厦門大學

信息学院软件工程系

《JavaEE 平台技术》实验报告

实验 3 Redis 缓存的效率

组号：2-6

完成时间：11.30

目录

一. 实验目的

1、掌握 Redis 缓存的使用方法
2、比较使用和不使用 Redis 缓存的效果

二. 实验环境

1、服务器 A: Ubuntu 18.04 服务器 2 核 4G 内存虚拟机一台，图形界面，安装 JDK 11，Maven、git，Redis 6.2.4
2、服务器 B：Ubuntu 18.04 服务器 2 核 2G 内存虚拟机一台，命令行界面，安装 JDK 11，Maven、git，JMeter 5.4.1
3、服务器 C：Ubuntu 18.04 服务器 2 核 2G 内存虚拟机一台，命令行界面，安装 JDK 11，Maven、git，MySQL 8.0

三. 实验背景

本次实验是比较使用 redis 和不使用 redis 的效率，此外还需要与实验 2 的结果对比，但是由于我们最终得到的实验结果相差过大，我们组后来直接在原 oomall 的代码基础上，比较了注释掉大部分 redis 和没注释 redis 缓存的读写效率。此外，我们还比较了在原 OOMALL 程序包下有 cloneobj 与无 cloneobj 的效率。

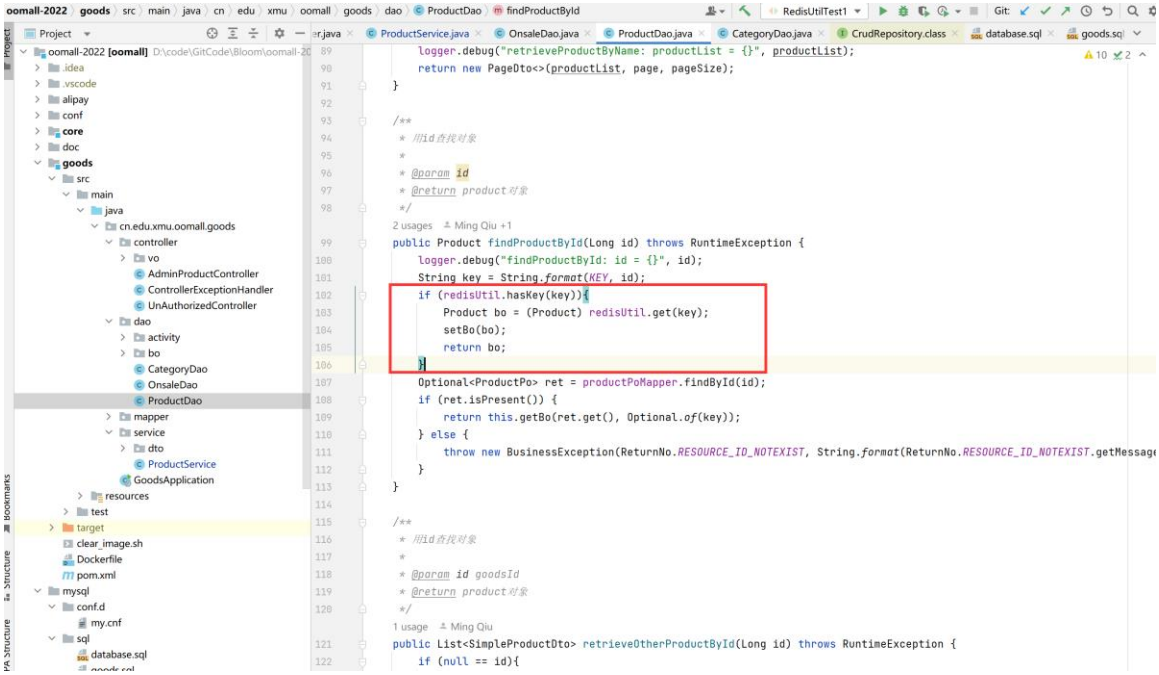


图 1 包含 redis 的 Oomall

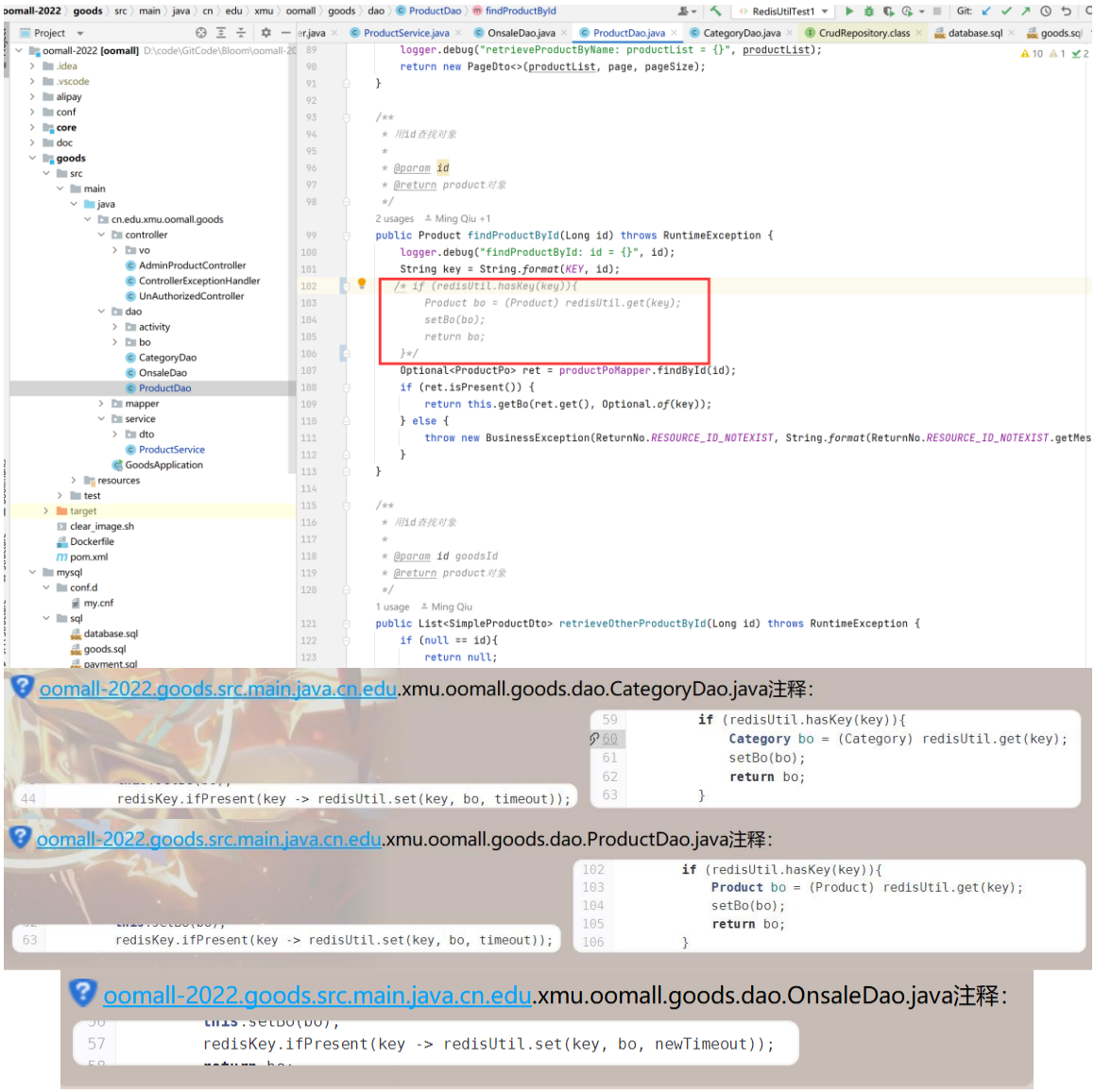


图 2 注释掉 redis 的 Oomall

随后比较了下面的 API 读写效率

API	API 描述
查询商品 SKU 完整信息	GET /products/{id}

四. 实验设计

设置 druid 初始化连接个数、最小连接池数量，最大连接池数量均与原 oomall 程序包的设置相同。

