Spring Cloud 框架
负载测试报告

小泰测试二组

1. 测试概述

1.1 项目背景

目前公司部分项目使用的分布式框架是阿里巴巴开源的 DUBBO，基于各方面考虑，技术部计划在新项目中采用 Spring Cloud 分布式框架。为了各开发团队更好的了解 Spring Cloud 分布式框架默认参数和各参数调整的性能情况，性能测试团队需要对该框架进行全面的基准测试。

1.2 测试目的

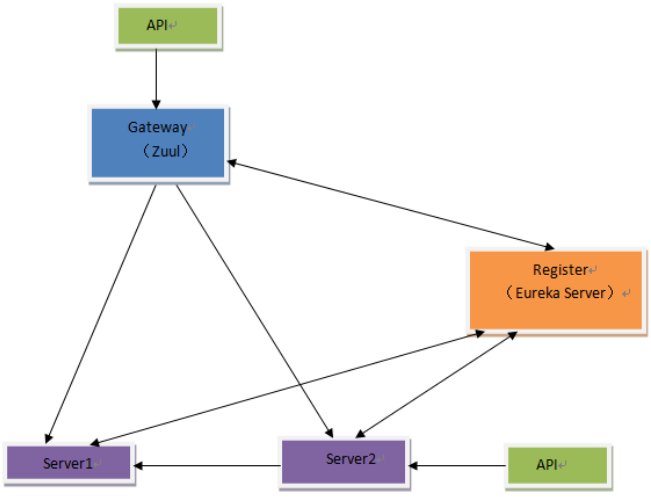
测试基准框架的负载情况，测试修改 Zuul、Eureka、Tomcat、Ribbon 等下面不同配置参数后系统负载情况，和测试该框架的稳定性。

1.3 测试内容

1.3.1 主要检测指标

1. 压测过程中事务成功率
2. 压测过程中服务器的 CPU、Memory、Load 使用情况
3. 压测过程中在一定并发用户数下的响应时间和 TPS

1.3.2 系统架构



Server1 和 Server2 周期性的向 Eureka 发送心跳来更新自己的服务租约，Server 向 Eureka 查询当前注册的服务列表来更新本地的服务列表缓存。API 先到网关 Gateway，网关根据配置和集群方法把请求发到 Server1 或者 Server2 上去，Server2 会再次调用

Server1。也可以请求直接不经过 Zuul，API 直接从 Server2 到 Server1。本框架是最基础的框架，不涉及到 DB 和 Config。

2. 测试环境与配置

- 测试负载机

  CPU：2 核 8G、win7 系统

- 服务器和配置表属性

  Gateway：4 核 4G　　Linux 2.6.32-642.el6.x86_64
  Server1：　2 核 4G　　Linux 2.6.32-696.el6.x86_64
  Server2：　4 核 4G　　Linux 2.6.32-642.el6.x86_64
  Register：4 核 4G　　Linux 2.6.32-642.el6.x86_64
  补充：四个服务都是直接通过 jar 包启动的

Gateway 配置表 bootstrap.yml	**default 值**
server.tomcat.accept-count	100
server.tomcat.max-threads	200
server.tomcat.max-connections	1000
hystrix.command.default.execution.isolation.thread.timeoutInMilliseconds	2000
ribbon.MaxConnectionsPerHost	50
ribbon.MaxTotalConnections	200
ribbon.PoolMaxThreads	200
ribbon.ConnectTimeout	1
ribbon.ConnectTimeout	2000
ribbon.ReadTimeout	5000
zuul.host.maxTotalConnections	200
zuul.host.maxPerRouteConnections	20
zuul.semaphore.maxSemaphores	100
Server2 配置表 application.properties	**default 值**
ribbon.ReadTimeout	5000
ribbon.MaxConnectionsPerHost	20
hystrix.command.default.execution.isolation.thread.timeoutInMilliseconds	5000
hystrix.threadpool.default.coreSize	10
server.tomcat.accept-count	100
server.tomcat.max-threads	200
server.tomcat.max-connections	1000
Server1 配置表 application.properties	**default 值**
server.tomcat.accept-count	100
server.tomcat.max-threads	200
server.tomcat.max-connections	1000

2.1 测试对象和方案

2.1.1 测试对象接口：

API 链路分别有 Zuul->Service1、Zuul->Service2->Service1、Service2->Service1，相对应的接口如下表格所示：

