TinyNginx

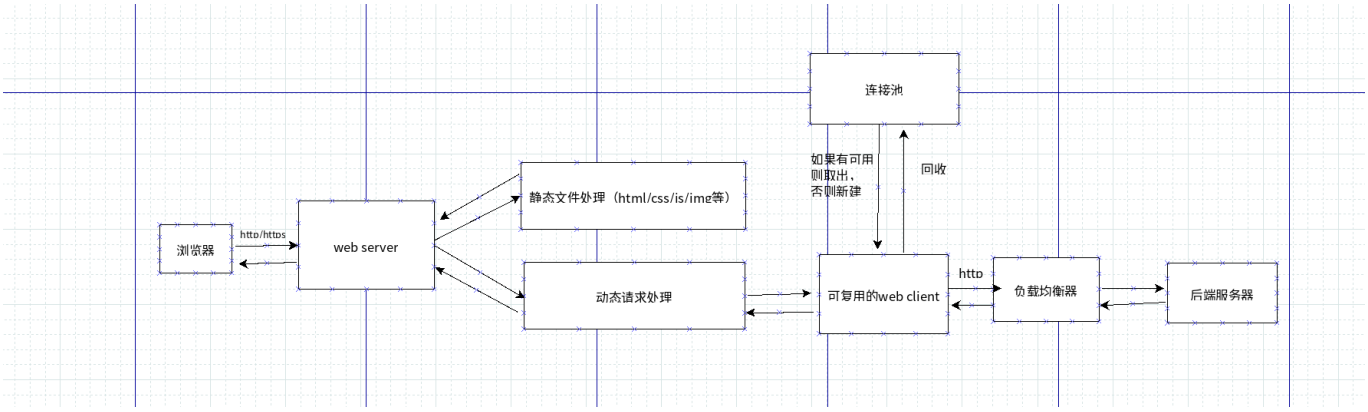
1. 概览

此项目为基于netty的L7负载均衡器(http1.1), jdk版本为11

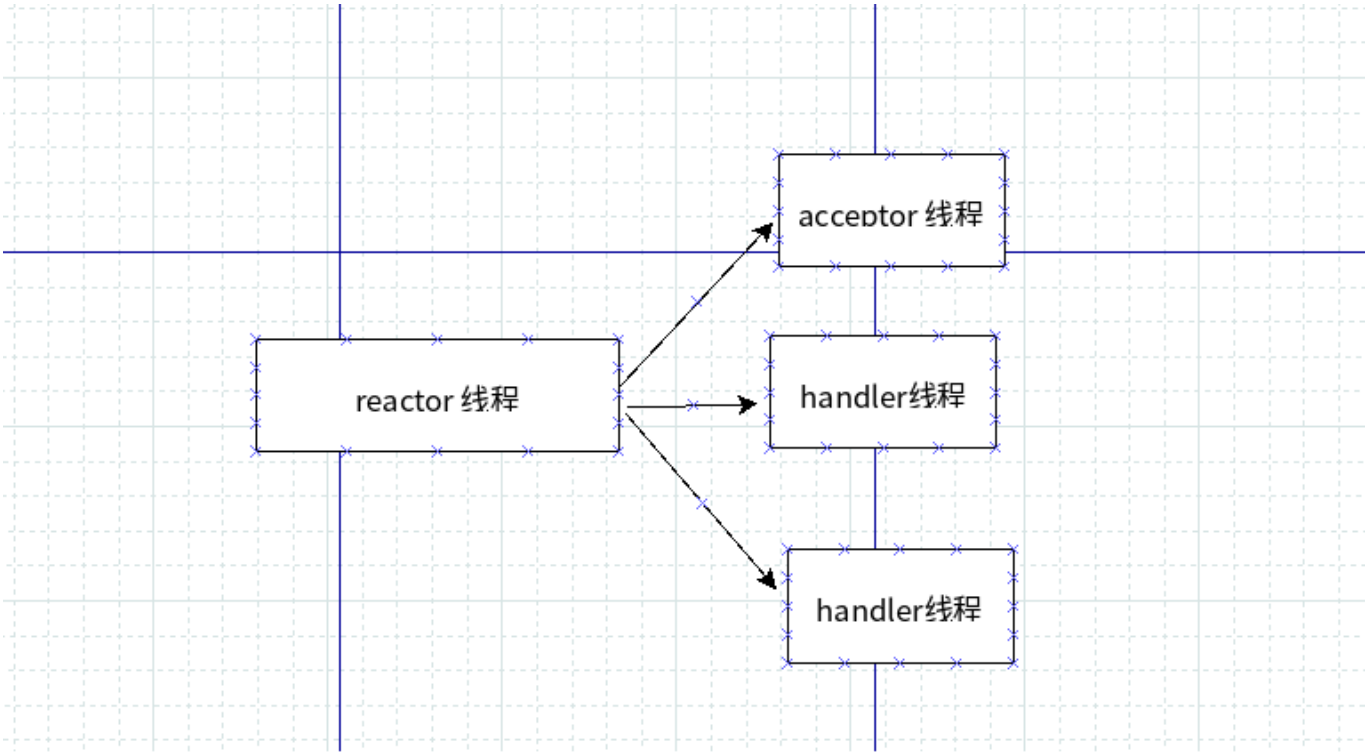
pom如下：

```
        <dependency>
            <groupId>io.netty</groupId>
            <artifactId>netty-all</artifactId>
            <version>4.1.32.Final</version>
        </dependency>
        <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-1.2-api</artifactId>
            <version>2.11.1</version>
        </dependency>
        <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-slf4j-impl</artifactId>
            <version>2.11.0</version>
        </dependency>
```