在 jmeter 下设置启动线程所需时间为 10 秒，循环次数 1 次。

设置请求中的参数 100%可以从数据库中取出，防止发生响应失败的情况对实验产生影响。

设置固定定时器 150ms 以避免服务器 connet 的影响，测试线程数设置为 100，200，400，600，1200 来寻找实验数据可行的线程数的边界，确定了边界的线程数后，再设计几组实验来查看有 cloneobj 和无 cloneobj 情况下的响应时间的差别。

而对于 redis 的实验，我们在已经探明实验数据可行的线程数的边界后。我们取了 100，200，300，400 线程来比较效率，且重复请求单个{id}。

五. 实验过程与分析

测试可用线程数边界通过对几组的线程数进行测试,观察其活跃线程数图标与响应时间百分比图标， 得到以下结果:

1. CloneVo 效率对比

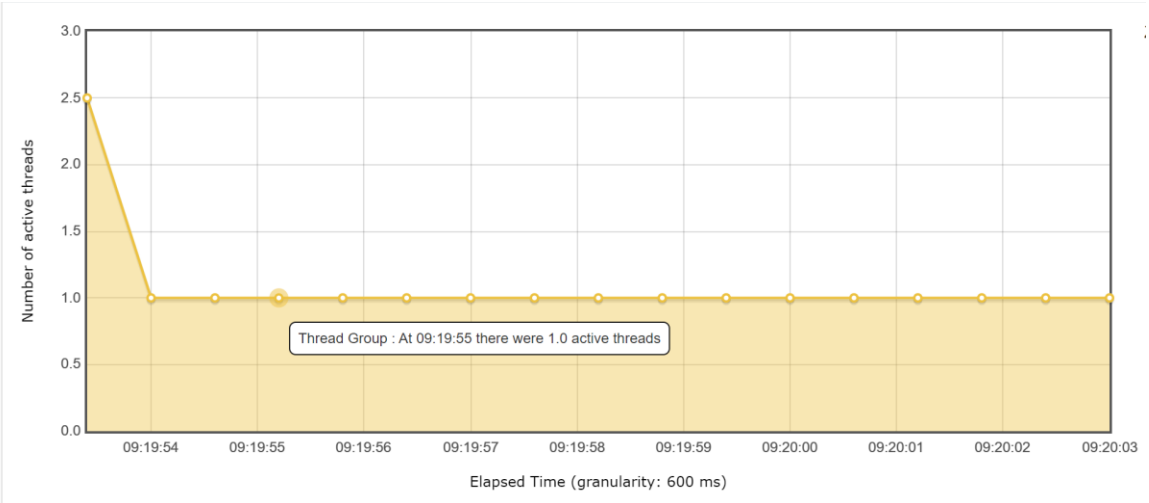
在进行记录数据前，先进行多次 1000 线程的测试，使 JVM 预热后再开始实验以避免 JVM 预热带来的实验误差。

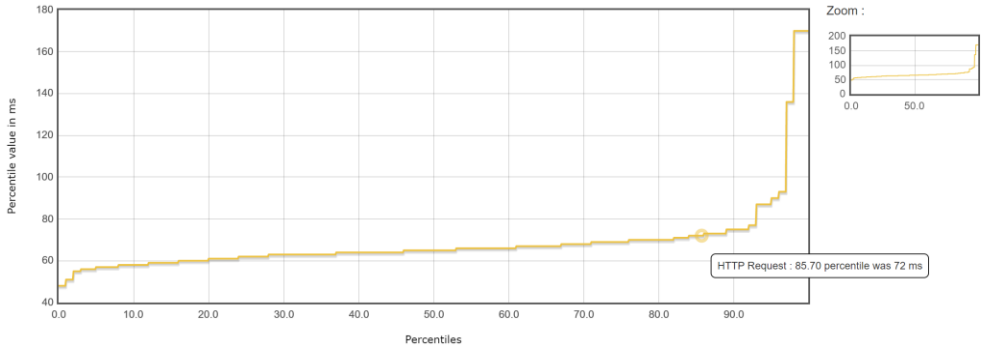
然后开始对几组的线程数进行测试,观察其活跃线程数图标与响应时间百分比图标，得到以下结果:

有 CloneObj 的情况:

100 线程数:响应时间取 92%线程响应 78ms

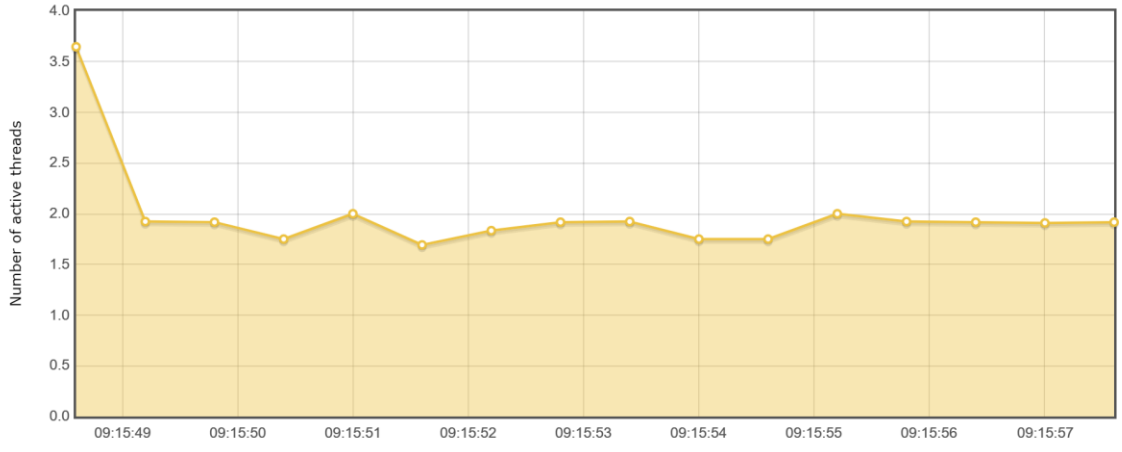
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	100	0	0.00%	68.42	48	170	65.00	75.00	89.85	170.00	10.33	7.46	1.31
HTTP Request	100	0	0.00%	68.42	48	170	65.00	75.00	89.85	170.00	10.33	7.46	1.31