Zuul->Server1	接口 URL
POST 无延时的 API	http://10.200.131.72:8000/performance-service/performance
GET 无延时的 API	http://10.200.131.72:8000/performance-service/performance
POST 延时时间为 200ms 的 API	http://10.200.131.72:8000/performance-service/performance/delay
GET 延时时间为 200ms 的 API	http://10.200.131.72:8000/performance-service/performance/delay/200
POST 延时时间为 600ms 的 API	http://10.200.131.72:8000/performance-service/performance/delay
GET 延时时间为 600ms 的 API	http://10.200.131.72:8000/performance-service/performance/delay/600
GET 混合接口	http://10.200.131.72:8000/performance-api/performance/mix
GET 延时时间为 1s 的 API	http://10.200.131.72:8000/performance-service/performance/file
Zuul->Server2->Server1:	接口 URL
GET 延时时间为 200ms 的 API	http://10.200.131.72:8000/performance-api/performance/delay/200
GET 延时时间为 600ms 的 API	http://10.200.131.72:8000/performance-api/performance/delay/600
GET 延时时间为 1s 的 API	http://10.200.131.72:8000/performance-api/performance/file
GET 混合接口	http://10.200.131.72:8000/performance-api/performance/mix
Server2->Server1	接口 URL
GET 延时时间为 200ms 的 API	http://10.200.131.75:8813/performance/delay/200
GET 延时时间为 600ms 的 API	http://10.200.131.75:8813/performance/delay/600
GET 延时时间为 1s 的 API	http://10.200.131.75:8813/performance/file
GET 混合接口	http://10.200.131.75:8813/performance/mix

2.1.2 测试方案

基准测试(默认配置)：

Zuul->Service1:

1. 对比相同并发数下，延时时间分别都为 200ms、600ms 的 POST/GET 2 种接口，对比相应的响应时间、TPS 和服务器压力，测试 POST/GET 两种请求方式有无区别；
2. 测试延时时间分别为 200ms、600ms、1s 和混合型接口，不报错情况下的平均响应时间在 2s 内的最大并发数和 TPS，并查看服务器的 CPU 情况；

3. 测试默认配置下的临界值；

Zuul->Service2->Service：

1. 测试延时时间分别为 200ms、600ms、1s 和混合型接口，不报错情况下的平均响应时间在 2s 内的最大并发数和 TPS，并查看服务器的 CPU 情况；
2. 测试默认配置下的临界值；

调整参数：

Zuul->Service1、Zuul->Service2->Service1、Service2->Service1：

1. 调整不同参数，测试在延时时间分别为 200ms、600ms、1s 和混合型接口的，不报错情况下且响应时间在 2s 内的最大并发数和 TPS，并查看服务器的 CPU 情况，并找出其规律；
2. 测试在延时时间分别为 200ms、600ms、1s 和混合型接口，相同配置下，经过 Server2 跳转和不跳转 2 种方式区别；
3. 测试在延时时间分别为 200ms、600ms、1s 和混合型接口，相同配置下，测试访问链路为 Zuul->Service1 和 Service2->Service1 两种方式的区别；
4. 调整参数，测试访问链路为 Zuul->Service1 和 Zuul->Service2->Service1 长时间压测，测试系统的稳定性；

补充：会根据测试结果及时调整一些参数的值，方便对比测试结果。

2.2 测试工具

LoadRunner11 压力测试工具

3. 测试场景和执行结果

3.1 测试场景

测试场景（一）描述

查看默认配置下请求链路为 Zuul->Service1，事物成功率 100%且平均响应时间在 2s 内的最大并发数和 TPS，并查看服务器的 CPU 情况。

测试场景（一）结果

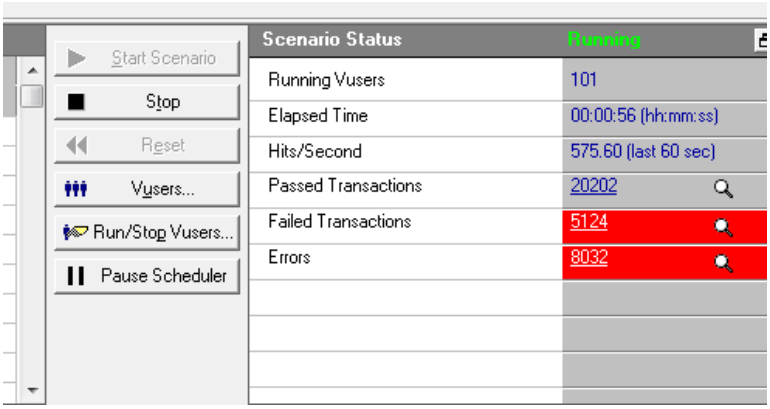
1. 并发用户数为 100

接口类型	压测时间	并发用户数	响应时间（s）	TPS
200ms	10min	100	0.406	243
600ms	10min	100	1.20	82
1000ms	10min	100	1.98	50
mix	10min	100	0.637	154

