1.背景

1.1 什么是API网关

API网关可以看做系统与外界联通的入口，我们可以在网关进行处理一些非业务逻辑的逻辑，比如权限验证，监控，缓存，请求路由等等。

1.2 为什么需要API网关

RPC协议转成HTTP。

由于在内部开发中我们都是以RPC协议(thrift or dubbo)去做开发，暴露给内部服务，当外部服务需要使用这个接口的时候往往需要将RPC协议转换成HTTP协议。

请求路由

在我们的系统中由于同一个接口新老两套系统都在使用，我们需要根据请求上下文将请求路由到对应的接口。

统一鉴权

对于鉴权操作不涉及到业务逻辑，那么可以在网关层进行处理，不用下层到业务逻辑。

统一监控

由于网关是外部服务的入口，所以我们可以在这里监控我们想要的数据，比如入参出参，链路时间。

流量控制，熔断降级

对于流量控制，熔断降级非业务逻辑可以统一放到网关层。

有很多业务都会自己去实现一层网关层，用来接入自己的服务，但是对于整个公司来说这还不够。

1.3 统一API网关

统一的API网关不仅有API网关的所有的特点，还有下面几个好处:

统一技术组件升级

在公司中如果有某个技术组件需要升级，那么是需要和每个业务线沟通，通常几个月都搞不定。举个例子如果对于入口的安全鉴权有重大安全隐患需要升级，如果速度还是这么慢肯定是不行，那么有了统一的网关升级是很快的。

统一服务接入

对于某个服务的接入也比较困难，比如公司已经研发出了比较稳定的服务组件，正在公司大力推广，这个周期肯定也特别漫长，由于有了统一网关，那么只需要统一网关统一接入。

节约资源

不同业务不同部门如果按照我们上面的做法应该会都自己搞一个网关层，用来做这个事，可以想象如果一个公司有100个这种业务，每个业务配备4台机器，那么就需要400台机器。并且每个业务的开发RD都需要去开发这个网关层，去随时去维护，增加人力。如果有了统一网关层，那么也许只需要50台机器就可以做这100个业务的网关层的事，并且业务RD不需要随时关注开发，上线的步骤。

