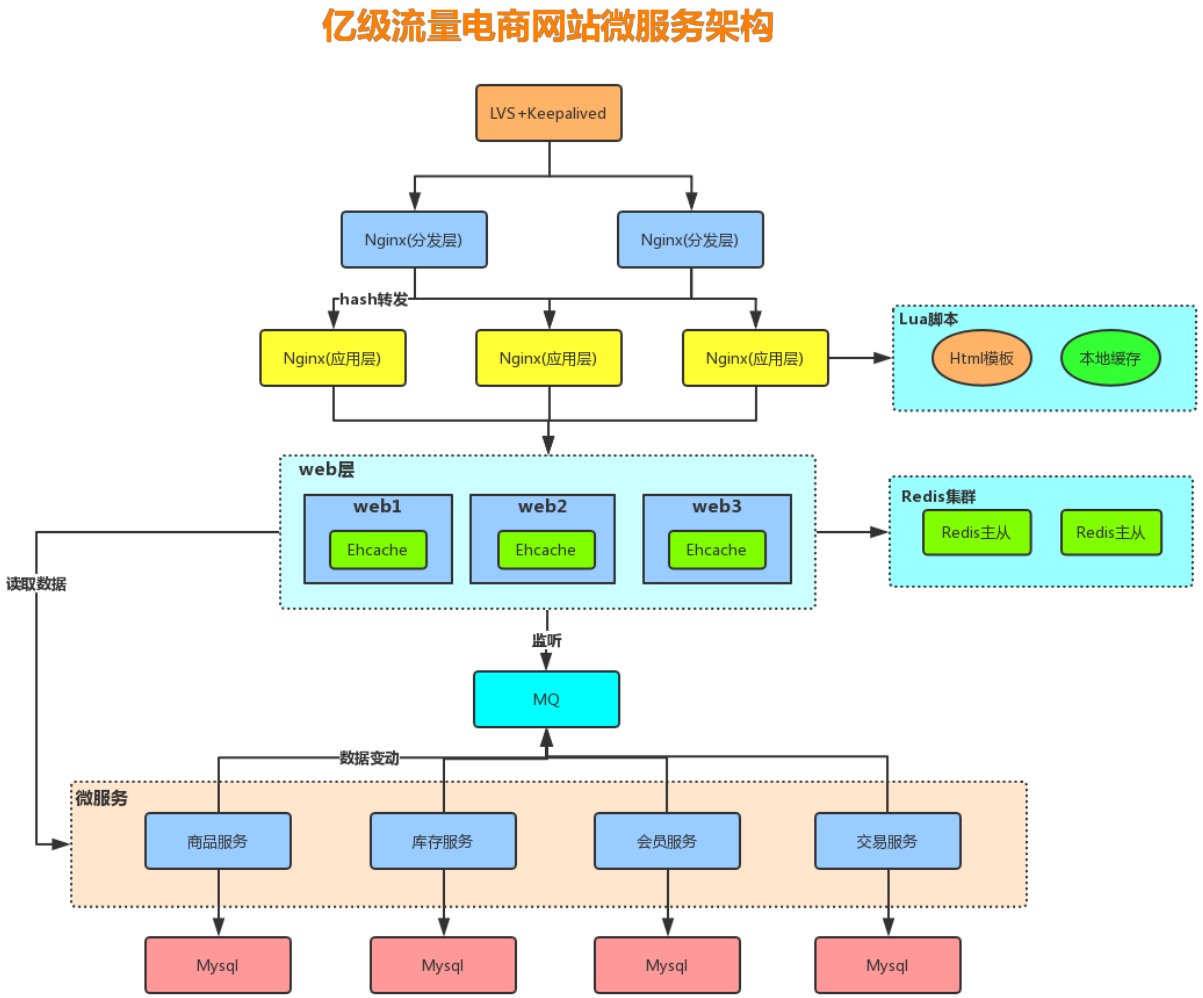
大型网站后台架构图



Openresty安装

```
mkdir -p /usr/servers
cd /usr/servers/

yum install -y readline-devel pcre-devel openssl-devel gcc

wget http://openresty.org/download/ngx_openresty-1.7.7.2.tar.gz
tar -xzvf ngx_openresty-1.7.7.2.tar.gz
cd /usr/servers/ngx_openresty-1.7.7.2/

 cd bundle/LuaJIT-2.1-20150120/
 make clean && make && make install
 ln -sf luajit-2.1.0-alpha /usr/local/bin/luajit

 cd /usr/servers/ngx_openresty-1.7.7.2/bundle
 wget https://github.com/FRiCKLE/ngx_cache_purge/archive/2.3.tar.gz
 tar -xvf 2.3.tar.gz

 cd bundle
 wget https://github.com/yaoweibin/nginx_upstream_check_module/archive/v0.3.0.tar.gz
 tar -xvf v0.3.0.tar.gz