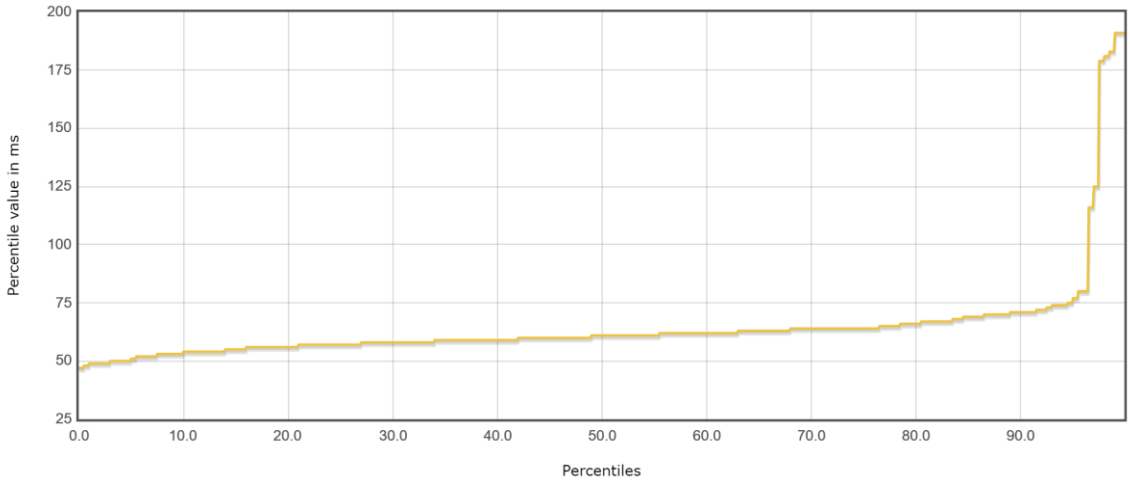




200 线程数：响应时间取 95%线程响应 76.9ms

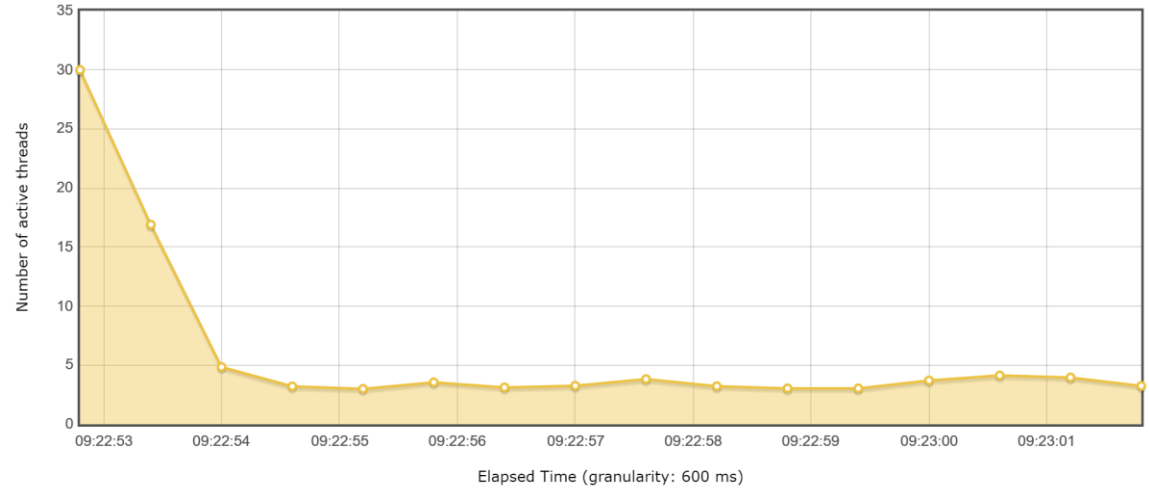
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**200**	**0**	**0.00%**	**64.54**	**47**	**191**	**61.00**	**71.00**	**76.90**	**190.92**	**20.84**	**15.06**	**2.65**
HTTP Request	200	0	0.00%	64.54	47	191	61.00	71.00	76.90	190.92	20.84	15.06	2.65

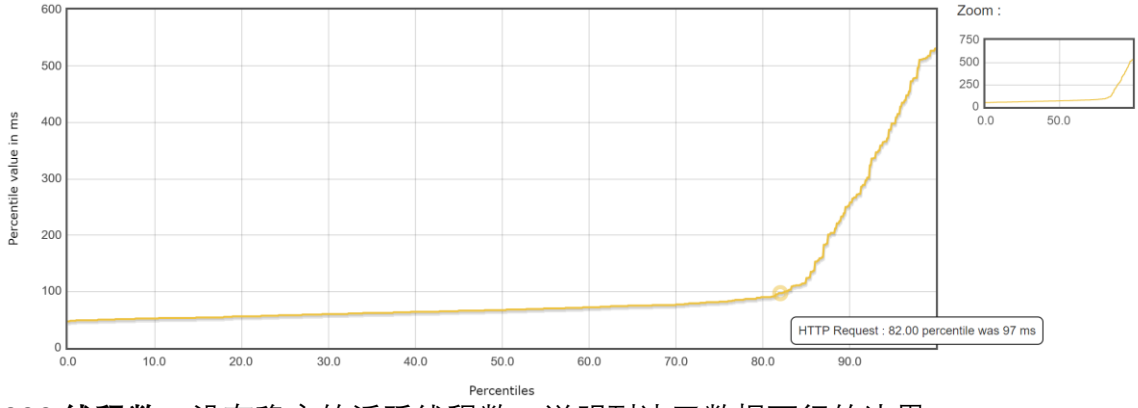




400 线程数：响应时间取 82%线程响应 97ms

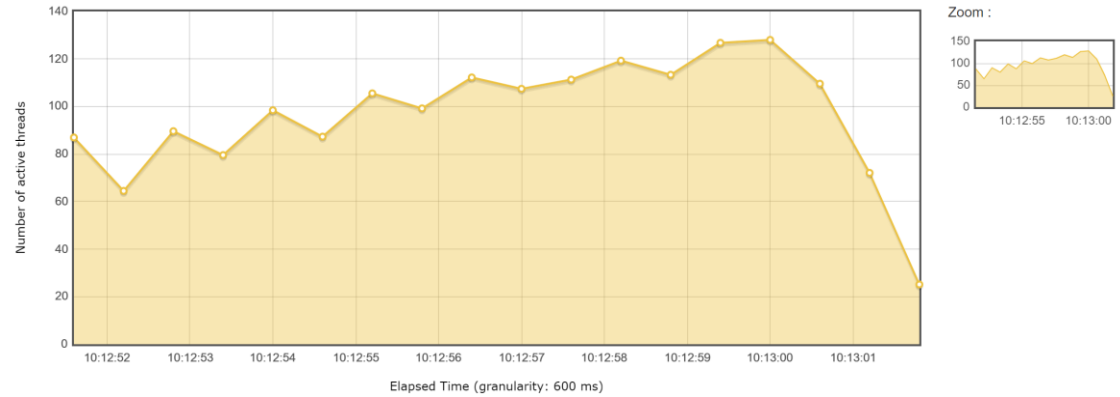
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	400	0	0.00%	105.84	47	531	67.00	257.40	398.00	517.97	43.90	31.73	5.57
HTTP Request	400	0	0.00%	105.84	47	531	67.00	257.40	398.00	517.97	43.90	31.73	5.57

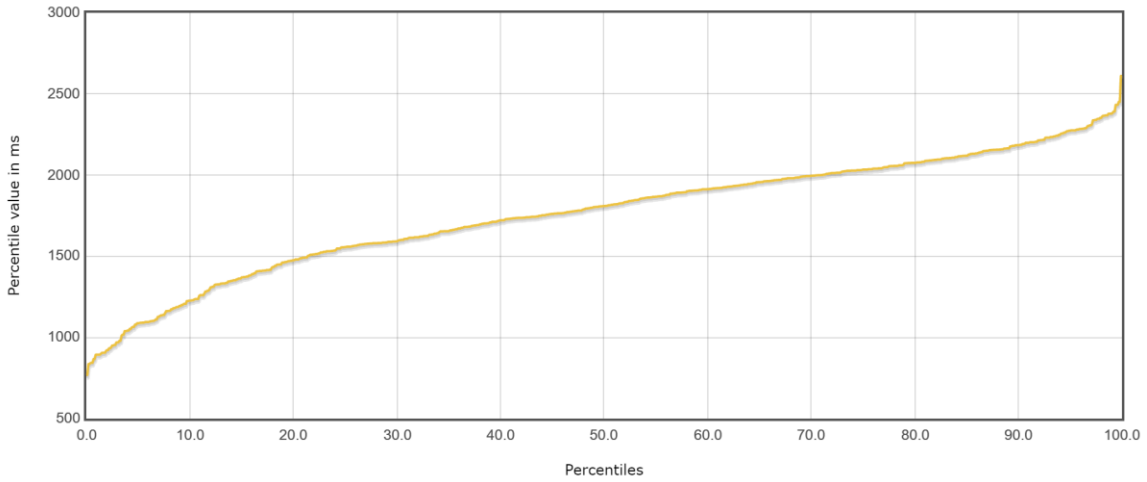




600 线程数：没有稳定的活跃线程数，说明到达了数据可行的边界

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	600	0	0.00%	1764.71	769	2611	1809.50	2185.00	2275.90	2383.95	54.76	39.57	6.95
HTTP Request	600	0	0.00%	1764.71	769	2611	1809.50	2185.00	2275.90	2383.95	54.76	39.57	6.95

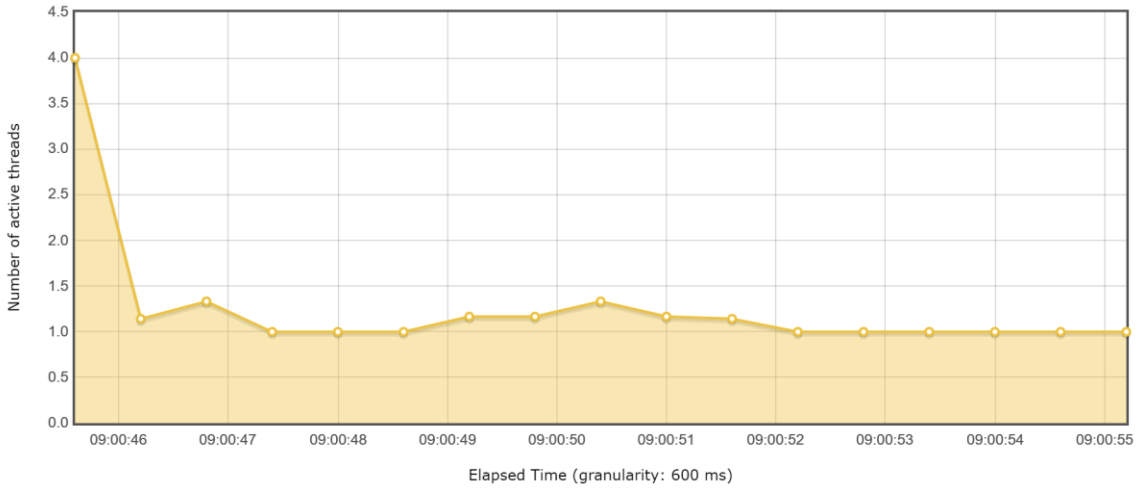


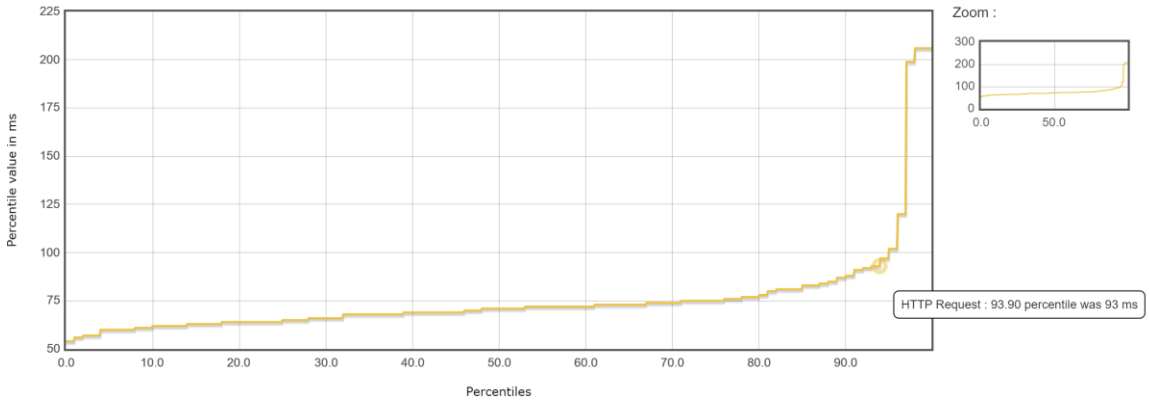


而对于无 cloneobj 的情况

100 线程数：响应时间取 93.9%线程响应 93.9ms

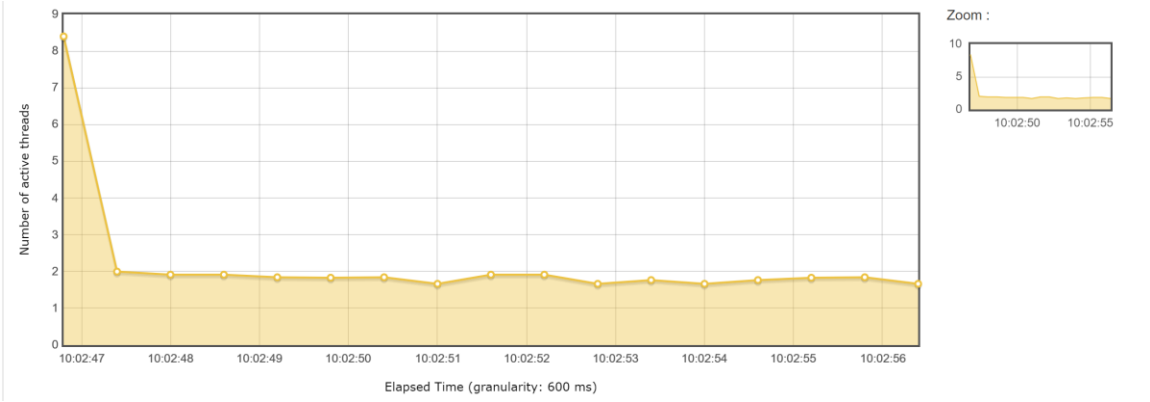
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	100	0	0.00%	75.55	54	206	71.00	87.90	101.75	206.00	10.46	7.56	1.36
HTTP Request	100	0	0.00%	75.55	54	206	71.00	87.90	101.75	206.00	10.46	7.56	1.36

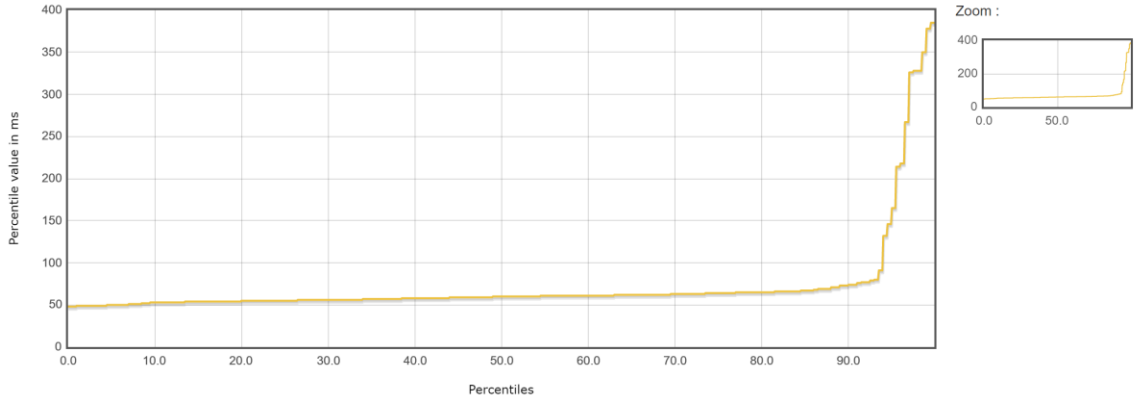




200 线程数： 响应时间取 93.9%线程响应 78.9ms

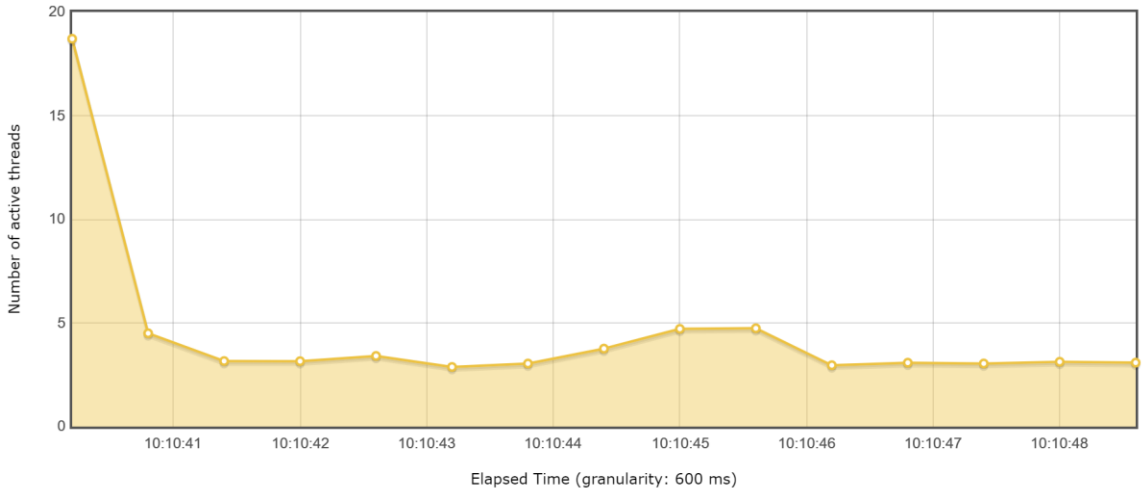
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**200**	**0**	**0.00%**	**72.17**	**48**	**385**	**60.00**	**73.90**	**164.05**	**377.72**	**20.94**	**15.13**	**2.72**
HTTP Request	200	0	0.00%	72.17	48	385	60.00	73.90	164.05	377.72	20.94	15.13	2.72

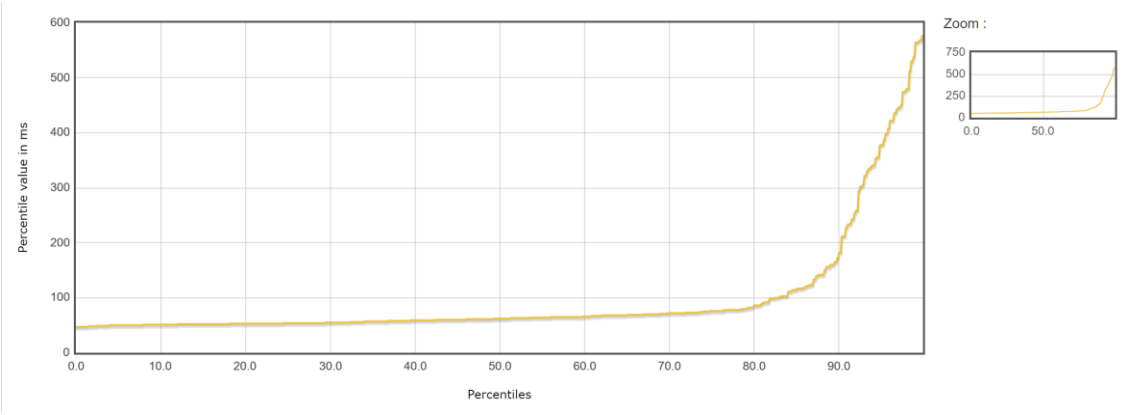




400 线程数: 响应时间取 80.9%线程响应 90.9ms

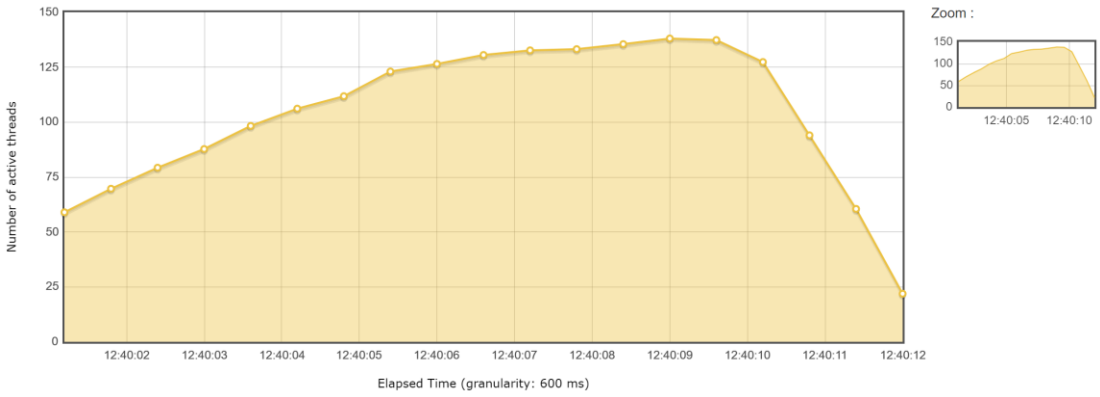
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**400**	**0**	**0.00%**	**97.87**	**47**	**576**	**62.00**	**181.10**	**377.95**	**563.75**	**43.63**	**31.53**	**5.67**
HTTP Request	400	0	0.00%	97.87	47	576	62.00	181.10	377.95	563.75	43.63	31.53	5.67

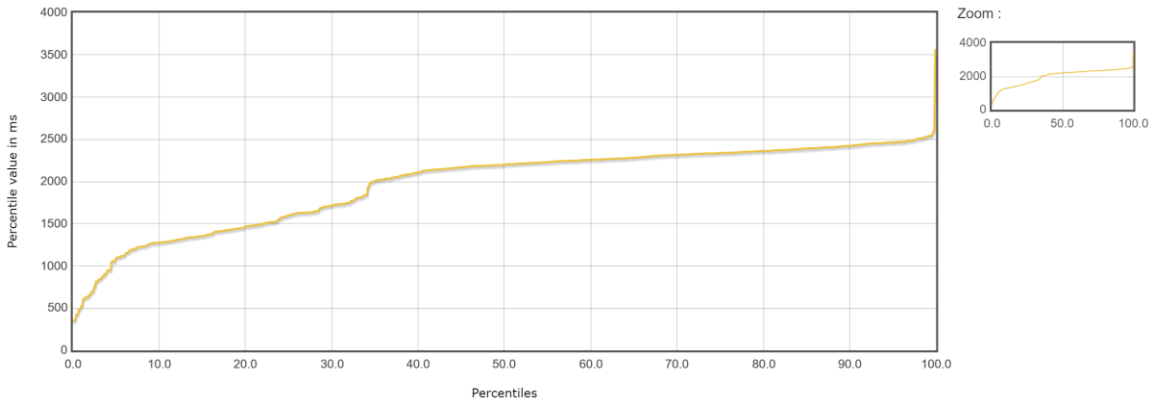




600 线程数: 没有稳定的活跃线程数，说明到达了数据可行的边界

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**600**	**0**	**0.00%**	**1974.53**	**354**	**3558**	**2199.50**	**2424.00**	**2464.95**	**2537.93**	**52.59**	**38.01**	**6.68**
HTTP Request	600	0	0.00%	1974.53	354	3558	2199.50	2424.00	2464.95	2537.93	52.59	38.01	6.68

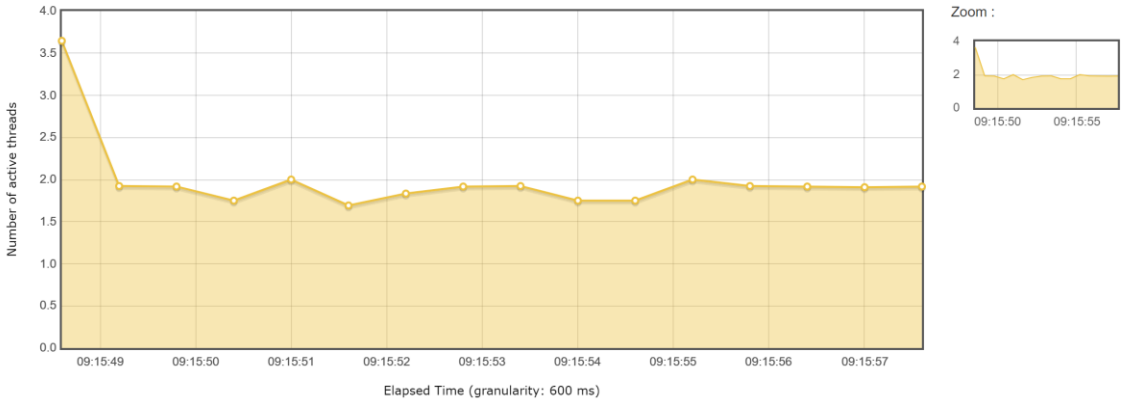


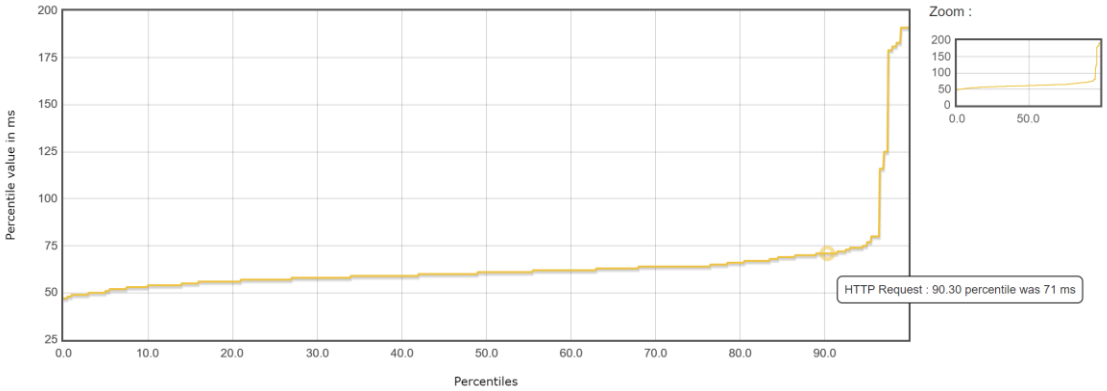


从上面的测试实验中，可以看出数据可行的边界为 600 线程数，发出线程线程所需为 10 秒，因此补充实验数据为 300，500 线程来确认两种方案下的响应时间的差别

300 线程且使用 cloneobj：响应时间取 90.3%线程响应 71ms

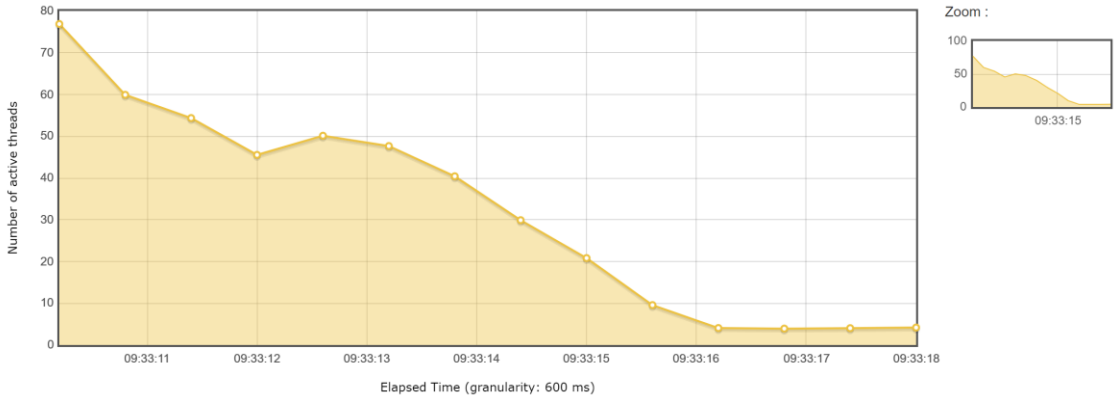
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**200**	**0**	**0.00%**	**64.54**	**47**	**191**	**61.00**	**71.00**	**76.90**	**190.92**	**20.84**	**15.06**	**2.65**
HTTP Request	200	0	0.00%	64.54	47	191	61.00	71.00	76.90	190.92	20.84	15.06	2.65

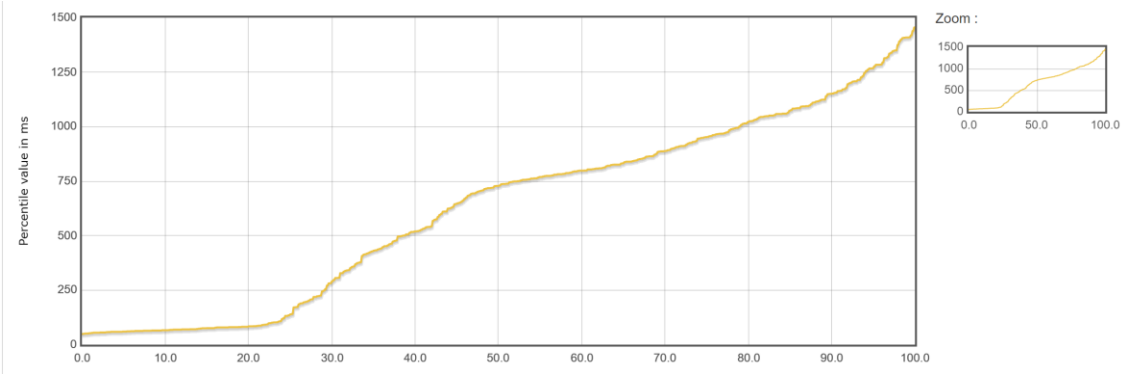




500 线程且使用 cloneobj：响应时间取 23.9%线程响应 90.9ms

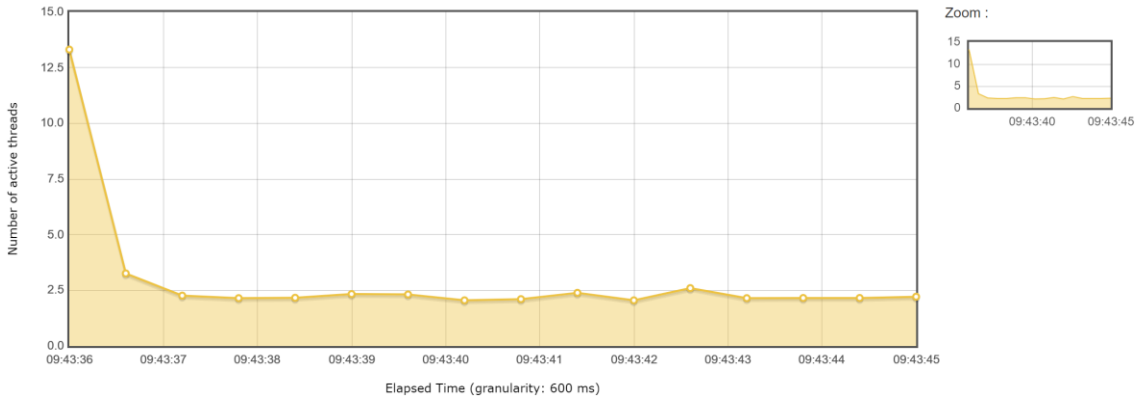
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	500	0	0.00%	625.96	49	1455	729.00	1151.90	1276.55	1409.00	55.02	39.76	6.98
HTTP Request	500	0	0.00%	625.96	49	1455	729.00	1151.90	1276.55	1409.00	55.02	39.76	6.98

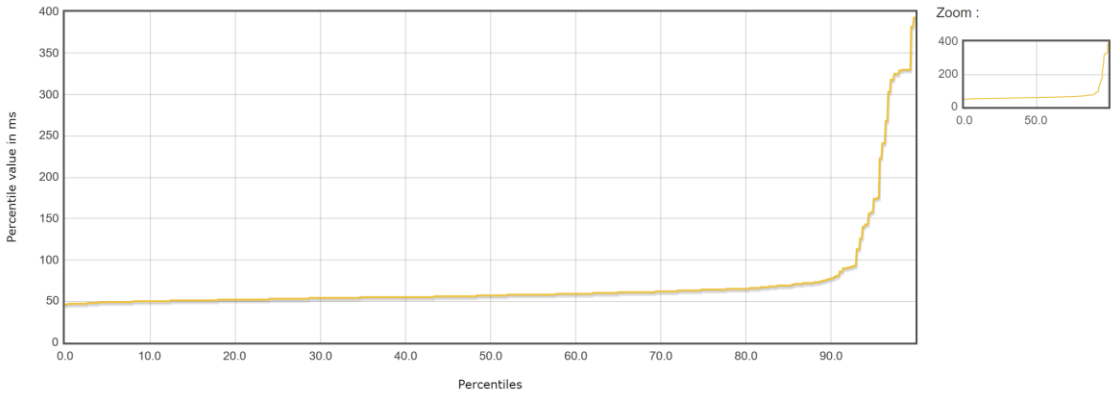




300 线程且不使用 cloneobj：响应时间取 92.9%线程响应 86.9ms

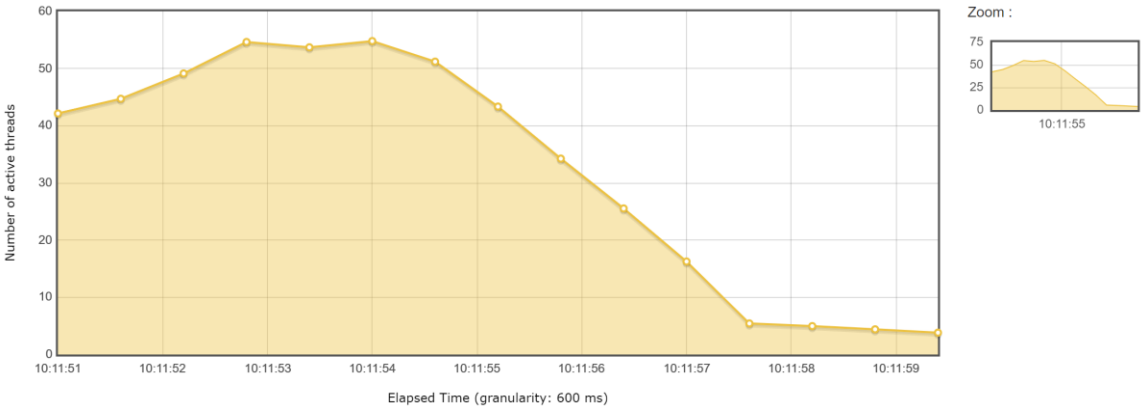
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	300	0	0.00%	71.66	46	393	57.00	77.90	173.20	330.00	32.38	23.40	4.21
HTTP Request	300	0	0.00%	71.66	46	393	57.00	77.90	173.20	330.00	32.38	23.40	4.21

