Nginx基础实践总结

> 在系统项目开发维护过程中，免不了与Nginx打交道，本次文章主要为总结记录下在过往实践过程的经验，作为抛砖引玉。

Nginx基础实践总结

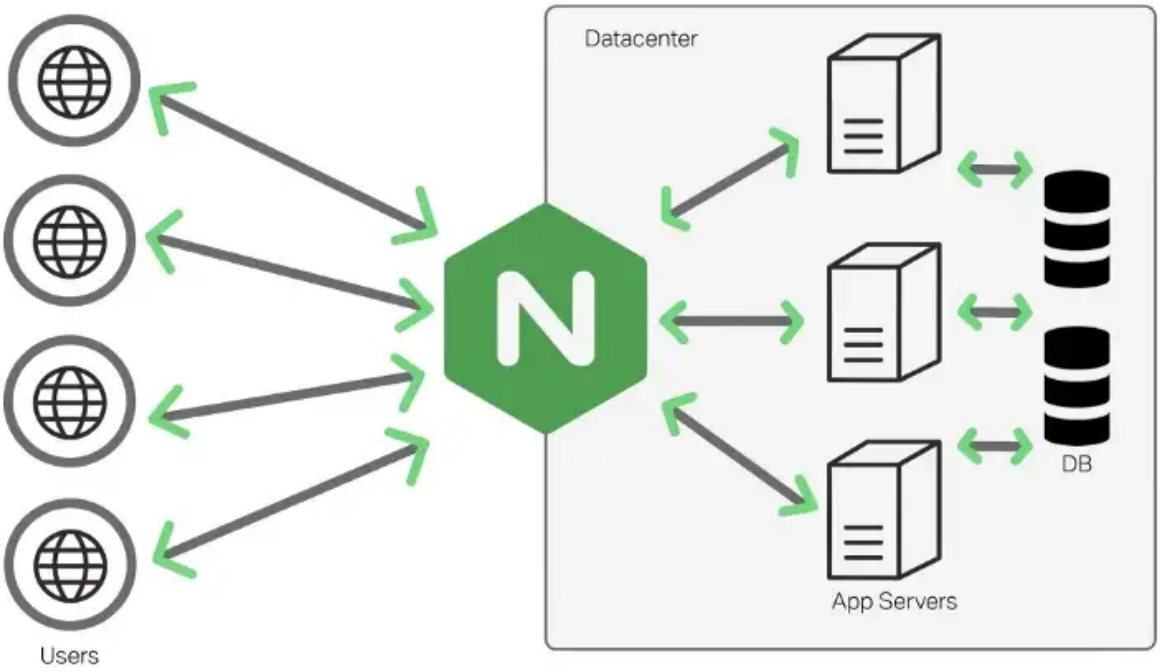
- 引言
- Nginx配置
  - worker_processes
  - worker_connections
  - keepalive_timeout
  - client_max_body_size
  - log_format
  - server_name
  - location
- Nginx进程
- Nginx踩坑
  - 配置静态资源访问
  - 反向代理
    - 域名解析
    - 加载顺序
  - 错误码
  - 配置参数优化
- Nginx灰度发布
- 后记

引言

以校招系统为例，系统由最初单体应用逐步迭代为单体应用和一组多个功能模块的微服务。通过Nginx的反向代理完成不同功能模块的请求转发。 同时也从AWS单机部署迁移到了丹炉云平台的容器化部署，在基础Docker镜像用也同步整合了Nginx相关配置信息。

作为一款高可用，高并发，热部署，高扩展，低消耗的轻量级HTTP服务器，Nginx在系统平稳运行和重构重构开发流程中，从Web Server出发，更多地充当了反向代理、负载均衡等方面的不可或缺的角色。

在一个完整的HTTP请求处理流程中，Nginx会根据Http请求中的Header信息和Nginx配置文件中的信息进行匹配对应的Http块，决定当前请求（静态请求或者是动态请求），直接返回或者是交由上游特定的服务器进行业务逻辑处理。