2.统一网关的设计

2.1 异步化请求

对于我们自己实现的网关层，由于只有我们自己使用，对于吞吐量的要求并不高所以，我们一般同步请求调用即可。

对于我们统一的网关层，如何用少量的机器接入更多的服务，这就需要我们的异步，用来提高更多的吞吐量。对于异步化一般有下面两种策略：

Tomcat/Jetty+NIO+servlet3

这种策略使用的比较普遍，京东，有赞，Zuul，都选取的是这个策略，这种策略比较适合HTTP。在Servlet3中可以开启异步。

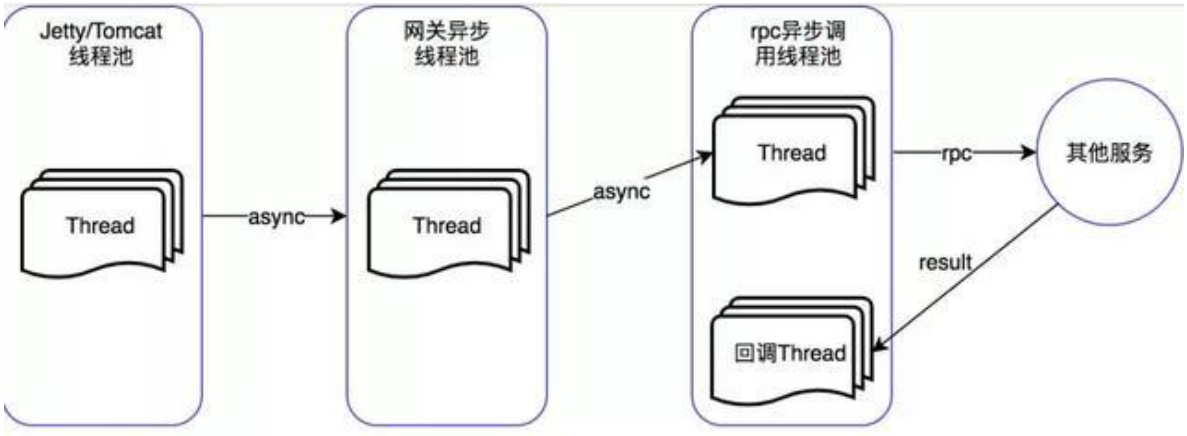
Netty+NIO

Netty为高并发而生，目前唯品会的网关使用这个策略，在唯品会的技术文章中在相同的情况下Netty是每秒30w+的吞吐量，Tomcat是13w+,可以看出是有一定的差距的，但是Netty需要自己处理HTTP协议，这一块比较麻烦。

对于网关是HTTP请求场景比较多的情况可以采用Servlet，毕竟有更加成熟的处理HTTP协议。如果更加重视吞吐量那么可以采用Netty。

2.1.1 全链路异步

对于来的请求我们已经使用异步了，为了达到全链路异步所以我们需要对去的请求也进行异步处理，对于去的请求我们可以利用我们rpc的异步支持进行异步请求所以基本可以达到下图:

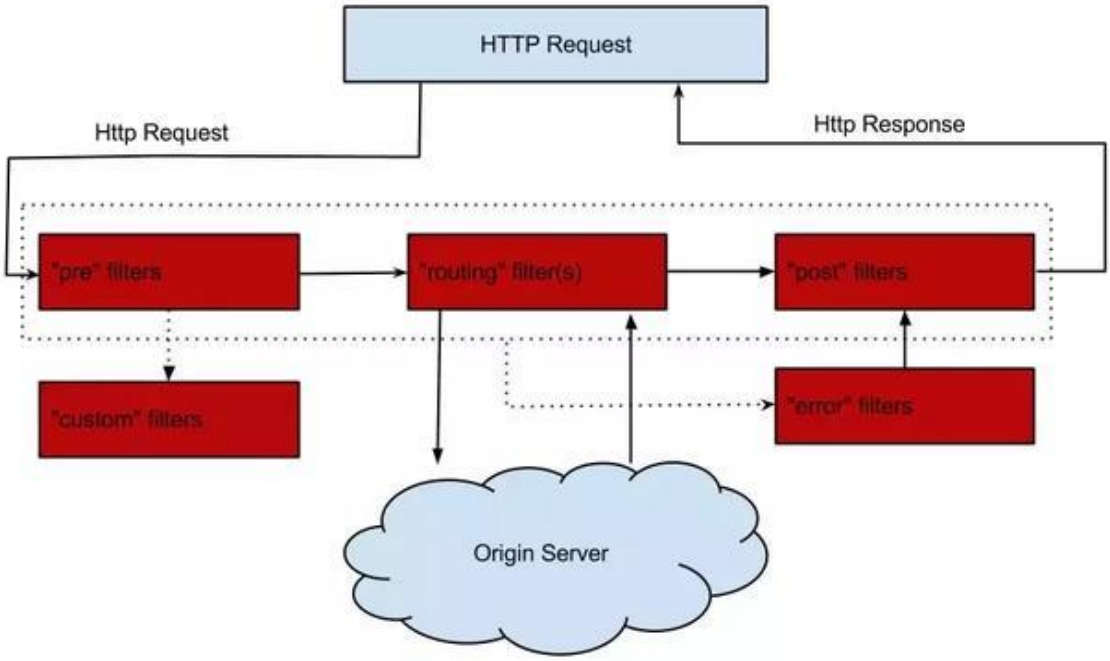


由在web容器中开启servlet异步，然后进入到网关的业务线程池中进行业务处理，然后进行rpc的异步调用并注册需要回调的业务，最后在回调线程池中进行回调处理。

2.2 链式处理

在设计模式中有一个模式叫责任链模式，他的作用是避免请求发送者与接收者耦合在一起，让多个对象都有可能接收请求，将这些对象连接成一条链，并且沿着这条链传递请求，直到有对象处理它为止。通过这种模式将请求的发送者和请求的处理者解耦了。在我们的各个框架中对此模式都有实现，比如servlet里面的filter，springmvc里面的Interceptor。