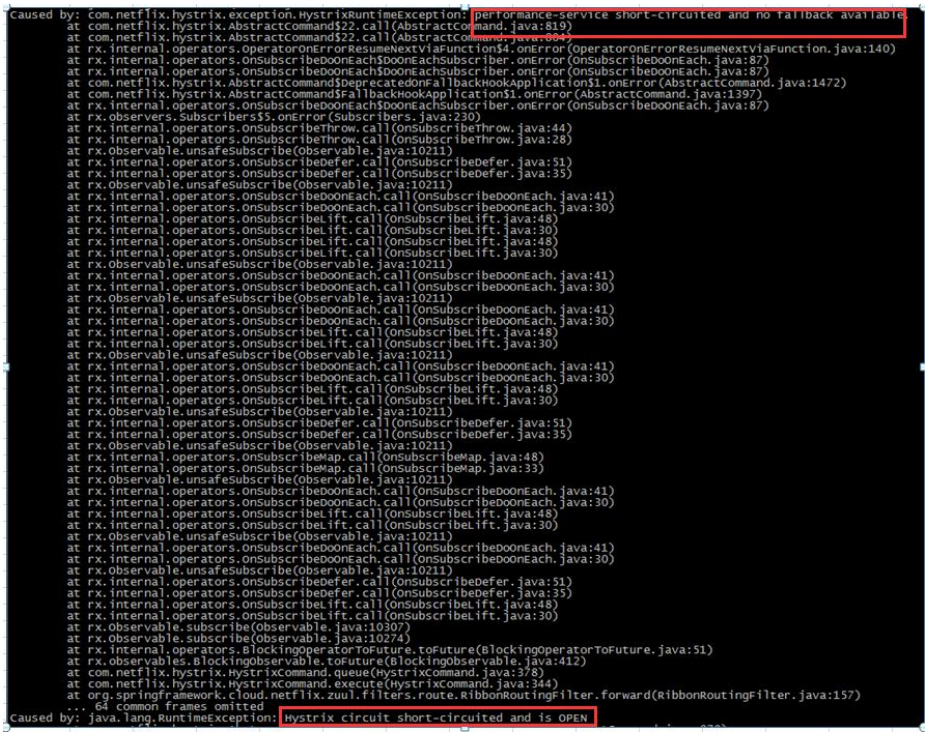
补充：以上都是 GET 接口，且 Gateway 和 Server1 的 CPU 都在 5%左右。

2. 并发用户数为 101

所有接口都报 500 错误，Load Runner 截图：



服务器 Gateway 日志截图：



补充：并发用户数为 101 时 Zuul 的 Hystrix 熔断开启，且 Gateway 的 CPU 暴增至 90%,Load 值也偏大。

测试场景（二）描述

在相同服务器配置、接口延时时间相同且 API 请求链路都为 Zuul->Service1 的情况下，API 请求方式分别为 GET/POST 两个接口，测试两种请求方式有无区别。

测试场景（二）结果

在相同服务器配置、接口延时时间相同且请求链路都一致的情况下，POST 和 GET 请求方法结果几乎相同，故后面测试只需要测试一种方式即可。测试结果数据如下：

接口类型	压测时间	并发用户数	响应时间（s）	TPS
200ms/GET	10min	100	0.406	243
200ms/POST	10min	100	0.413	245
600ms/GET	10min	100	1.20	82
600ms/POST	10min	100	1.19	81

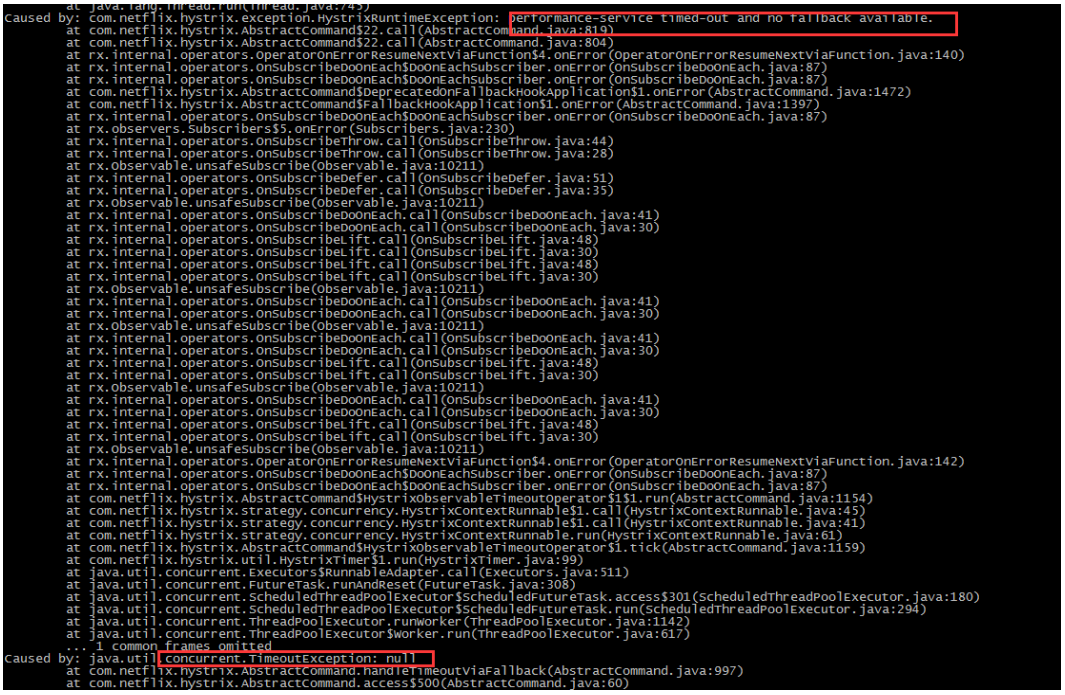
补充：1.上面场景里服务器的 CPU、Load、Memory 都在正常范围内。

测试场景（三）描述

调节网关相关配置，API 请求链路为 Zuul->Service1，测试各个网关配置对系统支持的最大并发数和 TPS 的影响（平均响应时间在 2s 内）。

测试场景（三）结果

1. hystrix.command.default.execution.isolation.thread.timeoutInMilliseconds 默认为 5000，设置为 8000 和 10000 的时候，测试 API 为延时时间为 600ms 的 GET 接口，并发用户数为 200 时，会报错，所以设置延时时间为 15s（后面都延时时间设置为 15s）。服务器 Gateway 日志截图：

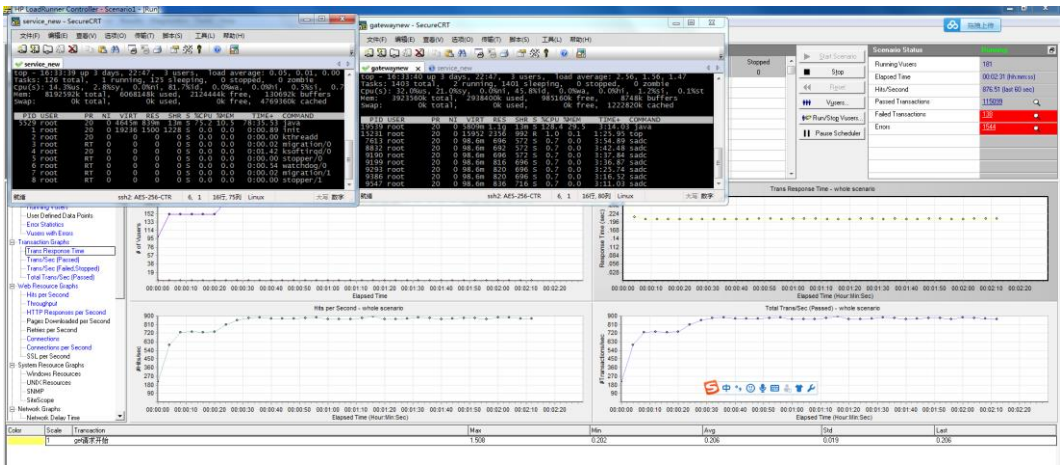


补充：Gateway 报响应时间延时 Server 不可用。

2. zuul.semaphore.maxSemaphores 主要是影响框架的并发用户数，值越大，支持的最大并发数越大。Tomcat 默认配置不变，ribbon. MaxConnectionsPerHost 设为 200（default 为 50），测试 API 都为延时时间为 200ms 的 GET 接口， zuul.semaphore.maxSemaphores 为下面值时结果如下：

maxSemaphores 的值	压测时间	并发用户数	响应时间	TPS	补充
100（default）	10min	100	0.203	492	101 个并发数报错
120	10min	100	0.203	482	101 个并发数报错
150	10min	150	0.205	745	151 发数报错
180	10min	180	0.2	876	181 个并发数报错
200	10min	1000	1.02	989	1010 并发用户数正常

maxSemaphores 为 180，并发用户数为 181 的压测图：

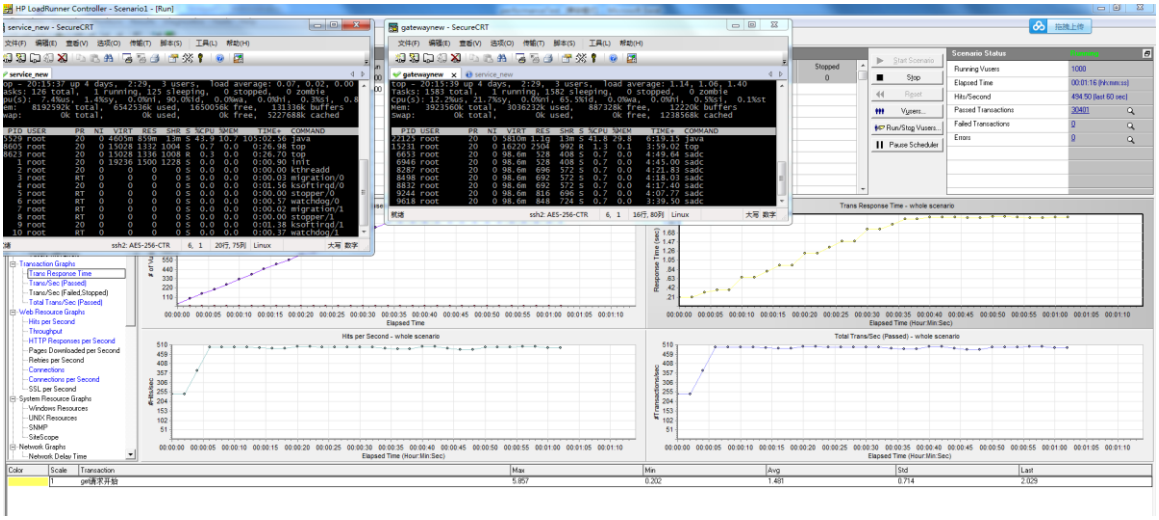


MaxConnectionsPerHost:200_maxSemaphores:180_vu181 报错截图

3. zuul.semaphore.maxSemaphores 主要影响系统的 TPS，值越大，支持的最大 TPS 越大。Tomcat 默认配置不变，zuul.semaphore.maxSemaphores 设为 200（default 为 100），测试 API 都为 200ms 的 GET 接口，ribbon. MaxConnectionsPerHost 为下面值时，测试结果如下表格：

MaxConnectionsPerHost 的值	压测时间	并发用户数	响应时间（s）	TPS
50（default）	10min	1000	1.014	987
100	10min	200	0.402	500
100	10min	500	1.00	489
100	10min	1000	2.02	500
200	10min	1000	1.02	989

MaxConnectionsPerHost:100_maxSemaphores:200 压测图：



MaxConnectionsPerHost:100_maxSemaphores:200_vu1000 截图

补充：1. 上面场景里服务器的 CPU、Load、Memory 都在正常范围内。

测试场景（四）描述

Service2 用默认配置，API 请求链路为 Zuul->Service2->Service1，查看不报错情况下的平均响应时间在 2s 内的最大并发数和 TPS，并查看服务器的 CPU 情况。

测试场景（四）结果

延时时间为 200ms 的 GET 接口，并发用户数为 10 的时候，响应时间为 0.2s 左右，并发用户数为 11 的时候 Server2 的熔断机制开启，TPS 暴增，响应时间减小，单接口访问返回值为 error（server2 的熔断处理机制），截图如下：

测试场景（五）描述

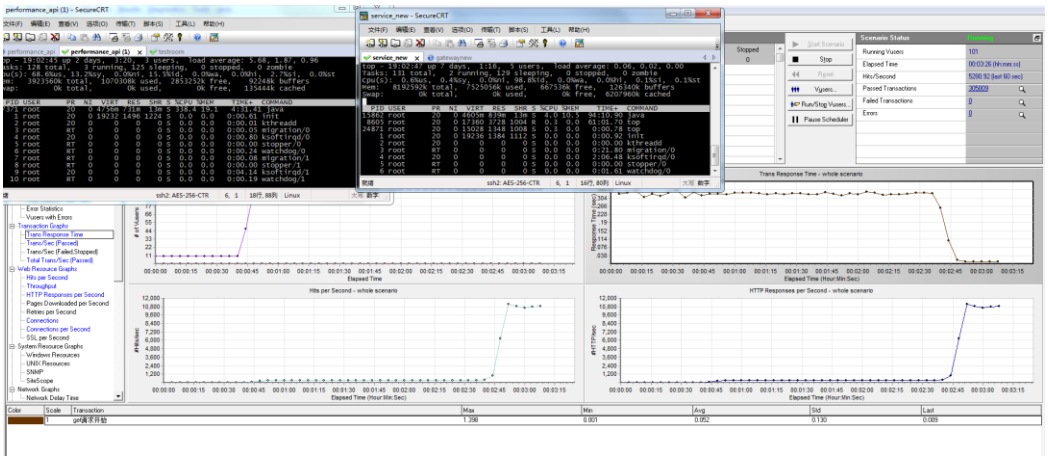
API 请求链路为 Zuul->Service2->Service1，修改 Server2 的 hystrix.threadpool.default.coreSize 值，测试对各接口的影响。

测试场景（五）结果

hystrix.threadpool.default.coreSiz 影响系统支持的最大并发数， 如果并发数达到该设置值，请求会被拒绝并会开启 Hystrix。延时时间为 200ms 的 GET 接口，修改 hystrix.threadpool.default.coreSize 的值，测试不同值下面最大正常运行的并发用户数和开启 Hystrix 的并发用户数。

coreSize 的值	最大正常运行的并发数	开启 Hystrix 的并发用户数
10（default）	10	11
30	30	31
100	100	101
190	190	191
200	1000	NA
500	1000	NA

coreSize 为 100 并发数从 100 加到 101 的压测截图：



coreSize 为 100 并发用户数从 100 加到 101 的截图

测试场景（六）描述

测试 Zuul 为默认配置下，在请求链路分别为 Zuul->Service1、Zuul->Service2->Service1、Service2->Service1，测试相同延时时间且请求方式都是 GET 接口之间的区别。因为 Server2 的默认配置下并发用户数为 10 熔断就开启，所以这里 hystrix.threadpool.default.coreSize 的值设置为 1000。

测试场景（六）结果

默认配置下，相同延时时间的接口在在 API 链路为 Zuul->Service1 和链路为 Zuul->Service2->Service1 两种链路方式下差异很小，结果如下表格：

接口类型	链路方式	压测时间	并发用户数	响应时间（s）	TPS
200ms	Zuul->Service1	10min	100	0.406	243
	Zuul->Service2->Service1	10min	100	0.412	242
	Service2->Service1	10min	100	0.205	500
600ms	Zuul->Service1	10min	100	1.20	82
	Zuul->Service2->Service1	10min	100	1.21	83.75
	Service2->Service1	10min	100	0.602	165
1000ms	Zuul->Service1	10min	100	1.98	50
	Zuul->Service2->Service1	10min	100	2.00	50
	Service2->Service1	10min	100	1.04	100
0.3 的 600ms 0.7 的 200ms	Zuul->Service1	10min	100	0.639	154
	Zuul->Service2->Service1	10min	100	0.649	155
	Service2->Service1	10min	100	0.323	311

补充：1.Service2->Service1 TPS 不一样是因为没有 ribbon 的限制。

2. 面场景里服务器的 CPU、Load、Memory 都在正常范围内。

测试场景（七）描述

在 Zuul 里面 Tomcat 为默认配置，ribbon.MaxConnectionsPerHost 为 200 且

zuul.semaphore.maxSemaphores 为 200 配置情况下，在网络链路分别为 Zuul->Service1、Zuul->Service2->Service1、Service2->Service1，测试相同延时时间且请求方式都是 GET 接口之间的区别。因为 Server2 的默认配置下并发用户数为 10 熔断就开启，所以这里 Server2 的 hystrix.threadpool.default.coreSize 设置为 1000。

测试场景（七）结果

默认配置下，相同延时时间的接口在三种链路情况下的结果几乎相同，结果如下表格：

接口类型	链路方式	压测时间	并发用户数	响应时间（s）	TPS
200ms	Zuul->Service1	10min	1000	1.02	989
	Zuul->Service2->Service1	10min	1000	1.02	983
	Service2->Service1	10min	1000	1.01	984
600ms	Zuul->Service1	10min	1000	1.80	330
	Zuul->Service2->Service1	10min	1000	1.79	334
	Service2->Service1	10min	1000	1.82	332
1000ms	Zuul->Service1	10min	1000	2.00	203
	Zuul->Service2->Service1	10min	1000	2.00	198
	Service2->Service1	10min	1000	2.03	200
0.3 的 600ms 0.7 的 200ms	Zuul->Service1	10min	1000	1.61	621
	Zuul->Service2->Service1	10min	1000	1.63	617
	Service2->Service1	10min	1000	1.63	622

补充：1. 上面场景里服务器的 CPU、Load、Memory 都在正常范围内。

测试场景（八）描述

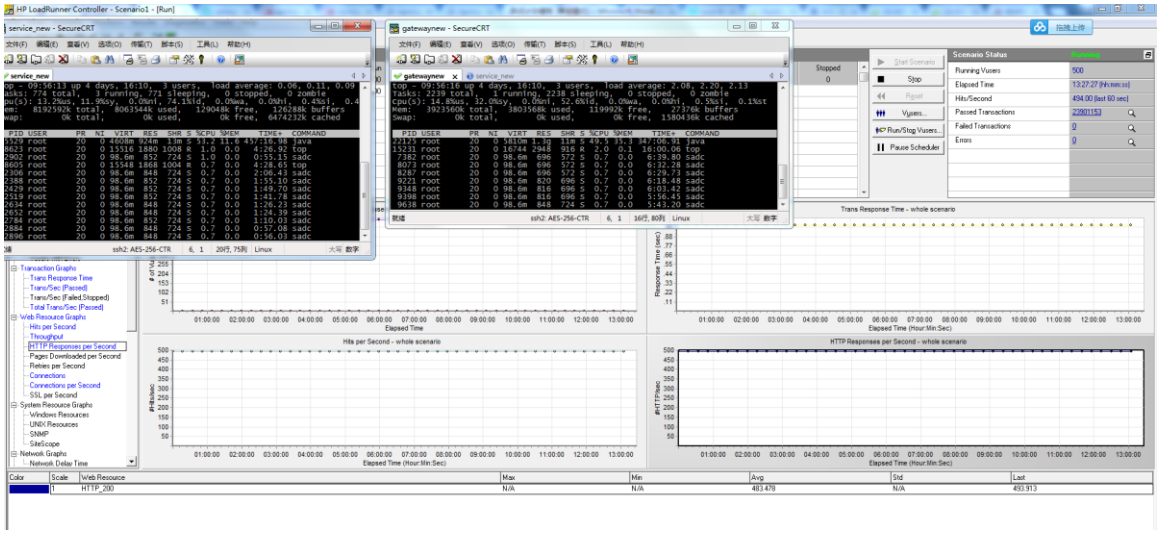
在 Zuul 里面 ribbon.MaxConnectionsPerHost 为 100、zuul.semaphore.maxSemaphores 为 200 且 Server2 的 hystrix.threadpool.default.coreSize 设置为 1000，其他为默认配置情况下，API 请求链路为 Zuul->Service1，长时间压测延时时间为 200 的 GET 接口，查看服务器的稳定性。

Gateway 配置表 bootstrap.yml 配置如下所示：

Gateway 配置表 bootstrap.yml	设置值
server.tomcat.accept-count	100
server.tomcat.max-threads	200
server.tomcat.max-connections	1000
hystrix.command.default.execution.isolation.thread.timeoutInMilliseconds	15000
ribbon.MaxConnectionsPerHost	200
ribbon.MaxTotalConnections	200
ribbon.PoolMaxThreads	200
ribbon.ConnectTimeout	100
ribbon.ConnectTimeout	2000
ribbon.ReadTimeout	15000
zuul.host.maxTotalConnections	200
zuul.host.maxPerRouteConnections	20
zuul.semaphore.maxSemaphores	200

