Nginx+Lua网关

工具

被zhangjunfeng1添加，被zhangjunfeng1最后更新于六月 03, 2015

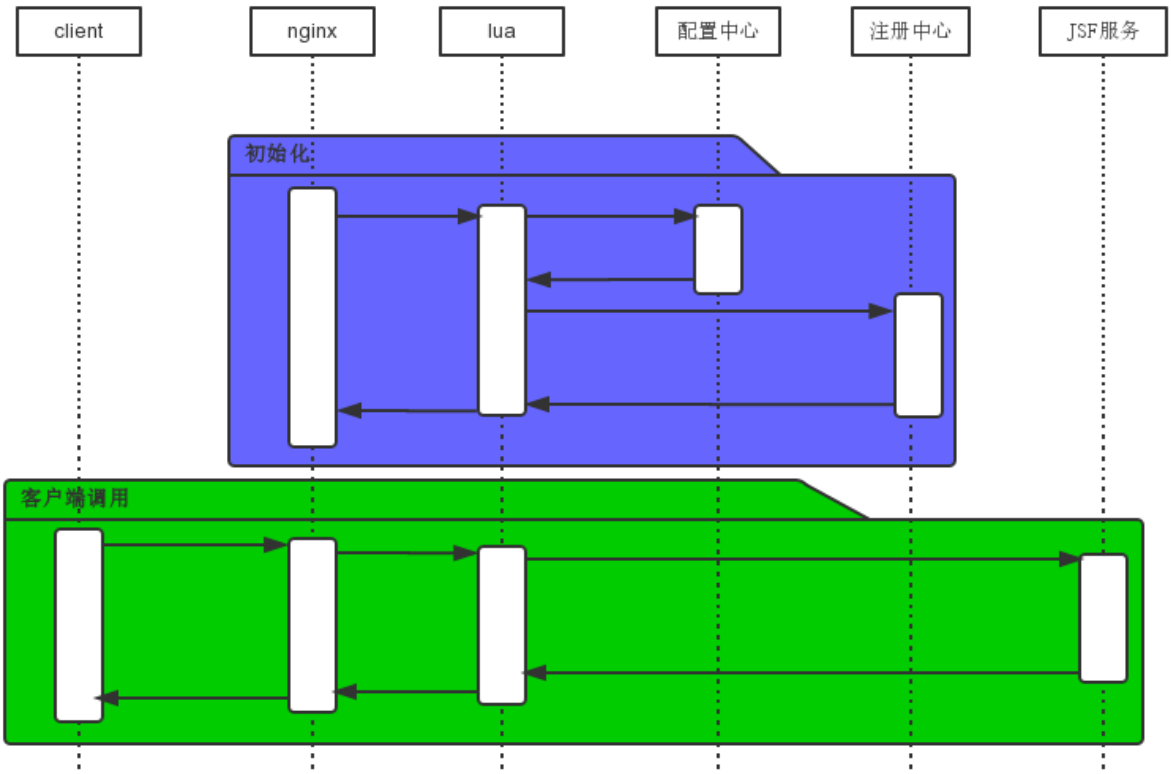
简介：

Nginx+lua配置的JSF网关，只供内网调用JSF服务使用

安装配置：

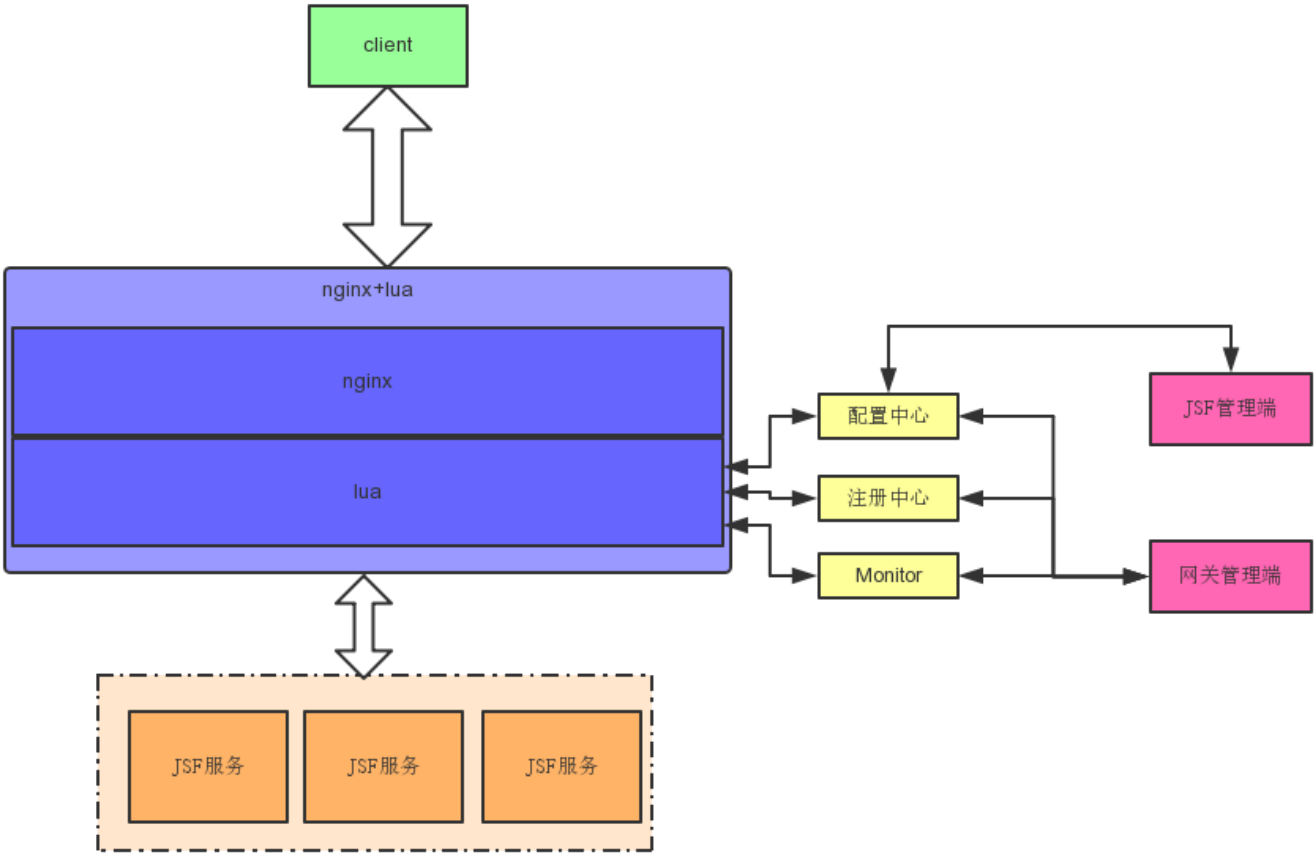
Nginx+lua环境搭建

业务流程图：



1. 应用方在JSF管理端(jsf.jd.com)申请对接口某个alias某个方法的网关调用
2. 网关server启动加载已开通网关调用的相关信息，接受请求调用
3. 应用通过http的方式调用网关，网关server接受请求代理请求对JSF服务进行调用，并将结果已json格式返回

架构图：



1、 Nginx：nginx启动时通过lua脚本从配置中心和注册中心取得开通网关调用的接口数据。启动后接收客户端的请求，把符合网关调用规则的URI转到lua中执行

2、Lua：启动时从配置中心和注册中心取得开通网关调用的接口数据。启动后接受符合网关调用的URL，解析URL后调用到适配的JSF服务端，取得返回数据返回给客户端。把调用耗时能数据发送到monitor服务上。

3、网关管理端：提供了对接口方法基本的网关关闭和开启功能，此次作为一个总的开关，如果此处接口的方法关闭了，则无法进行调用。管理端提供了监控数据的报表以及GW Server 状态查看功能，可以实时查看整个网关系统的运行基本情况

设计问题及实现方案：

问题**1**：

lua定时器：定时从配置中心取得开通网关的数据、及从注册中心取得接口的Provider列表

方案：在lua中使用nginx的延迟加载函数达到定时器功能。nginx延迟加载函数ngx.timer.at函数可以延迟指定秒数后执行指定函数，这样在函数内部再调用ngx.timer.at函数递归实现定时器功能。

代码：

定时器实例

```
local http = require("socket.http");
local json = require("json4lua");

timerTask = function()
   --[[
        定时执行功能代码
   ]]--
   ngx.timer.at(xx, timerTask);
end;

ngx.timer.at(yy, timerTask);
```

api说明：

ngx.timer.at(seconds， callback);

seconds：延迟秒数；

callback：执行的方法函数

问题**2**：

lua tcp连接池：lua语言是一个脚本语言，本身并不提供tcp连接池支持，那么如果和JSF服务端用长连接提高处理性能。

实现方案：配置完成以后，通过在nginx.conf中添加配置项来支持lua中的长连接；

配置如下：

nginx.conf

```
server{
      lua_socket_pool_size  1024; #最大长连接数
      lua_socket_keeplive_timeout   60s; #最大闲置时间

}
```

超过lua_socket_pool_size数后的连接都是短连接；

当一个连接超过lua_socket_keeplive_timeout设置的时间没有被使用后，会被释放。

问题**3**：

调用耗时计算：lua本身的时间处理只精确到秒，这样在计算调用耗时时不能进行精确统计。

实现方案：调用ngx.now解决。ngx.now()会返回一个浮点数，整数部分是秒，小数部分是毫秒。小数部分的位数是会变化的，例如：当毫秒是5毫秒时，会返回xxxx.5，即不会补0来保证三位数，所以取得结果后还需要进行判断计算。

使用指南：

Nginx+lua使用指南

代码示例：

1、socket使用代码示例：

公共函数：

公共函数

```
local floor = require'math'.floor

function bytesToString(bytes)
    local result = "";
    for i=1,#bytes do
        result = result..string.char(bytes[i]);
    end
    return result
end

--data为字符串，buffer为空byte数组，lua类型为table
function stringToBytes(data,buffer)
    if buffer == nil then
        buffer = {};
    end
    for i=1,#data do
        --local s = string.sub(data, i);
        buffer[#buffer+1] = string.byte(data,i);
    end
    return buffer;
end

function intToBytes(n)
    local result = {};
    result[1] = floor(n / 0x1000000);
    result[2] = floor(n / 0x10000) % 0x100;
    result[3] = floor(n / 0x100) % 0x100;
    result[4] = n % 0x100;
    return result;
end

function bytesToInt(bytes)
    local result = 0;
    result = bytes[1]*0x1000000 + bytes[2]*0x10000 + bytes[3]*0x100 + bytes[4];
    return result;
end

function shortToBytes(n)
    local result = {};
    result[1] = floor(n / 0x100) % 0x100;
    result[2] = n % 0x100;
    return result;
end

function bytesToShort(bytes)
    local result = 0;
    result = bytes[1]*0x100 + bytes[2];
    return result;
end
```

JSF请求用函数

```
--把invocation转为流形式的json
function getData(invocation)
    local values = "";
    if invocation.args ~= nil then
        for i=1, #invocation.args do
            if values ~= "" then
                values = values .. ",";
            end
            values = values .. invocation.args[i];
        end
    end
    local token = "";
    if invocation.token ~= nil then
        token = "\"token\":\"" .. invocation.token .. "\"";
    end
    local dataJson = "{\"clazzName\":\""..invocation.clazzName.. "\","
            .."\"alias\":\""..invocation.alias.."\","
            .."\"methodName\":\""..invocation.methodName.."\","
            --.."\"argsType\":[\""..types .."\"],"
            .."\"args\":[".. values .."],"
            .."\"attachments\":{"..token.."}}";
    return dataJson;
end


--组织invocation
function getInvocationJson(interfaceName, alias, methodName, data,token)
    local invocation = {};
    invocation.clazzName=interfaceName;
    invocation.alias=alias;
    invocation.methodName=methodName;
        --invocation.argsType={};
    invocation.args={data};
    if token ~= nil then
    invocation.token = token;
    end
    local sendJson= getData(invocation);
    return sendJson;
end

--组织JSF协议头。dataLen为数据长度，msgId为消息Id
function getProtoHeader(dataLen, msgId)
    local buffer = {}
    buffer[#buffer+1] = 0xAD;
    buffer[#buffer+1] = 0xCF;
    ------------------
    local intByte = intToBytes(dataLen+10+4);
    buffer[#buffer+1] = intByte[1];
    buffer[#buffer+1] = intByte[2];
    buffer[#buffer+1] = intByte[3];
    buffer[#buffer+1] = intByte[4];
    ------------------head len short
    buffer[#buffer+1] = 0x00;
    buffer[#buffer+1] = 0x08;
    ----protocol type
    buffer[#buffer+1] = 0x01;
    ----codec type
    buffer[#buffer+1] = 0x05;
    ----msg type
    buffer[#buffer+1] = 0x01;
    ----compress type
    buffer[#buffer+1] = 0x00;
    --int--msg id
    local msgByte = intToBytes(msgId);
    buffer[#buffer+1] = msgByte[1];
    buffer[#buffer+1] = msgByte[2];
    buffer[#buffer+1] = msgByte[3];
    buffer[#buffer+1] = msgByte[4];
    ----key map
--  buffer[#buffer+1] = 1;
--  buffer[#buffer+1] = 1;
    ---timeout
--  local timeout = 5000;
--  buffer[#buffer+1] = 0x00;
--  buffer[#buffer+1] = 0x00;
--  buffer[#buffer+1] = 0x00;
--  buffer[#buffer+1] = dataLen;
    return buffer;
end
```