在Netflix Zuul中也应用了这种模式，如下图所示:



这种模式在网关的设计中我们可以借鉴到自己的网关设计:

preFilters：前置过滤器，用来处理一些公共的业务，比如统一鉴权，统一限流，熔断降级，缓存处理等，并且提供业务方扩展。

routingFilters: 用来处理一些泛化调用，主要是做协议的转换，请求的路由工作。

postFilters: 后置过滤器，主要用来做结果的处理，日志打点，记录时间等等。

errorFilters: 错误过滤器，用来处理调用异常的情况。

这种设计在有赞的网关也有应用。

2.3 业务隔离

上面在全链路异步的情况下不同业务之间的影响很小，但是如果在提供的自定义Fillter中进行了某些同步调用，一旦超时频繁那么就会对其他业务产生影响。所以我们需要采用隔离之术，降低业务之间的互相影响。

2.3.1 信号量隔离

信号量隔离只是限制了总的并发数，服务还是主线程进行同步调用。这个隔离如果远程调用超时依然会影响主线程，从而会影响其他业务。因此，如果只是想限制某个服务的总并发调用量或者调用的服务不涉及远程调用的话，可以使用轻量级的信号量来实现。有赞的网关由于没有自定义filter所以选取的是信号量隔离。

2.3.2 线程池隔离

最简单的就是不同业务之间通过不同的线程池进行隔离，就算业务接口出现了问题由于线程池已经进行了隔离那么也不会影响其他业务。在京东的网关实现之中就是采用的线程池隔离，比较重要的业务比如商品或者订单 都是单独的通过线程池去处理。但是由于是统一网关平台，如果业务线众多，大家都觉得自己的业务比较重要需要单独的线程池隔离，如果使用的是Java语言开发的话那么，在Java中线程

是比较重的资源比较受限，如果需要隔离的线程池过多不是很适用。如果使用一些其他语言比如Golang进行开发网关的话，线程是比较轻的资源，所以比较适合使用线程池隔离。

2.3.3 集群隔离

如果有某些业务就需要使用隔离但是统一网关又没有线程池隔离那么应该怎么办呢？那么可以使用集群隔离，如果你的某些业务真的很重要那么可以为这一系列业务单独申请一个集群或者多个集群，通过机器之间进行隔离。

2.4 请求限流

流量控制可以采用很多开源的实现，比如阿里最近开源的Sentinel和比较成熟的Hystrix。

一般限流分为集群限流和单机限流:

利用统一存储保存当前流量的情况，一般可以采用Redis，这个一般会有一些性能损耗。

单机限流:限流每台机器我们可以直接利用Guava的令牌桶去做，由于没有远程调用性能消耗较小。

2.5 熔断降级

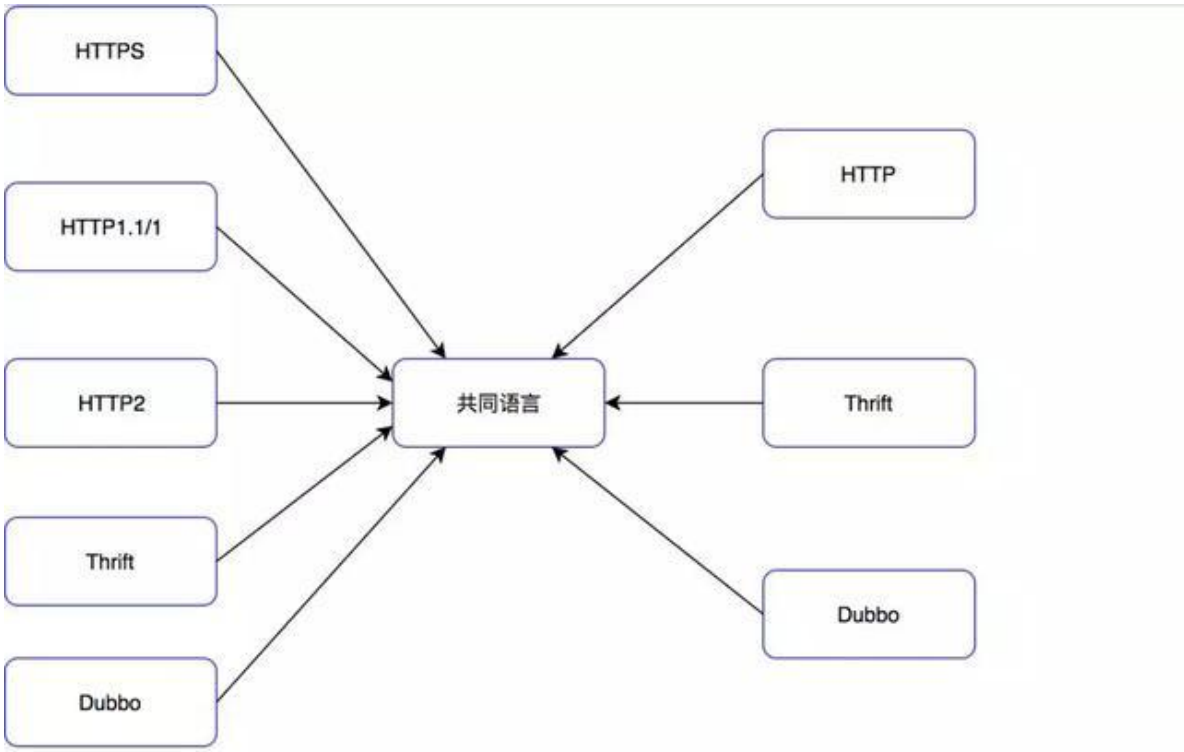
这一块也可以参照开源的实现Sentinel和Hystrix，这里不是重点就不多提了。

2.6 泛化调用

泛化调用指的是一些通信协议的转换，比如将HTTP转换成Thrift。在一些开源的网关中比如Zuul是没有实现的，因为各个公司的内部服务通信协议都不同。比如在唯品会中支持HTTP1,HTTP2,以及二进制的协议，然后转化成内部的协议，淘宝的支持HTTPS,HTTP1,HTTP2这些协议都可以转换成，HTTP,HSF,Dubbo等协议。

2.6.1泛化调用

如何去实现泛化调用呢？由于协议很难自动转换，那么其实每个协议对应的接口需要提供一种映射。简单来说就是把两个协议都能转换成共同语言，从而互相转换。



一般来说共同语言有三种方式指定:

json：json数据格式比较简单,解析速度快，较轻量级。在Dubbo的生态中有一个HTTP转Dubbo的项目是用JsonRpc做的，将HTTP转化成JsonRpc再转化成Dubbo。

比如可以将一个 www.baidu.com/id = 1 GET 可以映射为json：

代码块

```
{
    "method": "getBaidu"
    "param" : {
        "id" : 1
    }
}
```

xml:xml数据比较重，解析比较困难，这里不过多讨论。

自定义描述语言:一般来说这个成本比较高需要自己定义语言来进行描述并进行解析，但是其扩展性，自定义个性化性都是最高。例:spring自定义了一套自己的SPEL表达式语言

对于泛化调用如果要自己设计的话JSON基本可以满足，如果对于个性化的需要特别多的话倒是可以自己定义一套语言。

2.7 管理平台

上面介绍的都是如何实现一个网关的技术关键。这里需要介绍网关的一个业务关键。有了网关之后，需要一个管理平台如何去对我们上面所描述的技术关键进行配置,包括但不限于下面这些配置:

限流

熔断

缓存

日志

自定义filter

泛化调用

3.总结

最后一个合理的标准网关应该按照如下去实现: