_module

- 对客户端进行限制的相关配置
- 19、limit_rate rate;
  限制响应给客户端的传输速率，单位是bytes/second
  默认值0表示无限制
- 20、limit_except method ... { ... }，仅用于location
  限制客户端使用除了指定的请求方法之外的其它方法
  method:GET, HEAD, POST, PUT, DELETE
  MKCOL, COPY, MOVE, OPTIONS, PROPFIND,
  PROPPATCH, LOCK, UNLOCK, PATCH

```
limit_except GET {
        allow 192.168.1.0/24;
        deny  all;
} 除了GET和HEAD 之外其它方法仅允许192.168.1.0/24网段主机使用
```

ngx_http_core_module

- 文件操作优化的配置
- 21、aio on | off | threads[=pool];
  是否启用aio功能
- 22、directio size | off;
  是否同步（直接）写磁盘，而非写缓存，在Linux主机启用O_DIRECT标记，则文件大于等于给定大小时使用，例如directio 4m
- 23、open_file_cache off;
  open_file_cache max=N [inactive=time];
  nginx可以缓存以下三种信息：
  (1) 文件元数据：文件的描述符、文件大小和最近一次的修改时间
  (2) 打开的目录结构
  (3) 没有找到的或者没有权限访问的文件的相关信息
  max=N：可缓存的缓存项上限；达到上限后会使用LRU算法实现管理
  inactive=time：缓存项的非活动时长，在此处指定的时长内未被命中的或命中的次数少于open_file_cache_min_uses指令所指定的次数的缓存项即为非活动项，将被删除

ngx_http_core_module

- 24、open_file_cache_errors on | off;

  是否缓存查找时发生错误的文件一类的信息

  默认值为off

- 25、open_file_cache_min_uses number;

  open_file_cache指令的inactive参数指定的时长内，至少被命中此处指定的次数方可被归类为活动项

  默认值为1

- 26、open_file_cache_valid time;

  缓存项有效性的检查频率

  默认值为60s

ngx_http_access_module

- ngx_http_access_module模块
  实现基于ip的访问控制功能
- 1、allow address | CIDR | unix: | all;
- 2、deny address | CIDR | unix: | all;
  http, server, location, limit_except
  自上而下检查，一旦匹配，将生效，条件严格的置前
- 示例：

```
location / {
    deny  192.168.1.1;
    allow 192.168.1.0/24;
    allow 10.1.1.0/16;
    allow 2001:0db8::/32;
    deny  all;
}
```

ngx_http_auth_basic_module

- ngx_http_auth_basic_module模块
  实现基于用户的访问控制，使用basic机制进行用户认证
- 1、auth_basic string | off;
- 2、auth_basic_user_file file;

```
location /admin/ {
        auth_basic "Admin Area";
        auth_basic_user_file /etc/nginx/.ngxpasswd;
}
```

  - 用户口令文件：
    1、明文文本：格式name:password:comment
    2、加密文本：由htpasswd命令实现
    httpd-tools所提供

ngx_http_stub_status_module

- ngx_http_stub_status_module模块
  用于输出nginx的基本状态信息
  输出信息示例：

  Active connections: 291

  server accepts handled requests

  16630948 16630948 31070465

  上面三个数字分别对应accepts,handled,requests三个值

  Reading: 6 Writing: 179 Waiting: 106

ngx_http_stub_status_module

Active connections:当前状态，活动状态的连接数
accepts：统计总值，已经接受的客户端请求的总数
handled：统计总值，已经处理完成的客户端请求的总数
requests：统计总值，客户端发来的总的请求数
Reading：当前状态，正在读取客户端请求报文首部的连接的连接数
Writing：当前状态，正在向客户端发送响应报文过程中的连接数
Waiting：当前状态，正在等待客户端发出请求的空闲连接数

- 1、stub_status;