测试场景（八）结果

压测时间为 13:27 分钟，并发用户数为 500，压测中比较稳定，且查看了服务器的 CPU 都比较稳定在 50%以下，截图如下：



补充：1. 上面场景里服务器的 CPU、Load、Memory 都在正常范围内。

2. 长时间压测 TPS 在 500 左右，响应时间为 1s 左右。

测试场景（九）描述

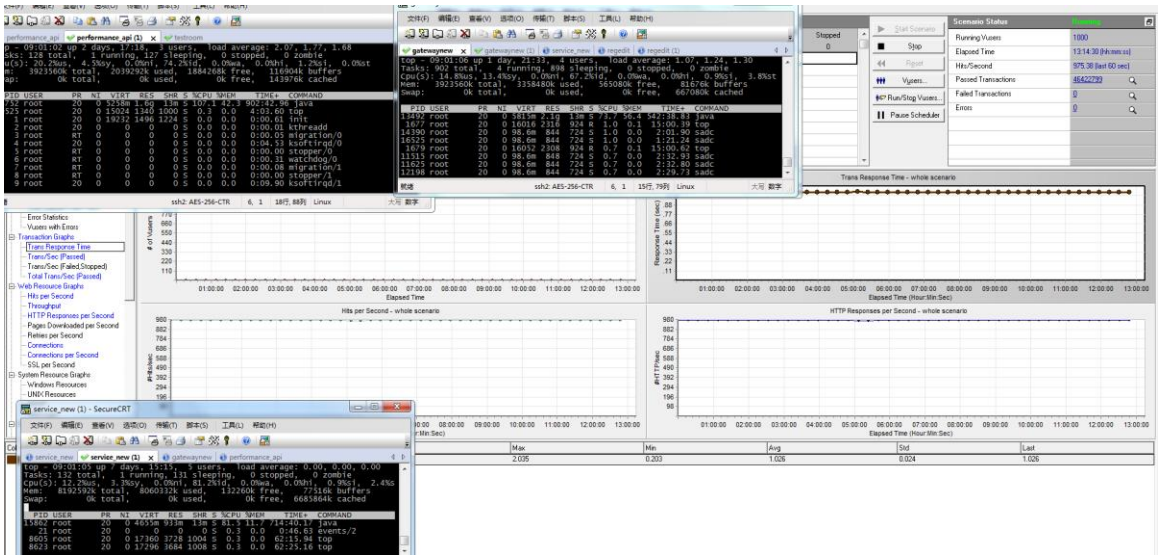
在 Zuul 里面 ribbon.MaxConnectionsPerHost 为 200 且 zuul.semaphore.maxSemaphores 为 200，Tomcat 为默认配置情况下，API 请求链路为 Zuul->Service2->Service1，长时间压测延时时间为 200 的 GET 接口，查看服务器的稳定性。

Gateway 配置表 bootstrap.yml 配置和 Server2 配置表 application.properties 如下所示：

Gateway 配置表 bootstrap.yml	设置值
server.tomcat.accept-count	100
server.tomcat.max-threads	200
server.tomcat.max-connections	1000
hystrix.command.default.execution.isolation.thread.timeoutInMilliseconds	15000
ribbon.MaxConnectionsPerHost	200
ribbon.MaxTotalConnections	200
ribbon.PoolMaxThreads	200
ribbon.ConnectTimeout	100
ribbon.ConnectTimeout	2000
ribbon.ReadTimeout	15000
zuul.host.maxTotalConnections	200
zuul.host.maxPerRouteConnections	20
zuul.semaphore.maxSemaphores	200
Server2 配置表 application.properties	**设置值**
ribbon.ReadTimeout	15000
ribbon.MaxConnectionsPerHost	20
hystrix.command.default.execution.isolation.thread.timeoutInMilliseconds	5000
hystrix.threadpool.default.coreSize	200
server.tomcat.accept-count	100
server.tomcat.max-threads	200
server.tomcat.max-connections	1000

测试场景（九）结果

压测时间为 13:17 分钟，并发用户数为 1000，压测中 TPS 和响应时间比较稳定，且查看了服务器的 CPU 都比较稳定在 50%以下，服务器的 Load 和内存使用都在正常范围内，截图如下：