首先先放一下线程模型



其中 webserver和 webclient 都采用的reactor线程模型



2. 使用说明

1. 下载

```
git clone https://github.com/hg460713171/TinyNginx.git
```

2. run

main 函数为 TinyNginxBoot.main();

3. 配置前端

resources/dist 里面是一个前端项目 可以替换成你自己的前端。

resources/dist 里的是我从 https://github.com/lin-xin/vue-manage-system.git 下载的。

config.properties的 frontend.url 修改为你自己的前端目录。

3. 启动一个或多个springmvc后端 可以使用自己的后端服务器，也可以使用配套提供的简易后端

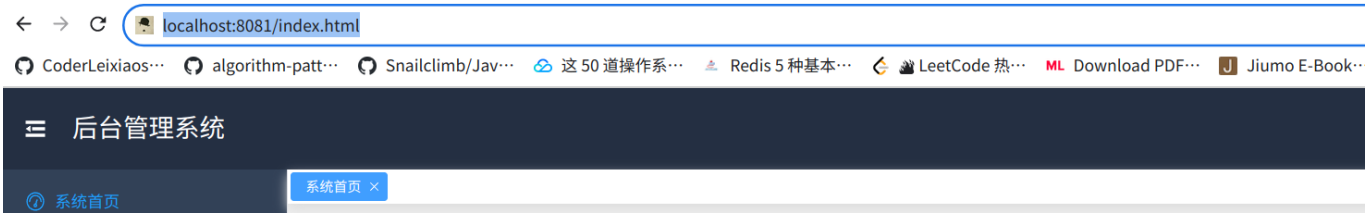
```
git clone https://github.com/hg460713171/TinyNginxBackend.git
```