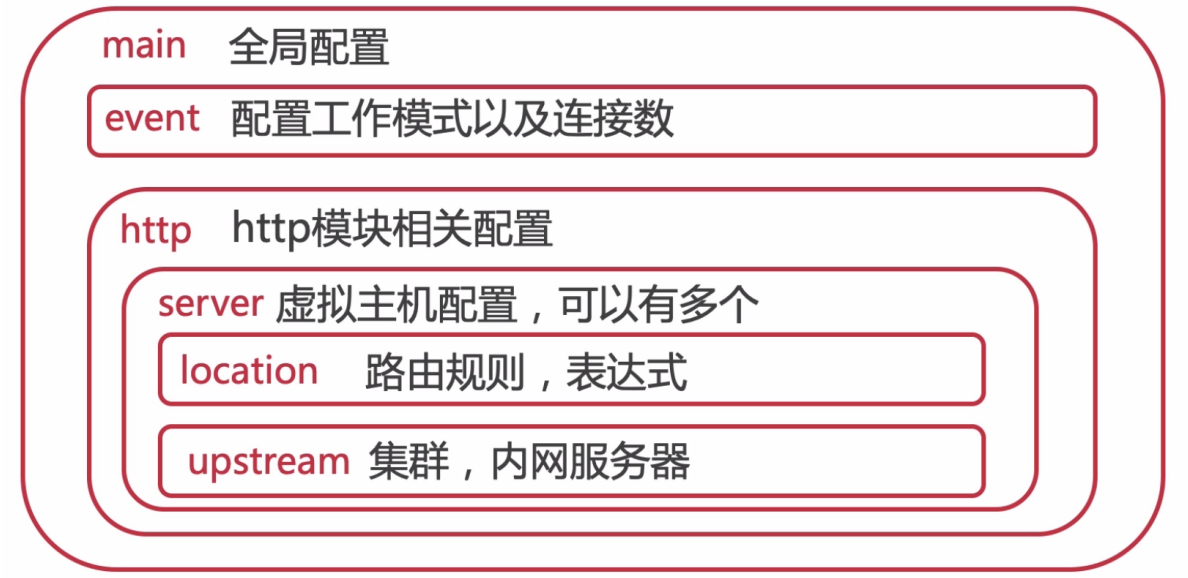


Nginx配置

作为Nginx的主配置文件Nginx.conf，Nginx服务启动时会载入配置文件，配置文件中的每一行都是一个命令，支持大量可配置的指令。同时生成Master-Worker 的多进程模型处理Http请求。

Nginx配置文根据简化维护设计思路，同时也支持将不同部分的配置放置到单独的配置文件中，自动包含到主配置文件中。 Nginx配置文件主要分为以下部分：

- main（全局设置，作用域为Nginx服务器全局环境）
- event（Nginx服务器与用户网络连接配置）
- http（代理、缓存以及日志等供以及第三方模块配置）
- server（虚拟主机设置）
- location（URL匹配特定位置后的设置）
- upstream（上游服务器设置，主要为反向代理、负载均衡相关配置）



以校招系统的Nginx配置文件为例，在不断开发迭代的过程中，主配置文件和拆分的具体域名配置文件如下，主要整理下常用的一些参数的使用方法。

主配置文件：

```
user application;
worker_processes 1;
worker_rlimit_nofile 65535;
pid /run/nginx.pid;

events {
        worker_connections 65535;
        use epoll;
}

http {
        sendfile on;
        tcp_nopush on;
        tcp_nodelay on;
        keepalive_timeout 120;
        types_hash_max_size 2048;
        server_tokens off;

        client_max_body_size 50m;

        include /etc/nginx/mime.types;
        default_type application/octet-stream;

        ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
        ssl_prefer_server_ciphers on;

        access_log /docker.stdout ;
        error_log /docker.stderr ;

        gzip on;
        gzip_disable "msie6";

        log_format access '$remote_addr - $remote_user [$time_local] "$request" $status
$body_bytes_sent $request_length "$request_body" "$http_referer" "$http_user_agent"
"$http_x_forwarded_for" $request_time $upstream_response_time';

        include /etc/nginx/conf.d/*.conf;
        include /etc/nginx/sites-enabled/*;
}
```

具体域名配置文件最小集配置：

```
server {
    listen 80;
    server_name xiaozhao-test.leihuo.netease.com consumemgr-test.apps-hp.danlu.netease.com;
    root /srv/sites/phpapi/consumemgr.leihuo.netease.com/public;
    index index.html index.htm index.php;
    client_body_buffer_size 850k;
    client_max_body_size 50m;

    # 设置指定日志格式
    access_log /srv/logs/nginx/consumemgr.access.log access;
    error_log /srv/logs/nginx/consumemgr.error.log;

    location / {
        #include snippets/fastcgi-php.conf;
        rewrite ^/(.*)$ /index.php?&query_string break;
        fastcgi_pass unix:/var/run/php-fpm.sock;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        fastcgi_read_timeout 300;
        include fastcgi_params;
    }
}
```

worker_processes

对应为Nginix启动时对应生成的工作线程数，可以设置固定的进程数，也可使用自动模式（auto）。 例如worker_processes设置为4时，会创建1个master进程和4个worker进程。

需要注意的是，需要根据部署环境资源情况考虑，一般默认情况为,不用修改。官方建议可以修改为CPU的内核数，以提高性能。 目前系统部署在丹炉平台，设置参数与应用对应的CPU核数保持一致即可。 但需要注意的是，在丹炉平台中，如果设置为自动模式，则会读取当前Pod容器对应Node节点的CPU值来设置Nginx的工作线程数量。 这里实际读取的数量与应用配置的值不一致，会导致Nginx工作进程数量过大，容易启发应用OOM而不断重新启动的问题。

worker_connections

单个进程的最大连接数。可以根据硬件资源进行调整，默认是1024。 可以增加到1024或更高的值，以允许同时处理更多连，但是需要注意需要小于系统进程的最大打开文件数量（一般为65535）。

keepalive_timeout

用于指定KeepAlive的超时时间，即每一个TCP链接最大保持时间，默认为75秒。 对于一些特定场景下请求开销比较大的情况，可以适当调整为更大值（例如120秒）。

client_max_body_size

客户端请求服务器上传文件的最大允许值，默认为1M。 这里可以根据实际情况进行调整（例如50M）。 由于一个Http请求除了会经过Nginx这一层外，还会在对应业务框架等不同阶段遇到对应的限制配置，也需要同步修改。 例如，在Springboot中，需要修改内置Tomcat的maxPostsize值，以及Springmvc设置文件上传大小限制spring.servlet.multipart.max-file-size值，与Nginx配置保持一致。在PHP中，需要同步调整post_max_size和upload_max_filesize字段对应值，与Nginx配置保持一致。

log_format

用于定义日志格式，设置完成后，可以应用到对应access_log等地方，例如"access_log log/path access;" 在实际应用中，可以通过自定义$request_time、$upstream_response_time等参数增加请求处理时间记录， 便于通过日志分析具体接口请求处理时间在哪些场景和时间段比较慢，进行针对性的排查优化。

server_name

用于配置虚拟主机的名称。不同的域名会根据Header中的Host信息匹配到特定的server块中，进入对应应用服务处理流程中。 通常情况下，server_name一对一即可，在丹炉平台中应用可以设置一个自定义域名（*.apps-hp.danlu.neatease.com），同时也可以指定固定域名（xiaozhao-test.leihuo.netease.com）。这里可以用空格间隔，第一个域名名称为此虚拟主机的主要名称。 需要注意的是，例如在PHP中调用$_SERVER["SERVER_NAME"]时，获取的始终是server_name中的第一个值，容易在业务逻辑相关域名判断中引发错误。 这里是由于在Nginx中fastcgi_params配置文件中的定义为`fastcgi_param SERVER_NAME $server_name;`导致， 一般解决方法可以是调整server_name的顺序和分拆为多个server块，保证一个域名对应一个server块。也可以通过修改SERVER_NAME的定义（即修改为`fastcgi_param SERVER_NAME $host;`）

同时server_name还支持通配符和正则表达式两种配置名称的方式。 优先级方面，准确匹配>通配符在开始时匹配>通配符在结尾时匹配>正则表达式匹配，同时以首次匹配优先。

location

用于对除了server_name外的URL部分进行匹配处理，一般地址重定向以及第三方模块配置都是在location块中提供。 location的语法结构为：`location [ = | ~ | ~* | ^~ ] uri { ... }`

方括号对应的是请求字符串与uri的匹配方式，当不添加匹配方式时，按照遵循普通匹配（最大前缀匹配规则，记录匹配度最大的一个，然后用正则匹配，如果正则失效则对应匹配度最大的这一个location块进行处理）。

对于具体匹配方式来说，具体如下：

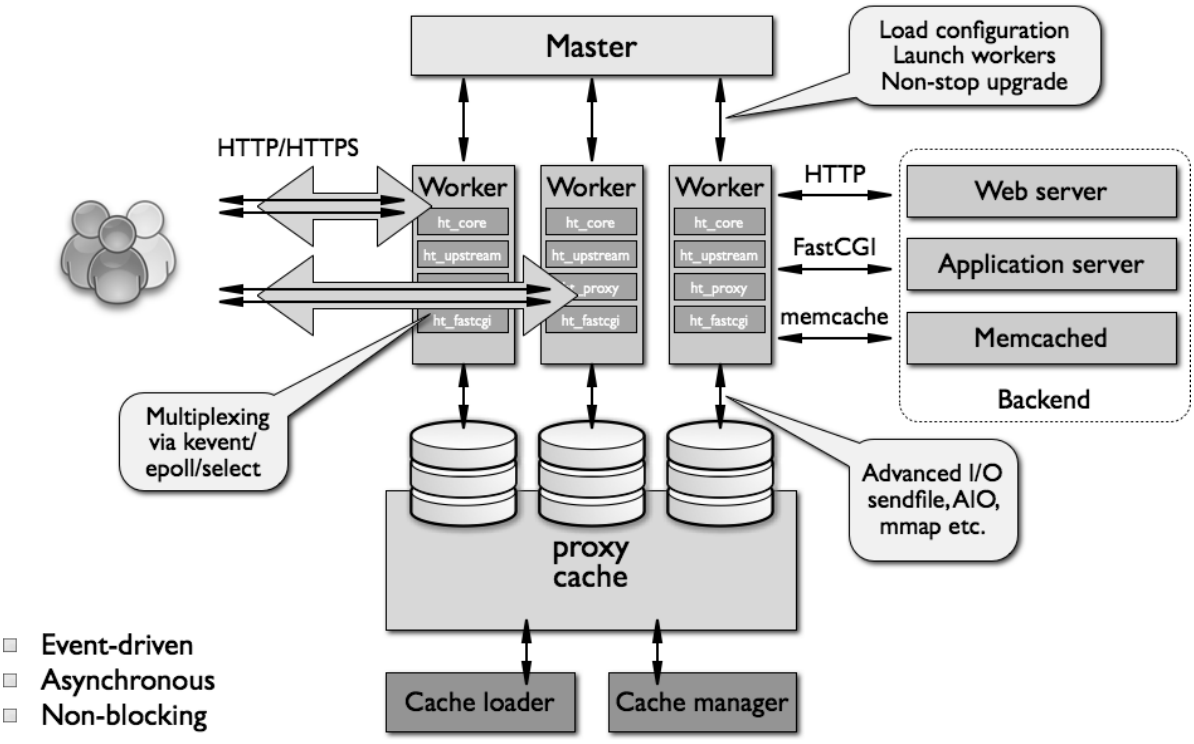
匹配符	匹配规则	优先级	备注
=	精确匹配	1	匹配成功后终止搜索
^~	以某个字符串开头	2	匹配成功且为最大前缀匹配后终止搜索
~	区分大小写的正则匹配	3	第一个匹配成功后终止搜索
~*	不区分大小写的正则匹配	4	第一个匹配成功后终止搜索
/	通用匹配	5	

uri为待匹配的请求字符串，可以为不包含正则表达式的字符串，同时也支持包含正则表达式。

前缀字符串匹配和在文件中的位置无关，但是和匹配长度有关。正则匹配和location规则在文件中的位置有关。

这里也是会经常踩坑的地方之一。

Nginx进程



Nginx采用多进程模型，Nginx启动以后以守护进程的方式在后台运行，会产生一个master主进程，主进程执行一系列的工作后会产生一个或者多个工作进程worker进程。 在客户端请求动态站点的过程中，Nginx服务器还涉及和后端服务器的通信。Nginx将接收到的 Web请求通过代理转发到后端服务器，由后端服务器进行数据处理和组织； Nginx为了提高对请求的响应效率，降低网络压力，采用了缓存机制，将历史应答数据缓存到本地。保障对缓存文件的快速访问。

同时借助于master-worker进程结构，Nginx可实现热部署。Nginx通过信号控制Nginx进程，我们可以通过信号量完成Nginx的重载以及升级。 以经常使用的Nginx重载配置文件为例， `nginx -s reload` 具体流程如下：

- 向master进程发送HUP信号（reload命令）
- master检查配置文件是否正确
- master进程打开监听端口
- master进程使用新的配置文件启动新的worker子进程
- master进程向老的worker子进程发送QUIT信号
- 旧的worker进程关闭监听句柄，处理完当前连接后关闭进程

此外，也会用到USR2信号，用于热部署以及平滑升级：`kill -USR2 旧版程序的主进程号或进程文件名`

Nginx踩坑

配置静态资源访问

在微信公众号以及伯乐系统等场景下，需要将校验文件放在域名下以供校验，实现文件高效快速的访问。

```
location /MP_verify_NgcEP4Ksr7PH6CMY.txt {
    alias /srv/sites/phpapi/consumemgr.leihuo.netease.com/public/MP_verify_NgcEP4Ksr7PH6CMY.txt;
}
```

此外，对于开发联调中的资源共享场景，通过autoindex（目录浏览功能），可以在本地快速搭建一个文件服务器，提供目录浏览提供下载功能。

反向代理

目前校招系统在重构开发过程中，对于不同的功能模块接口，通过location块反向代理到不同应用中。

域名解析

通常反向代理中proxy_pass对应的应用的集群内地址或者是集群外地址，为域名形式。当proxy_pass指令带有变量名时，需要设置resolver，不然会提示“no resolver defined to resolve xxxx.com”。 在丹炉平台中，可以添加`resolver kube-dns.kube-system.svc.cluster.local;`

```
location = /interview/collect/interviewer/show {
    # 丹炉平台集群内访问设置对应的DNS配置
    resolver kube-dns.kube-system.svc.cluster.local;
    rewrite ^/(interview/collect/interviewer/show) /v2/$1 break;
    proxy_pass $scheme://recruit-server-master.apps-qa.danlu.netease.com;
}
```

加载顺序

location中不同匹配模式对应的匹配规则和顺序都会有所不同，这里往往配置的不同会导致与预期不一致的结果。 例如大部分场景下，通过正则匹配相同接口前缀对应不同应用，由于正则匹配首次匹配成功后就会停止搜索，location的顺序就会影响匹配结果。如图中所示，预期test/admin前缀接口应该转发到test2域名，但是实际上会转发的test1域名，引发404等问题。

```
location ~* ^/test {
    rewrite ^/test/(.*) /$1 break;
    proxy_pass $scheme://test1.com;
}

location ~* ^/test/admin {
    rewrite ^/test/admin/(.*) /$1 break;
    proxy_pass $scheme://test2.com;
}
```

同时，与proxy_pass相结合使用的rewrite命令，结合正则表达式和标志位实现URL重写以及重定向。语法结构如下：`rewrite <regex> <replacement> [flag];`目前校招系统常用方法如下，通过rewrite重写URL匹配到不同应用对应的接口地址：

```
location ~* ^/neitui {
    resolver kube-dns.kube-system.svc.cluster.local;
    rewrite ^/neitui/(.*) /$1 break;
    proxy_pass $scheme://neitui-frontend.apps-hp.danlu.netease.com;
}

location ~* ^/api/v2/filters {
    resolver kube-dns.kube-system.svc.cluster.local;
    rewrite ^/(.*) /$1 break;
    proxy_pass $scheme://recruit-server-master.apps-qa.danlu.netease.com;
}

location = /offer/list {
    resolver kube-dns.kube-system.svc.cluster.local;
    rewrite ^/(offer/list) /$1 break;
    proxy_pass $scheme://recruit-server-master.apps-qa.danlu.netease.com;
}
```

rewrite中的flag标记，共有last、break、redirect和permanent四种，最常用的为break，即本条规则匹配完成即终止，不再匹配后面的任何规则。

错误码

当系统错误发生时，Nginx返回的错误码信息也是第一手资料进行问题分析排查，以下也是校招系统遇到过的常见错误码类型：

错误码	错误码含义	错误码发生场景	备注
404	访问文件或者接口不存在		404 Not Found
413	上传文件大小超过限制	没有及时更新默认配置参数	413 Request Entity Too Large
414	URL长度超过限制	GET请求URL参数过长	414 Request-URI Too Large
499	Nginx自定义	Nginx返回结果前主动断开客户端链接	业务处理时间过长

错误码	错误码含义	错误码发生场景	备注
500	内部错误	业务逻辑处理错误	500 Internal Server Error
502	错误网关	丹炉应用更新后会有短时间502	502 Bad Geteway
504	网关超时	业务处理时间过长	504 Gateway Time-out

面对错误情况，往往采用"问题现象确认——梳理访问请求流程——查看日志分析"思路，主要通过Nginx的error.log并结合业务框架中记录对应的业务日志进行结合分析排查修复问题。坚信所有的问题总能从日志中找到原因。

配置参数优化

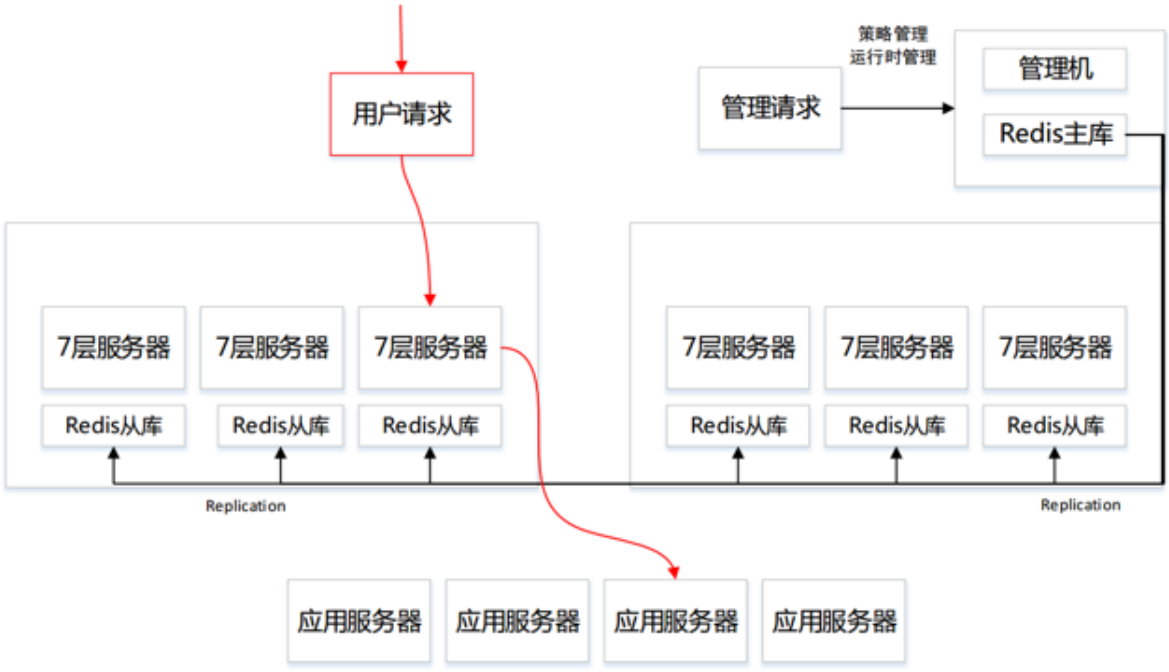
校招系统转为丹炉平台容器化部署后，从Nginx配置参数来说会更加容易和灵活，对于Nginx参数优化也会有更多的实践机会。 根据物理资源以及系统运行情况反馈，常用优化参数如下：

- 优化Nginx进程数量，对应worker_processes。
- 优化单个进程允许的客户端最大连接数，对应worker_connections。
- 优化worker 进程的最大打开文件数，对应worker_rlimit_nofile。
- 优化事件处理模型，对应event块。
- 优化TCP，对应sendfile、tcp_nopush以及tcp_nodelay。
- 优化压缩，对应Gzip压缩参数。

Nginx灰度发布

Nginx不仅仅有这些作用，还有更多强大功能可供探索实践。灰度发布就是其中的一个。

灰度发布（金丝雀发布，GrayRelease或Dark launch）是为了能够让用户逐步过渡到新功能一种发布方式。 A/B Test就是一种灰度发布方式，让一部用户继续用A，一部分用户开始用B，如果用户对B没有什么反对意见，那么逐步扩大范围，把所有用户都迁移到B上面 来。灰度发布可以保证整体系统的稳定，在初始灰度的时候就可以发现、调整问题，以保证其影响度。 在客户端方面，客户端依据特殊条件，例如用户ID/IP等实现分流。 在服务端灰度，依托Nginx反向代理，对不同IP/用户ID等条件实现分流。



目前开源项目中，ABTestingGateway是一个可以动态设置分流策略的灰度发布系统。基于Nginx和ngx-lua开发，使用 redis作为分流策略数据库，可以实现动态调度功能。支持iprange: ip段分流、uidrange: 用户uid段分流、uidsuffix: uid尾数分流以及uidappoint: uid白名单分流等多种策略。

demo实践效果如下：

```
{
    "1":{
        "divtype":"uidsuffix",
        "divdata":[
            {
                "suffix":"1",
                "upstream":"beta1"
            },
            {
                "suffix":"3",
                "upstream":"beta2"
            },
            {
                "suffix":"5",
                "upstream":"beta1"
            },
            {
                "suffix":"0",
                "upstream":"beta3"
            }
        ]
    },
    "2":{
        "divtype":"arg_city",
        "divdata":[
            {
                "city":"BJ",
                "upstream":"beta1"
            },
            {
                "city":"SH",
                "upstream":"beta2"
            },
            {
                "city":"XA",
                "upstream":"beta1"
            },
            {
                "city":"HZ",
                "upstream":"beta3"
            }
        ]
    },
    "3":{
        "divtype":"iprange",
        "divdata":[
            {
                "range":{
                    "start":1111,
                    "end":2222
                },
                "upstream":"beta1"
            },
            {
                "range":{
                    "start":3333,
                    "end":4444
                },
                "upstream":"beta2"
            },
            {
                "range":{
                    "start":7777,
                    "end":2130706433
                },
                "upstream":"beta2"
            }
        ]
    }
}
```

```
maling1@HIH-D-18049:~$ curl 127.0.0.1:8030 -H 'X-Uid:39' -H 'X-Real-IP:192.168.1.1'
this is stable server
maling1@HIH-D-18049:~$ curl 127.0.0.1:8030 -H 'X-Uid:30' -H 'X-Real-IP:192.168.1.1'
this is beta3 server
maling1@HIH-D-18049:~$ curl 127.0.0.1:8030/?city=BJ -H 'X-Uid:39' -H 'X-Real-IP:192.168.1.1'
this is beta1 server
maling1@HIH-D-18049:~$
```

校招系统目前更新频率较为频繁，在开发过程中，新旧版本功能开发并行。 HR优先验收指定功能，QA需要同时验证多个功能对，诸多场景都需要不同环境对应不同功能内容。 目前通过独立部署多套环境用最简单的方法去实现，不能叫做真正的“灰度发布”，灰度发布和分流可以及早获取用户反馈，降低升级影响用户范围，提升产品质量，这也是后续系统开发努力的方向之一。

后记

感谢大家的阅读，本文是对于系统开发过程中在Nginx实践以及学习过程中比较浅显的总结，主要是日常开发以及部署中的技巧和实践方法，希望可以给大家带来一些帮助。对于更多深入的例如Nginx源码，Nginx与Lua相结合的Openresty开发等有很多方向可以在后续系统中进行实践，当做抛砖引玉，与大家共同学习。