补充：1. 上面场景里服务器的 CPU、Load、Memory 都在正常范围内。

2. 长时间压测 TPS 在 980 左右，响应时间为 1s 左右。

测试场景（十）描述

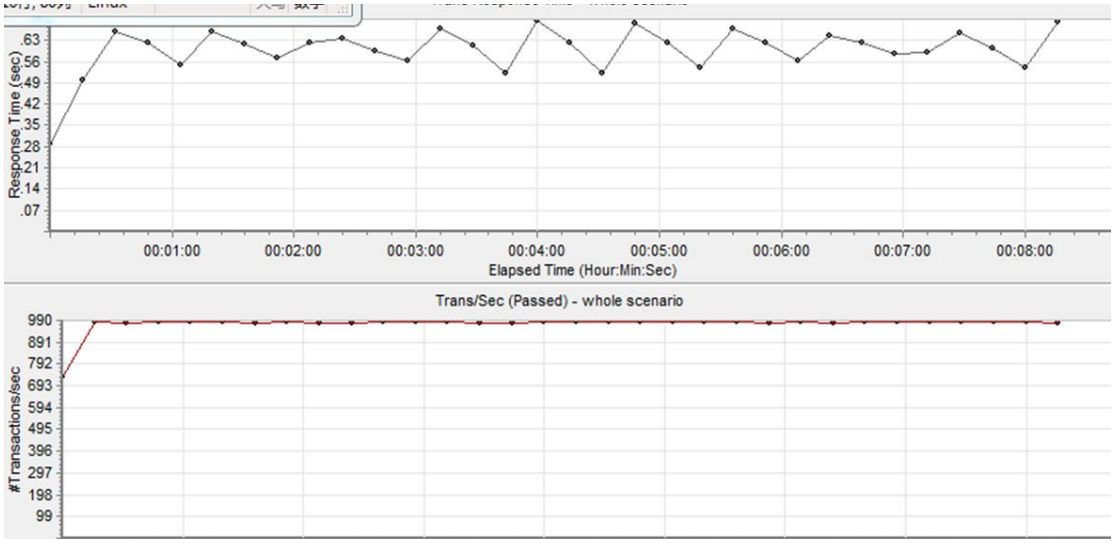
API 请求链路为 Zuul->Service1，调整 Tomcat 的内核 max-connections 或者 max-threads 值，查看对系统影响。

测试场景（十）结果

1. 调整 max-threads 时，相同延时时间的接口访问时，TPS 会相应的改变。下面为延时时间为 200ms 的 GET 接口（ribbon.MaxConnectionsPerHost 为 200，zuul.semaphore.maxSemaphores 为 200，其他为默认配置）

max-threads 的值	压测时间	并发用户数	响应时间（s）	TPS
200（default）	10min	1000	1.014	987
100	10min	1000	2.07	487

2. 调整 max-connections 值，相同延时时间的接口访问时，TPS 会相应的改变。测试配置为 max-connections 为 500（ribbon.MaxConnectionsPerHost 为 200，zuul.semaphore.maxSemaphores 为 200，其他为默认配置），测试延时时间为 200ms 的 GET 接口，并发用户数为 600 的截图：



补充说明：

1. 可看出响应时间有一定波动，主要是因为 accept-count 为 100，支持排队 100 个请求，多余 600 个就会报错
2. max-threads 和 max-connections 配置建议不需要调整，场景（九）只是为了测试 Tomcat 内核几个参数对系统影响，且把这两个值调大，相同接口，相同并发用户数的 TPS 并没有相应的变大

4. 结果分析和结论

1. Zuul 默认配置最大支持并发用户数只有 100，需要调大最大并发用户数请调节

zuul.semaphore.maxSemaphores 值（default：100），超过 100 就会触发熔断机制，调节到 200 时，在一般接口响应时间为 200ms 时，支持最大并发用户数 1000 以上。

2. Zuul 在默认配置下，服务器响应时间为 0.2s 时服务器支持的最大 TPS 为 250 左右。需要调整系统的 TPS 可以调整 ribbon.MaxConnectionsPerHost 的值（default：50），值越大，支持的 TPS 也相应增大，当调整为 200 时，服务器响应时间为 200ms 时最大支持的 TPS 为 1000 左右。当然多个服务连接时，ribbon.MaxConnectionsPerHost 调大，ribbon. MaxTotalConnections 也要相应的调整。
3. Server 跳转到另外一个 Server，需要调整 hystrix.threadpool.default.coreSize 的值，默认只有 10，并发用户数超过 10 个就会触发熔断机制，建议设置为 200，即可支持最大并发用户数 1000。
4. 触发 Hystrix 延时时间默认为 5s，即使设置为 10s，200 个并发响应时间为 600ms 时，部分事件会因为时间导致事务失败，建议 Zuul 的 hystrix.Command.Default.Execution.Isolation.Thread.timeoutInMilliseconds 设置为 15s。
5. 相同配置下，接口响应时间越长，最大支持的 TPS 相应减小，响应时间相应增大。
6. 从测试长时间负载情况看，该框架稳定性不错。

5. 风险分析

1. 目前接口比较简单，没有复杂的逻辑计算，方法不太耗用系统的 CPU 执行时间，所以相同并发用户下，耗得 CPU 执行时间比较短，所以实际中的响应时间会比测试的响应时间更长，实际接口中的 TPS 会比测试结果更小，实际中的服务器的 CPU 比实际中的更大。
2. 目前测试的核数和线上的核数内存不成比例，该测试结果只能提供配置方面参考。
3. Hystrix 熔断响应时间如果设置为 15s，真实线上服务器无响应，会进行多次重试，时间会有点长。且需要做相应的熔断处理，否则会直接报 500 错误。