```

```
cd /usr/servers/ngx_openresty-1.7.7.2
./configure --prefix=/usr/servers --with-http_realip_module --with-pcre --with-luajit --add-module=./bundle/ngx_cache_purge-2.3/ --add-module=./bundle/nginx_upstream_check_module-0.3.0/ -j2
make && make install

cd /usr/servers/
ll

/usr/servers/luajit
/usr/servers/lualib
/usr/servers/nginx

/usr/servers/nginx/sbin/nginx -V

启动nginx: /usr/servers/nginx/sbin/nginx
```

nginx+lua开发的hello world

```
cd /usr/servers/nginx/conf

vim lua.conf
server {
 listen 80;
 server_name _;

 location /lua {
 default_type 'text/html';
  content_by_lua 'ngx.say("hello world")';
  }
}

vim nginx.conf
在http部分引入lua包
lua_package_path "/usr/servers/lualib/?.lua;;";
lua_package_cpath "/usr/servers/lualib/?.so;;";
include lua.conf;

../sbin/nginx -s reload

访问: http://192.168.0.60/lua
```

nginx+lua开发的流量分发逻辑

```
流量分发的nginx，会发送http请求到后端的应用层nginx上去，所以要先引入lua http lib包
cd /usr/servers/lualib/resty
wget https://raw.githubusercontent.com/pintsized/lua-resty-http/master/lib/resty/http_headers.lua
wget https://raw.githubusercontent.com/pintsized/lua-resty-http/master/lib/resty/http.lua

cd /usr/servers/nginx/conf

vim lua-distribution.lua
local uri_args = ngx.req.get_uri_args()
 local productId = uri_args["productId"]
 local host = {"192.168.0.61", "192.168.0.62"}
 local hash = ngx.crc32_long(productId)
 hash = (hash % 2) + 1
 backend = "http://"..host[hash]
```

```
local method = uri_args["method"]
local requestBody = "/"..method.."?productId="..productId
local http = require("resty.http")
local httpc = http.new()
local resp, err = httpc:request_uri(backend, {
 method = "GET",
 path = requestBody,
 keepalive=false
})
if not resp then
 ngx.say("request error :", err)
 return
end
ngx.say(resp.body)
httpc:close()

vim lua.conf
在server部分加入
location /product {
 default_type 'text/html';
 content_by_lua_file /usr/servers/nginx/conf/lua-distribution.lua;
 }

../sbin/nginx -s reload

访问：http://192.168.0.60/lua?productId=XX
会根据productId将请求分发到不同的应用层nginx
```

nginx+lua开发应用层页面缓存与模板动态渲染逻辑

```
应用层需要访问服务http接口，所以也需要引入lua http lib包
cd /usr/servers/lualib/resty
wget https://raw.githubusercontent.com/pintsized/lua-resty-http/master/lib/resty/http_headers.lua
wget https://raw.githubusercontent.com/pintsized/lua-resty-http/master/lib/resty/http.lua

应用层还需要用到模板动态渲染技术，所以还需要引入lua的template包
cd /usr/servers/lualib/resty
wget https://raw.githubusercontent.com/bungle/lua-resty-template/master/lib/resty/template.lua
mkdir /usr/servers/lualib/resty/html
 cd /usr/servers/lualib/resty/html
 wget https://raw.githubusercontent.com/bungle/lua-resty-template/master/lib/resty/template/html.lua

 增加html静态模板
 cd /usr/servers/templates
 vim product.html
 <html>
 <head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
 </head>
 <body>
 商品id: {* productId *}<br/>
 商品名称: {* productName *}<br/>
 商品原价: {* productPrice *}<br/>
 商品现价: {* productNowPrice *}<br/>
 商品库存: {* productStock *}<br/>
 商品描述: {* productHTML *}<br/>
 </body>
```

```
</html>

cd /usr/servers/nginx/conf
vim lua.conf
在server部分加入
set $template_location "/templates";
set $template_root "/usr/servers/templates";

增加ngxin缓存配置
vim lua.conf
在最前面加入
lua_shared_dict my_cache 128m;

增加商品详情页渲染的lua脚本
vim lua.conf
在server部分增加
location /product {
 default_type 'text/html';
 content_by_lua_file /usr/servers/nginx/conf/product.lua;
 }

最后，编写商品详情页渲染lua脚本
vim product.lua
local uri_args = ngx.req.get_uri_args()
local productId = uri_args["productId"]
local cache_ngx = ngx.shared.my_cache
local productCacheKey = "product_info_"..productId
local productCache = cache_ngx:get(productCacheKey)
if productCache == "" or productCache == nil then
local http = require("resty.http")
local httpc = http.new()
local resp, err = httpc:request_uri("http://192.168.0.175:8080",{
method = "GET",
path = "/shop-web/product/cache/"..productId,
keepalive=false
})
productCache = resp.body
local expireTime = math.random(600,1200)
cache_ngx:set(productCacheKey, productCache, expireTime)
end
ngx.log(ngx.ERR,"json--------2", productCache)
local cjson = require("cjson")
local productCacheJSON = cjson.decode(productCache)
local context = {
productId = productCacheJSON.id,
productName = productCacheJSON.name,
productPrice = productCacheJSON.price,
productNowPrice = productCacheJSON.nowPrice,
productStock = productCacheJSON.stock,
productHTML = productCacheJSON.productHTML
}
local template = require("resty.template")
template.render("product.html", context)

../sbin/nginx -s reload

访问：http://192.168.0.60/product?productId=XX&method=product
会根据productId将请求分发到不同的应用层nginx获取相应的商品详情页面
```

86