4.参数配置 参数在 config.properties 里面

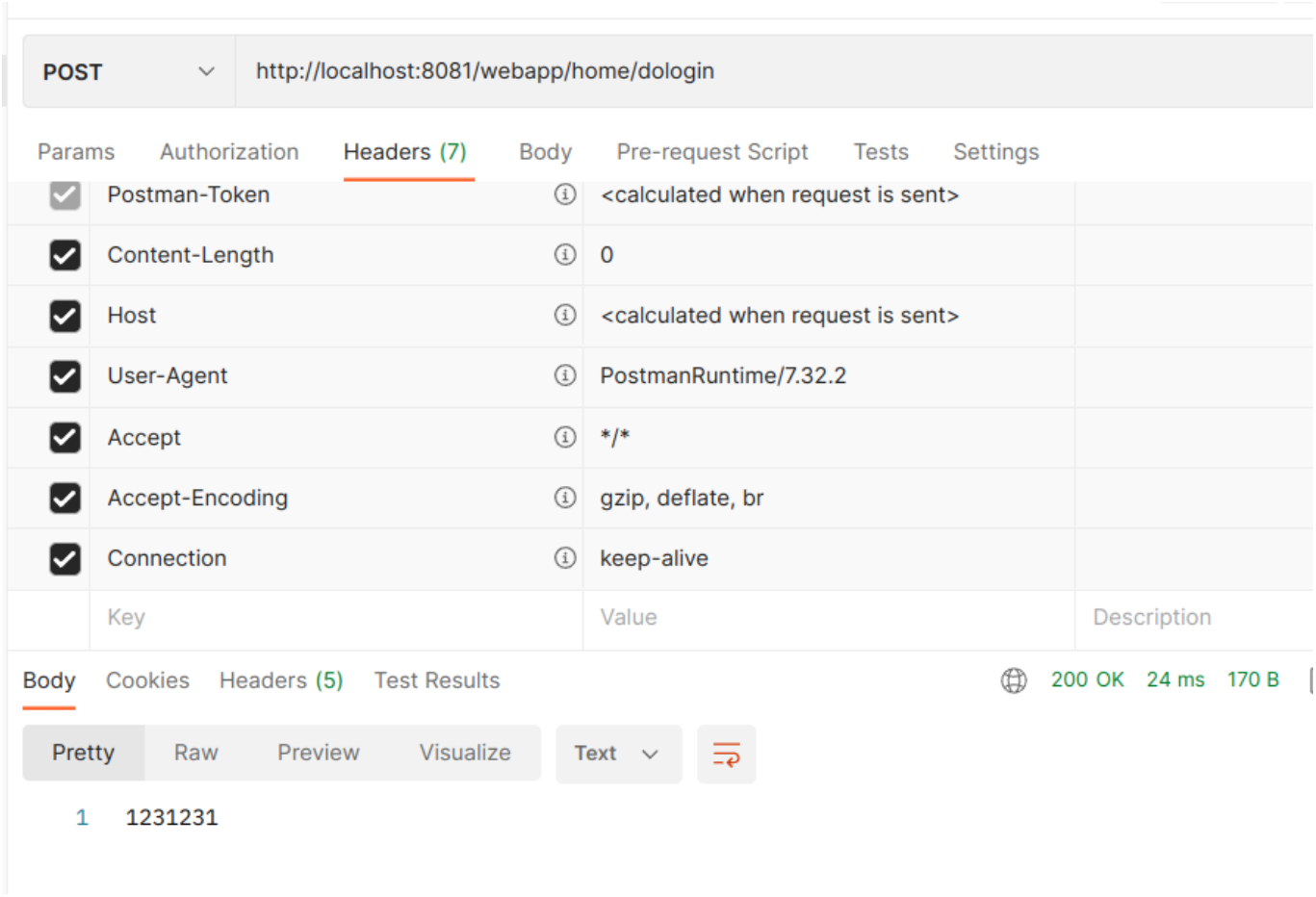
```
port=8081
# tinynginx启动的端口号

backend.prefix = webapp
# 以/webapp为前缀的将会进行反向代理，否则访问本地资源

backend.ip = 192.168.31.141:8081,192.168.31.141:8082
# 后端ip 改为自己的后端 ip

backend.value = 100,100
# 权重

backend.loadbalancer =
com.h.system.tinynignx.loadbalance.RoundRobinByWeightLoadBalance
## 负载均衡的类 默认为roundrobin

frontend.url = /home/hou/IdeaProjects/TinyNginx/src/main/resources/dist
## 前端目录
```

5. 测试

前端：浏览器输入 http://localhost:8081/index.html



后端：在postman上 发送post请求 http://localhost:8081/webapp/home/dologin



3. 负载均衡的介绍

负载均衡器在实际的生产中的有非常多的应用， 比较常用的L4的均衡器有 F5等, L7的均衡器有nginx，haproxy

除此之外，还有很多隐蔽的角落用到了负载均衡，比如rpc的负载均衡，异地多活场景下的全局机房负载均衡，数据库读写分离proxy等。

未来在servicemesh化的过程中 ingress，sidecar等的出现 使得负载均衡器将更加重要。

4. server的实现

基于netty 实现server，相当于此项目的主线程 netty接收前端的请求，根据不同的类型分发到不同的handler 主要有两种handler

一种用来处理反向代理请求 详情见HttpChannelHandler

另一种用来处理静态文件请求 详情见FileChannelHandler

详情见 NettyServer 类

5. client+连接池的实现

5.1. 为什么要实现连接池

由于TCP连接是个重量级的开销，所以现行的http1.1都是以长连接的形式存在，也就是keepalive==true 由于由于http1.1 的channel 这个连接必须等到一个http接收后，才能再进行复用。

这就决定了入池和出池的规则：

```
入池： 当请求完毕时 也就是在channelRead的最后


出池： 1、请求来临时 2、连接被关闭
```

5.2. 连接池处理过程

5.2.1. 新连接来临

新请求来临时 如果采用池中的连接是很快的 但是如果此时池子中没有连接，则需要新建连接，此时netty的优势就体现出来了 核心代码见sendRequestUseNewChannel方法

伪代码如下：

```
server.channelRead {
    future = client.connect();
    future.onComplete{
         wirteAndFlush()
    }
}
```

server并不需要等待这个client的连接成功，而是可以继续处理下一个请求

5.2.2. 老连接

如果发现存在空闲连接 直接从map中取就好

```
注：同ip+port下，http2是可以复用的，

http3 改为udp了 所以任何连接都可以使用同一个channel.

大致的原因是 http2 在http层面加入了某种标识，所以无论发送方顺序如何，接收方总是能找到对应的http响应
（但tcp包不能乱，所以无法实现不同ip+port的channel的复用）

http3 在不仅在应用层还在传输层加入了某种标识，所有可以整个系统共用一个socket内存区

经典八股 就不再赘述了
```

6. 负载均衡算法的实现

RoundRobinByWeightLoadBalance 实现平滑权重负载均衡 最常见的负载均衡算法实现。

核心思路是上一次用过的，尽量就不用了。

负载均衡算法网上比较多，就略了

6.1. 扩展负载均衡算法

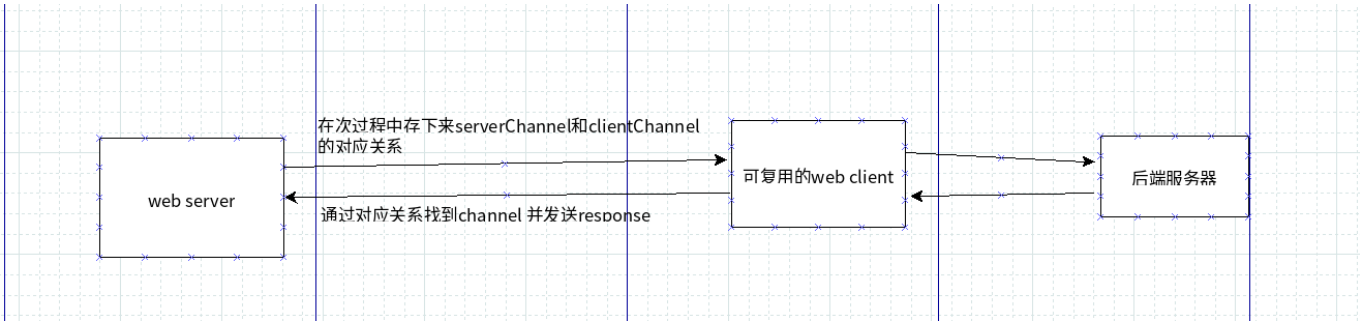
只需要实现 AbstractLoadBalancer ,可以根据request 里的header或 body或者ip 等实现自己的负载均衡策略