示例：

```
location  /status {
        stub_status;
        allow 172.16.0.0/16;
        deny all;
}
```

ngx_http_log_module

- ngx_http_log_module模块
  指定日志格式记录请求
- 1、log_format name string ...;
  string可以使用nginx核心模块及其它模块内嵌的变量
- 2、access_log path [format [buffer=size] [gzip[=level]] [flush=time] [if=condition]];
  access_log off;
  访问日志文件路径，格式及相关的缓冲的配置
  buffer=size
  flush=time

ngx_http_log_module

◆示例

```
log_format compression '$remote_addr-$remote_user [$time_local] '
        '"$request" $status $bytes_sent '
        '"$http_referer" "$http_user_agent" "$gzip_ratio"';

access_log /spool/logs/nginx-access.log compression buffer=32k;
```

ngx_http_log_module

- 3、open_log_file_cache max=N [inactive=time] [min_uses=N] [valid=time];

  open_log_file_cache off;

  缓存各日志文件相关的元数据信息

  max：缓存的最大文件描述符数量

  min_uses：在inactive指定的时长内访问大于等于此值方可被当作活动项

  inactive：非活动时长

  valid：验证缓存中各缓存项是否为活动项的时间间隔

ngx_http_gzip_module

- ngx_http_gzip_module
  用gzip方法压缩响应数据，节约带宽
- 1、gzip on | off;
  启用或禁用gzip压缩
- 2、gzip_comp_level level;
  压缩比由低到高：1 到 9
  默认：1
- 3、gzip_disable regex ...;
  匹配到客户端浏览器不执行压缩
- 4、gzip_min_length length;
  启用压缩功能的响应报文大小阈值

ngx_http_gzip_module

- 5、gzip_http_version 1.0 | 1.1;
  设定启用压缩功能时，协议的最小版本
  默认：1.1
- 6、gzip_buffers number size;
  支持实现压缩功能时缓冲区数量及每个缓存区的大小
  默认：32 4k 或 16 8k
- 7、gzip_types mime-type ...;
  指明仅对哪些类型的资源执行压缩操作；即压缩过滤器
  默认包含有text/html，不用显示指定，否则出错
- 8、gzip_vary on | off;
  如果启用压缩，是否在响应报文首部插入“Vary: Accept-Encoding”

ngx_http_gzip_module

- 9、gzip_proxied off | expired | no-cache | no-store | private | no_last_modified | no_etag | auth | any ...;

  nginx对于代理服务器请求的响应报文，在何种条件下启用压缩功能

  off：对被代理的请求不启用压缩

  expired,no-cache, no-store，private：对代理服务器请求的响应报文首部Cache-Control值任何一个，启用压缩功能

- 示例：

```
gzip  on;
gzip_comp_level 6;
gzip_min_length 64;
gzip_proxied any;
gzip_types text/xml text/css  application/javascript;
```

ngx_http_ssl_module

- ngx_http_ssl_module模块：
- 1、ssl on | off;
  为指定虚拟机启用HTTPS protocol， 建议用listen指令代替
- 2、ssl_certificate file;
  当前虚拟主机使用PEM格式的证书文件
- 3、ssl_certificate_key file;
  当前虚拟主机上与其证书匹配的私钥文件
- 4、ssl_protocols [SSLv2] [SSLv3] [TLSv1] [TLSv1.1] [TLSv1.2];
  支持ssl协议版本，默认为后三个
- 5、ssl_session_cache off | none | [builtin[:size]] [shared:name:size];
  builtin[:size]：使用OpenSSL内建缓存，为每worker进程私有
  [shared:name:size]：在各worker之间使用一个共享的缓存

ngx_http_ssl_module

- 6、ssl_session_timeout time;

  客户端连接可以复用ssl session cache中缓存的ssl参数的有效时长，默认5m

- 示例：

```
server {
        listen 443 ssl;
        server_name www.magedu.com;
        root /vhosts/ssl/htdocs;
        ssl on;
        ssl_certificate /etc/nginx/ssl/nginx.crt;
        ssl_certificate_key /etc/nginx/ssl/nginx.key;
        ssl_session_cache shared:sslcache:20m;
        ssl_session_timeout 10m;
}
```

ngx_http_rewrite_module

- ngx_http_rewrite_module模块：

The ngx_http_rewrite_module module is used to change request URI using PCRE regular expressions, return redirects, and conditionally select configurations.