①**lua**带**socket**示例：

参数说明：dataSend是要发送数据的byte数组。

lua socket

```
local socket = require("socket");
....

function sendAndGet(dataSend, ip, port)
    local tcpClient = socket.tcp();
    local client = tcpClient:connect(ip, port);
    tcpClient:settimeout(5000);
    tcpClient:setoption("tcp-nodelay", true);
    local result = "";
    local conn = tcpClient:send(dataSend);
    if conn == nil then
        return nil;
    end
    local chunk, status = tcpClient:receive(6);
    local data = string.sub(chunk, 3, 6);
    local len = bytesToInt(stringToBytes(data,nil));
    if status ~= "closed" and status ~= "timeout" then
        chunk, status = tcpClient:receive(len-4);
        if chunk ~= nil then
            result = result..chunk;
        end
    end
    tcpClient:close();
    local headerLen = bytesToShort(stringToBytes(string.sub(result,1,2),nil));
    result = string.sub(result, headerLen + 2+1, #result);
    return result;
end
```

②使用**nginx socket**示例：

前提是在nginx中配置：

```
lua_socket_pool_size  1024; #最大长连接数
lua_socket_keeplive_timeout   60s; #最大闲置时间
```

`local ok,err =` tcpClient:setkeepalive()表示放回连接池。

如果ok 为nil或者false，表示放回连接池失败。

参数说明：dataSend是要发送数据的byte数组。

nginx tcp

```
function sendAndGet(dataSend, ip, port)

    local tcpClient = ngx.socket.tcp();
    tcpClient:connect(ip, port);
    tcpClient:settimeout(5000);
    tcpClient:setoption("tcp-nodelay", true);
    local result = "";
    if tcpClient then
        assert(tcpClient:send(dataSend));
        local data = string.sub(chunk, 3, 6);
        local len = bytesToInt(stringToBytes(data,nil));
        if status ~= "closed" and status ~= "timeout" then
            chunk, status = tcpClient:receive(len-4);
            if chunk ~= nil then
                result = result..chunk;
            end
        end

    end
    local ok,err = tcpClient:setkeepalive();
    if not ok then
        assert(tcpClient:close());
    end
    local headerLen = bytesToShort(stringToBytes(string.sub(result,1,2),nil));
    result = string.sub(result, headerLen + 2+1, #result);
    return result;
end
```

请求**JSF**服务实例：

调用**JSF**服务实例：

```
--以调用注册中心lookup为例：
function getDataFromRC(iface, alias, addressList)
    --JsfUrl数据
    local requestTable = {['ip']='127.0.0.1',['iface']=iface, ['alias']=alias,['protocol']=1,
['insKey']='nginx_lua',['dataVer']='0'}
    local requestData = json.encode(requestTable);
    local sendData = getInvocationJson("com.jd.jsf.service.RegistryService", "reg", "lookup", requestData,
"1qaz2wsx");
    --组织消息头
    local headBuf = getProtoHeader(string.len(sendData), 1);
    --把请求数据加入消息体中（消息头的后面），组成一个完整数据包
    stringToBytes(sendData,headBuf);
    local dataSend = bytesToString(headBuf);
    local result;
    for i=1,#addressList do
        local addresses = split(addressList[i], ":");
        print(addresses[1]);
        result = sendAndGet(dataSend, addresses[1], addresses[2]);
        if result ~= nil then
            return result;
        end
    end
    return result;
end
```

2、**http**代码示例：

代码实例**1**：

一般处理GET方式：

直接请求

```
local http = require("socket.http");
……
function getAddressList()
    local providerDatas = {};
    local indexUrl = 'http://i.jsf.jd.com/addrs?sv=210&p=jsfRegistry&lan=java';
    local indexResult = http.request(indexUrl);
    if indexResult == nil then
        return nil;
    end
    local indexData = json.decode(indexResult);
    local address = indexData['address'];
    local addressList = split(address, ',');
    return addressList;
end
```

代码实例**2**：

支持GET和POST。

取得数据放在response_body中，以table方式存储，格式为：

response_body ={ "str1", "str2" , "str3"}

返回的数据长度最大长度为2k，超过2k会拆分为多个字符串按顺序放到response_body中。使用时需要遍历response_body组合为一个整体的字符串。

```
local http = require("socket.http");
local ltn12 = require("ltn12");
……
local regUrl = 'http://'..addressList[1]..'/com.jd.jsf.service.RegistryService/reg/lookup';
local requestTable = {['ip']='127.0.0.1',['iface']='com.jd.cap.data.api.service.XElasticService',
['alias']='fwen_join',['protocol']=1,['insKey']='nginx_lua',['dataVer']='0'}
local requestData = '[' .. json.encode(requestTable) ..']';

--local responseData = http.request(regUrl, requestData);
local response_body = {};
local lient, code, headers, status = http.request{url = regUrl,method = "POST",
    headers = {
        ["Content-Length"] = #requestData,
        ["token"] = "1qaz2wsx"
    },
    source = ltn12.source.string(requestData), sink = ltn12.sink.table(response_body)
}
```

	无