比如：异地多活下，需要对不同地域的人进行负载均衡，南方人访问上海服务器，北方人访问北京服务器，这时候可以利用ip的地域性特征 进行负载均衡

再比如：金丝雀发布下，body体里面 userid 尾数为0的访问 A服务器，尾数为1-9的访问B服务器

```
    public abstract BaseRouter getRouter(FullHttpRequest request, String
host);
```

7. 为什么这个模型可以支持高并发



一次完整的请求需要经历以上4个阶段，这些阶段全部是基于事件模型进行处理， 简单理解：事件不发生，服务器可以去干其他的事情

8. 动态负载均衡（未实现）

8.1. 整合Prometheus----实现根据负载动态上下线

这个实现起来比较简单 通过Grafana 的url 定时拉取

根据规则匹配 实现动态上线下线

9. 限流的实现（未实现）

9.1. 基于tps限流

思路 tps限流比较简单只需要在入口处 用滑窗或令牌桶统计即可，

9.2. 基于并发数的限流

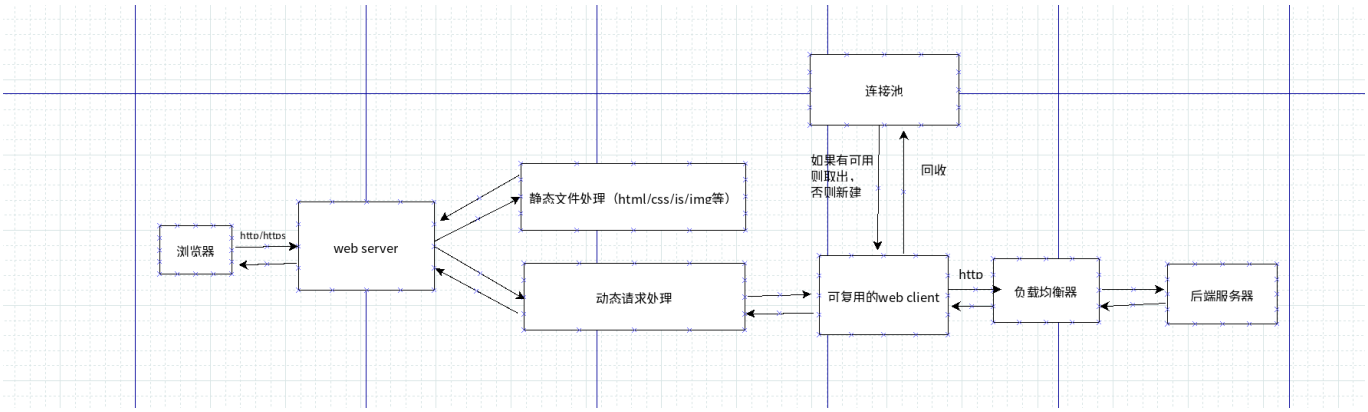
并发数的限流 不仅需要在入口处进行+1，还需要在出口进行-1

10. 压测 （未测试）

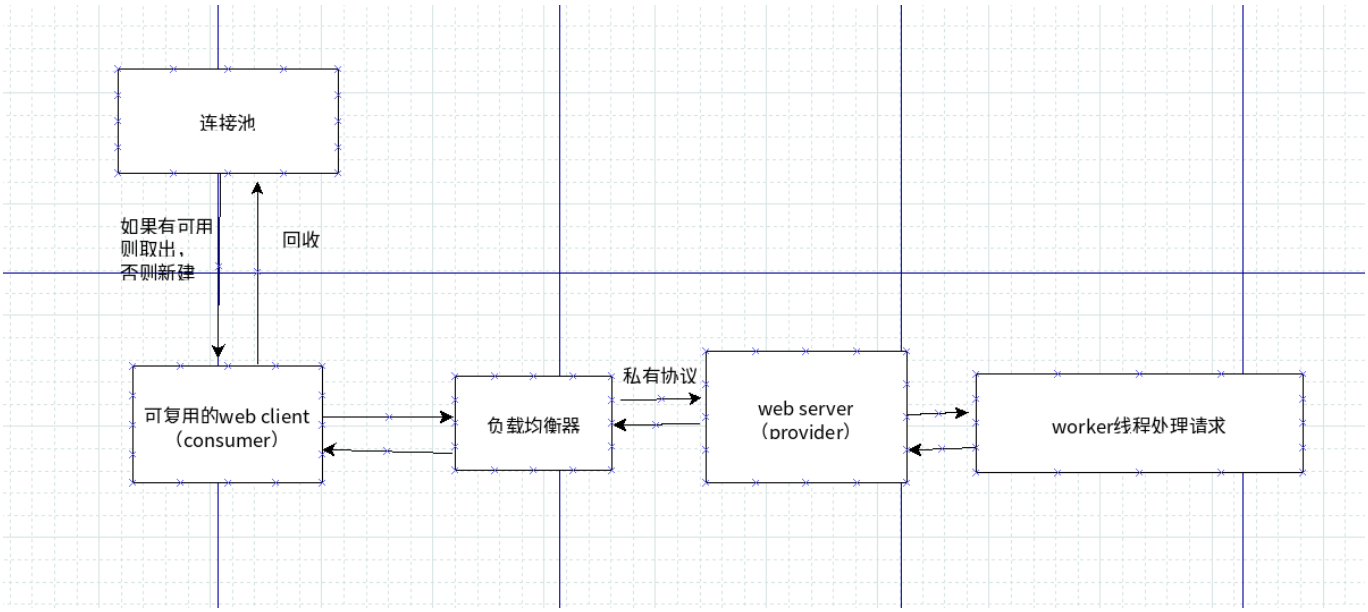
jmeter

11. nginx与tomcat与rpc

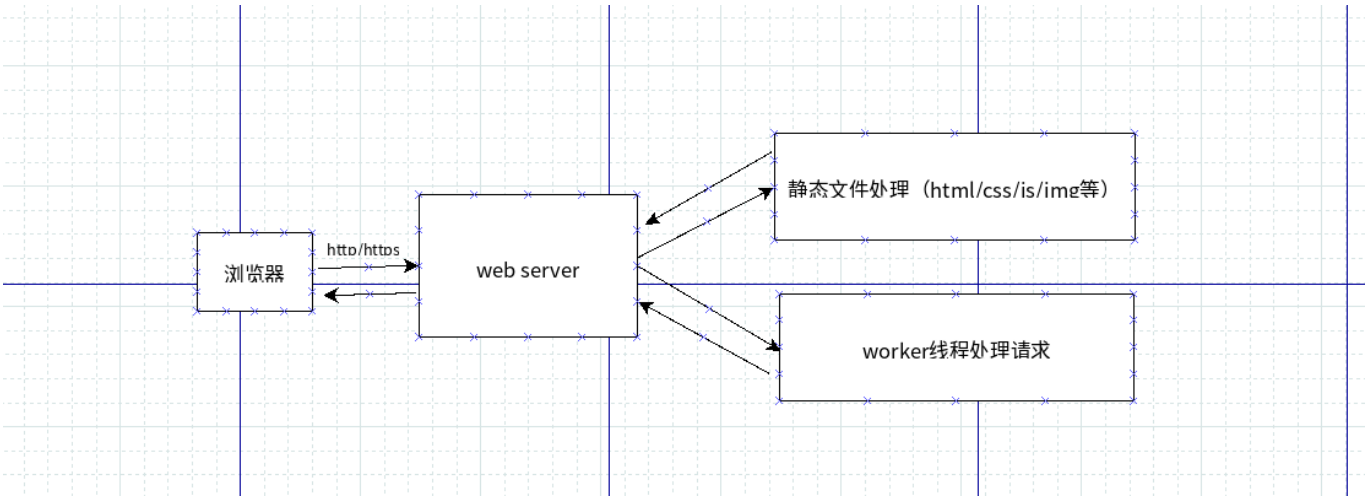
因为nginx是c写的 对java程序员来说，只能看个大概思路，细节上很难在短时间内把握，所以我参考了tomcat和sofa 和dubbo的一些实现 我发现这几个存在很多共通之处，下面放一下他们之间架构图的对比 负载均衡器



RPC



tomcat



12. Http2 协议的实现(未实现)