将用户请求的URI基于PCRE regex所描述的模式进行检查，而后完成重定向替换

- 示例：

http://www.magedu.com/hn

--> http://www.magedu.com/henan

http://www.magedu.com

--> https://www.magedu.com/

ngx_http_rewrite_module

- 1、rewrite regex replacement [flag]

将用户请求的URI基于regex所描述的模式进行检查，匹配到时将其替换为replacement指定的新的URI

注意：如果在同一级配置块中存在多个rewrite规则，那么会自下而下逐个检查；被某条件规则替换完成后，会重新一轮的替换检查

隐含有循环机制,但不超过10次；如果超过，提示500响应码，[flag]所表示的标志位用于控制此循环机制

如果replacement是以http://或https://开头，则替换结果会直接以重向返回给客户端

301：永久重定向

ngx_http_rewrite_module

- [flag]：

last：重写完成后停止对当前URI在当前location中后续的其它重写操作，而后对新的URI启动新一轮重写检查；提前重启新一轮循环，不建议在lation中使用

break：重写完成后停止对当前URI在当前location中后续的其它重写操作，而后直接跳转至重写规则配置块之后的其它配置；结束循环，建议在location中使用

redirect：临时重定向，重写完成后以临时重定向方式直接返回重写后生成的新URI给客户端，由客户端重新发起请求；不能以http://或https://开头，使用相对路径，状态码：302

permanent:重写完成后以永久重定向方式直接返回重写后生成的新URI给客户端，由客户端重新发起请求，状态码：301

ngx_http_rewrite_module

- 2、return

  return code [text];

  return code URL;

  return URL;

  停止处理，并返回给客户端指定的响应码

- 3、rewrite_log on | off;

  是否开启重写日志, 发送至error_log（notice level）

- 4、set $variable value;

  用户自定义变量

  注意：变量定义和调用都要以$开头

ngx_http_rewrite_module

- 5、if (condition) { ... }

引入新的上下文,条件满足时，执行配置块中的配置指令；server, location

condition：

比较操作符：

== 相同

!= 不同

~：模式匹配，区分字符大小写

~*：模式匹配，不区分字符大小写

!~：模式不匹配，区分字符大小写

!~*：模式不匹配，不区分字符大小写

文件及目录存在性判断：

-e, !-e 存在（包括文件，目录，软链接）

-f, !-f 文件

-d, !-d 目录

-x, !-x 执行

ngx_http_referer_module

- ngx_http_referer_module模块：

  用来阻止Referer首部无有效值的请求访问，可防止盗链

- 1、valid_referers none|blocked|server_names|string ...;

  定义referer首部的合法可用值，不能匹配的将是非法值

  none：请求报文首部没有referer首部

  blocked：请求报文有referer首部，但无有效值

  server_names：参数，其可以有值作为主机名或主机名模式

  arbitrary_string：任意字符串，但可使用*作通配符

  regular expression：被指定的正则表达式模式匹配到的字符串,要使用~开头，例如： ~.*\.magedu\.com

ngx_http_referer_module

- 示例：

```
valid_referers none block server_names *.magedu.com *.mageedu.com magedu.*
mageedu.* ~\.magedu\.;
if  ($invalid_referer)  {
                return 403  http://www.magedu.com;
}
```

ngx_http_proxy_module

- ngx_http_proxy_module模块：
  转发请求至另一台主机

  1、 proxy_pass URL;

  Context:location, if in location, limit_except

  注意：proxy_pass后面路径不带uri时，会将location的uri传递（附加）给后端主机

```
server {
        ...
        server_name HOSTNAME;
        location /uri/ {
                proxy_pass http://host[:port]; 最后没有/
        }
        ...
}
```

  上面示例：http://HOSTNAME/uri --> http://host/uri

  如果上面示例中有 / ，即：http://host[:port]/

  意味着：http://HOSTNAME/uri --> http://host/ 即置换

ngx_http_proxy_module

- proxy_pass后面的路径是一个uri时，其会将location的uri替换为proxy_pass的uri

```
server {
        ...
        server_name HOSTNAME;
        location /uri/ {
                proxy_pass http://host/new_uri/;
        }
        ...
}
```

http://HOSTNAME/uri/ --> http://host/new_uri/

ngx_http_proxy_module

- 如果location定义其uri时使用了正则表达式的模式，则proxy_pass之后必须不能使用uri; 用户请求时传递的uri将直接附加至后端服务器之后

```
server {
        ...
        server_name HOSTNAME;
        location ~|~* /uri/ {
                proxy_pass http://host; 不能加/
        }
        ...
}
```

http://HOSTNAME/uri/ --> http://host/uri/

ngx_http_proxy_module

- 2、proxy_set_header field value;

  设定发往后端主机的请求报文的请求首部的值

  Context: http, server, location

  proxy_set_header X-Real-IP $remote_addr;

  proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

  请求报文的标准格式如下：

  X-Forwarded-For: client1, proxy1, proxy2

- 3、proxy_cache_path;

  定义可用于proxy功能的缓存；Context:http

  proxy_cache_path path [levels=levels] [use_temp_path=on|off] keys_zone=name:size [inactive=time] [max_size=size] [manager_files=number] [manager_sleep=time] [manager_threshold=time] [loader_files=number] [loader_sleep=time] [loader_threshold=time] [purger=on|off] [purger_files=number] [purger_sleep=time] [purger_threshold=time];

ngx_http_proxy_module

- 4、proxy_cache zone | off; 默认off
  指明调用的缓存，或关闭缓存机制；Context:http, server, location
- 5、proxy_cache_key string;
  缓存中用于“键”的内容
  默认值：proxy_cache_key $scheme$proxy_host$request_uri;
- 6、proxy_cache_valid [code ...] time;
  定义对特定响应码的响应内容的缓存时长
  定义在http{...}中
  示例:
  proxy_cache_valid 200 302 10m;
  proxy_cache_valid 404 1m;

ngx_http_proxy_module

- 示例：

在http配置定义缓存信

```
proxy_cache_path /var/cache/nginx/proxy_cache
    levels=1:1:1 keys_zone=proxycache:20m
    inactive=120s max_size=1g;
```

调用缓存功能，需要定义在相应的配置段，如server{...}；

```
proxy_cache proxycache;
proxy_cache_key $request_uri;
proxy_cache_valid 200 302 301 1h;
proxy_cache_valid any 1m;
```

ngx_http_proxy_module

- 7、proxy_cache_use_stale;

  proxy_cache_use_stale error | timeout | invalid_header | updating | http_500 | http_502 | http_503 | http_504 | http_403 | http_404 | off ...

  在被代理的后端服务器出现哪种情况下，可以真接使用过期的缓存响应客户端

- 8、proxy_cache_methods GET | HEAD | POST ...;

  对哪些客户端请求方法对应的响应进行缓存，GET和HEAD方法总是被缓存

ngx_http_proxy_module

- 9、proxy_hide_header field;

  默认nginx在响应报文不传递后端服务器的首部字段Date, Server, X-Pad, X-Accel-等，用于隐藏后端服务器特定的响应首部

- 10、proxy_connect_timeout time;

  定义与后端服务器建立连接的超时时长，如超时会出现502错误，默认为60s，一般不建议超出75s，

- 11、proxy_send_timeout time;

  将请求发送给后端服务器的超时时长；默认为60s

- 12、proxy_read_timeout time;

  等待后端服务器发送响应报文的超时时长，默认为60s

ngx_http_headers_module

- ngx_http_headers_module模块

  向由代理服务器响应给客户端的响应报文添加自定义首部，或修改指定首部的值

- 1、add_header name value [always];

  添加自定义首部

  add_header X-Via $server_addr;

  add_header X-Cache $upstream_cache_status;

  add_header X-Accel $server_name;

- 2、add_trailer name value [always];

  添加自定义响应信息的尾部

ngx_http_fastcgi_module

- ngx_http_fastcgi_module模块

  转发请求到FastCGI服务器，不支持php模块方式

- 1、fastcgi_pass address;

  address为后端的fastcgi server的地址

  可用位置：location, if in location

- 2、fastcgi_index name;

  fastcgi默认的主页资源

  示例：fastcgi_index index.php;

- 3、fastcgi_param parameter value [if_not_empty];

  设置传递给 FastCGI服务器的参数值，可以是文本，变量或组合

ngx_http_fastcgi_module

- 示例1：
- 1）在后端服务器先配置fpm server和mariadb-server
- 2）在前端nginx服务上做以下配置：

```
location ~* \.php$ {
        fastcgi_pass   后端fpm服务器IP:9000;
        fastcgi_index  index.php;
        fastcgi_param  SCRIPT_FILENAME
/usr/share/nginx/html$fastcgi_script_name;
        include        fastcgi_params;
        ...
}
```

ngx_http_fastcgi_module

- 示例2：通过/pm_status和/ping来获取fpm server状态信息

```
location ~* ^/(pm_status|ping)$ {
        include        fastcgi_params;
        fastcgi_pass  后端fpm服务器IP:9000;
        fastcgi_param  SCRIPT_FILENAME  $fastcgi_script_name;
}
```

ngx_http_fastcgi_module

- 4、fastcgi_cache_path path [levels=levels] [use_temp_path=on|off] keys_zone=name:size [inactive=time] [max_size=size] [manager_files=number] [manager_sleep=time] [manager_threshold=time] [loader_files=number] [loader_sleep=time] [loader_threshold=time] [purger=on|off] [purger_files=number] [purger_sleep=time] [purger_threshold=time];
- 定义fastcgi的缓存；

  path　缓存位置为磁盘上的文件系统

  max_size=size

  　　磁盘path路径中用于缓存数据的缓存空间上限

  levels=levels：缓存目录的层级数量，以及每一级的目录数量

  　levels=ONE:TWO:THREE

  　示例：leves=1:2:2

  keys_zone=name:size

  　　k/v映射的内存空间的名称及大小

  inactive=time

  　　非活动时长

ngx_http_fastcgi_module

- 5、fastcgi_cache zone | off;
  调用指定的缓存空间来缓存数据
  可用位置：http, server, location
- 6、fastcgi_cache_key string;
  定义用作缓存项的key的字符串
  示例：fastcgi_cache_key $request_rui;
- 7、fastcgi_cache_methods GET | HEAD | POST ...;
  为哪些请求方法使用缓存
- 8、fastcgi_cache_min_uses number;
  缓存空间中的缓存项在inactive定义的非活动时间内至少要被访问到此处所指定的次数方可被认作活动项
- 9、fastcgi_keep_conn on | off;
  收到后端服务器响应后，fastcgi服务器是否关闭连接，建议启用长连接
- 10、fastcgi_cache_valid [code ...] time;
  不同的响应码各自的缓存时长

ngx_http_fastcgi_module

- 示例：

```
http {
        fastcgi_cache_path /var/cache/nginx/fcgi_cache levels=1:2:1
keys_zone=fcgicache:20m inactive=120s;
        ...
        server {
                location ~* \.php$ {
                ...
                fastcgi_cache fcgicache;
                fastcgi_cache_key $request_uri;
                fastcgi_cache_valid 200 302 10m;
                fastcgi_cache_valid 301 1h;
                fastcgi_cache_valid any 1m;
        ...
        }
}
```

练习

- 定义四个虚拟主机，混合使用三种类型的虚拟主机
  仅开放给来自于本地网络中的主机访问
- 实现lnmp，提供多个虚拟主机
  - http, 提供wordpress
  - https, 提供pma

ngx_http_upstream_module

- ngx_http_upstream_module模块

  用于将多个服务器定义成服务器组，而由proxy_pass, fastcgi_pass等指令进行引用

- 1、upstream name { ... }

  定义后端服务器组，会引入一个新的上下文

  默认调度算法是wrr

  Context: http

  ```
  upstream httpdsrvs {
          server ...
          server...
          ...
  }
  ```

ngx_http_upstream_module

◆ 2、server address [parameters];

在upstream上下文中server成员，以及相关的参数；Context:upstream

address的表示格式：

unix:/PATH/TO/SOME_SOCK_FILE

IP[:PORT]

HOSTNAME[:PORT]

parameters：

weight=number　　权重，默认为1

max_conns　连接后端报务器最大并发活动连接数，1.11.5后支持

max_fails=number　　失败尝试最大次数；超出此处指定的次数时，server将被标记为不可用,默认为1

fail_timeout=time　　后端服务器标记为不可用状态的连接超时时长，默认10s

backup将服务器标记为“备用”，即所有服务器均不可用时才启用

down　标记为“不可用”，配合ip_hash使用，实现灰度发布

ngx_http_upstream_module

- 3、ip_hash　　源地址hash调度方法
- 4、least_conn　最少连接调度算法，当server拥有不同的权重时其为wlc，当所有后端主机连接数相同时，则使用wrr，适用于长连接
- 5、hash key [consistent]　　基于指定的key的hash表来实现对请求的调度，此处的key可以直接文本、变量或二者组合

作用：将请求分类，同一类请求将发往同一个upstream server，使用consistent参数，将使用ketama一致性hash算法，适用于后端是Cache服务器（如varnish）时使用

hash $request_uri consistent;

hash $remote_addr;

- 6、keepalive 连接数N;

为每个worker进程保留的空闲的长连接数量,可节约nginx端口，并减少连接管理的消耗

ngx_http_upstream_module

- 7、health_check [parameters];

健康状态检测机制；只能用于location上下文

常用参数：

interval=time检测的频率，默认为5秒

fails=number：判定服务器不可用的失败检测次数；默认为1次

passes=number：判定服务器可用的失败检测次数；默认为1次

uri=uri：做健康状态检测测试的目标uri；默认为/

match=NAME：健康状态检测的结果评估调用此处指定的match配置块

注意：仅对nginx plus有效

ngx_http_upstream_module

- 8 match name { ... }

对backend server做健康状态检测时，定义其结果判断机制；只能用于http上下文

- 常用的参数：

status code[code ...]: 期望的响应状态码

header HEADER[operator value]：期望存在响应首部，也可对期望的响应首部的值基于比较操作符和值进行比较

body：期望响应报文的主体部分应该有的内容

注意：仅对nginx plus有效

ngx_stream_core_module

- nginx的其它的二次发行版：

  Tengine：由淘宝网发起的Web服务器项目。它在Nginx的基础上，针对大访问量网站的需求，添加了很多高级功能和特性。Tengine的性能和稳定性已经在大型的网站如淘宝网，天猫商城等得到了很好的检验。它的最终目标是打造一个高效、稳定、安全、易用的Web平台。从2011年12月开始，Tengine成为一个开源项目，官网 http://tengine.taobao.org/

  OpenResty：基于 Nginx 与 Lua 语言的高性能 Web 平台

- ngx_stream_core_module模块

  模拟反代基于tcp或udp的服务连接，即工作于传输层的反代或调度器

ngx_stream_core_module

◆ 1、stream { ... }

定义stream相关的服务；Context:main

```
stream {
        upstream mysqlsrvs {
                server 192.168.22.2:3306;
                server 192.168.22.3:3306;
                least_conn;
        }
        server {
                listen 10.1.0.6:3306;
                proxy_pass mysqlsrvs;
        }
}
```

ngx_stream_core_module

- 2、listen

listen address:port [ssl] [udp] [proxy_protocol] [backlog=number] [bind] [ipv6only=on|off] [reuseport] [so_keepalive=on|off|[keepidle]:[keepintvl]:[keepcnt]];

ngx_stream_proxy_module

- ngx_stream_proxy_module模块
  可实现代理基于TCP，UDP (1.9.13), UNIX-domain sockets的数据流
- 1 proxy_pass address;
  指定后端服务器地址
- 2 proxy_timeout timeout;
  无数据传输时，保持连接状态的超时时长
  默认为10m
- 3 proxy_connect_timeout time;
  设置nginx与被代理的服务器尝试建立连接的超时时长
  默认为60s

示例


```
stream {
        upstream mysqlsrvs {
                server 192.168.0.10:3306;
                server 192.168.0.11:3306;
                hash $remote_addr consistent;
        }
        server {
                listen 172.16.100.100:3306;
                proxy_pass mysqlsrvs;
                proxy_timeout 60s;
                proxy_connect_timeout 10s;
        }
}
```

练习

- nginx--> AMPs（ wordpress ）
- nginx--> FPMs（ wordpress ）
- 自定义错误404和5xx错误页，文本静态内容传输压缩
- 实现动静分离：动态资源存储一组服务器、图片资源存在一组服务器、静态的文本类资源存储在一组服务器

```
nginx-->images servers ( imgs.magedu.com )
        location ~* \.(jpg|png|gif|jpeg)$ {
                ...
        }
nginx-->dynamic content servers (shop.magedu.com)
        location ~* \.php$ {
                ...
        }
```

关于马哥教育

马哥教育
IT人的高薪职业学院

- 博客：http://mageedu.blog.51cto.com
- 主页：http://www.magedu.com
- QQ：1661815153, 113228115
- QQ群：203585050, 279599283

祝大家学业有成

谢　谢