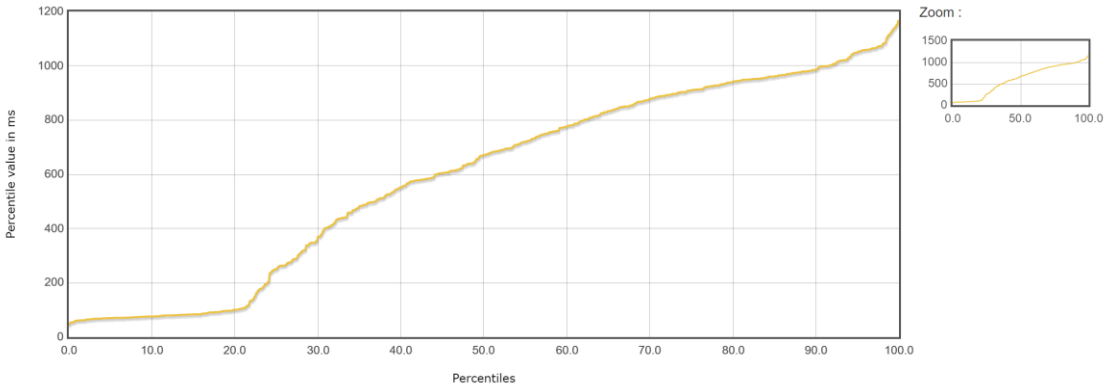




500 线程且不使用 cloneobj：响应时间取 22.9%线程响应 99.9ms

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**500**	**0**	**0.00%**	**593.58**	**47**	**1166**	**671.50**	**987.80**	**1052.80**	**1126.91**	**54.58**	**39.44**	**7.09**
HTTP Request	500	0	0.00%	593.58	47	1166	671.50	987.80	1052.80	1126.91	54.58	39.44	7.09

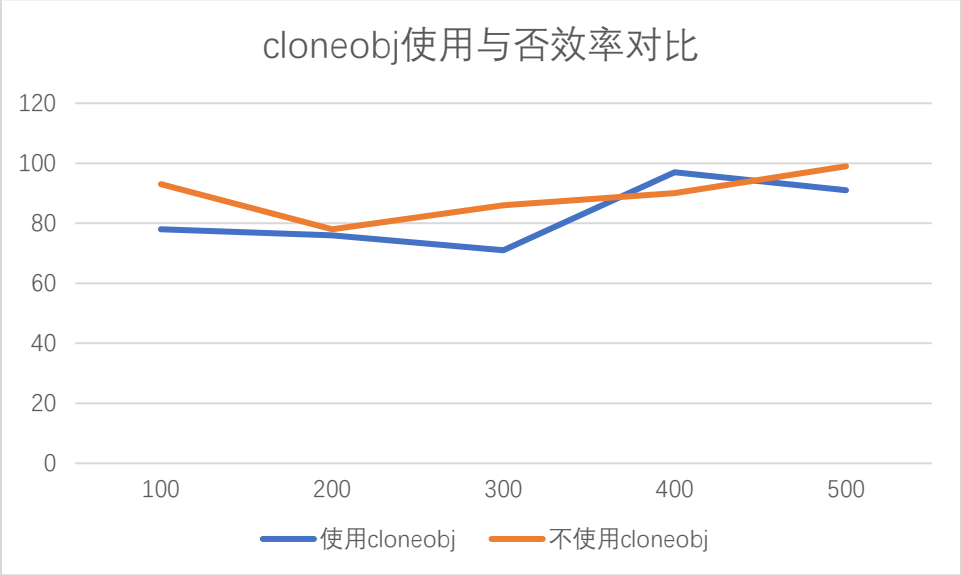




画出表格如下：

线程数	使用 cloneobj	不使用 cloneobj	响应时间比
100	78	93	0.838709677
200	76	78	0.974358974
300	71	86	0.825581395
400	97	90	1.077777778
500	91	99	0.919191919

画出响应时间对比图如下：



所以我们可以得到结论，使用 cloneobj 与否，不会对效率产生很大的影响。

2. Redis 缓存的使用效率对比

在这部分的实验中，我们在进行记录数据前，先进行多次 1000 线程的测试，使 JVM 预热后再开始实验以避免 JVM 预热带来的实验误差。

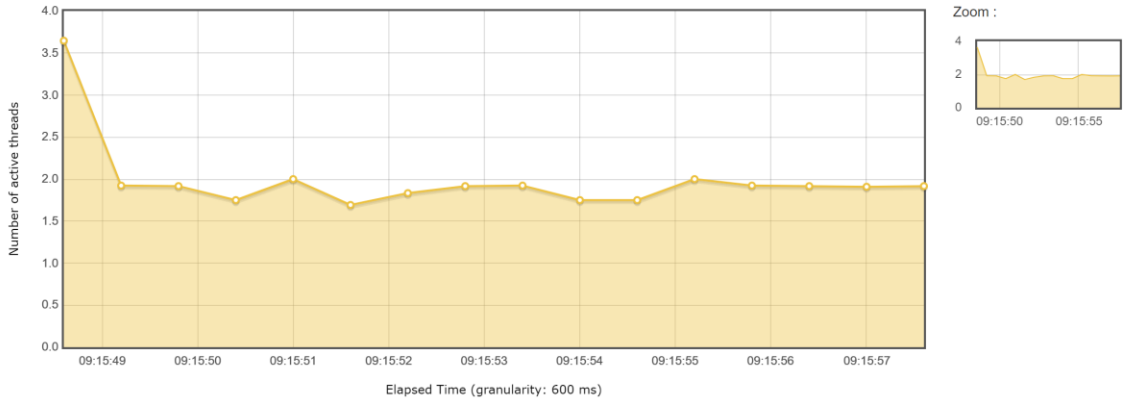
我们一开始尝试与实验 2 的结果对比，但是实验结果显示，实验 2 的效率大幅优于本次实验。所以我们怀疑是 redis 没有发挥作用，我们又有了以下的实验：把 oomall 中 goods 模块的 redis 相关代码尽量注释掉（有些代码包含在流式编程中，去不掉就不去除），然后与不注释掉的结果对比。我们认为，如果 redis 能提高读取的效率，那么预期结果应该是使用了 redis 的代码效率远快于注释掉 redis 的 diamond

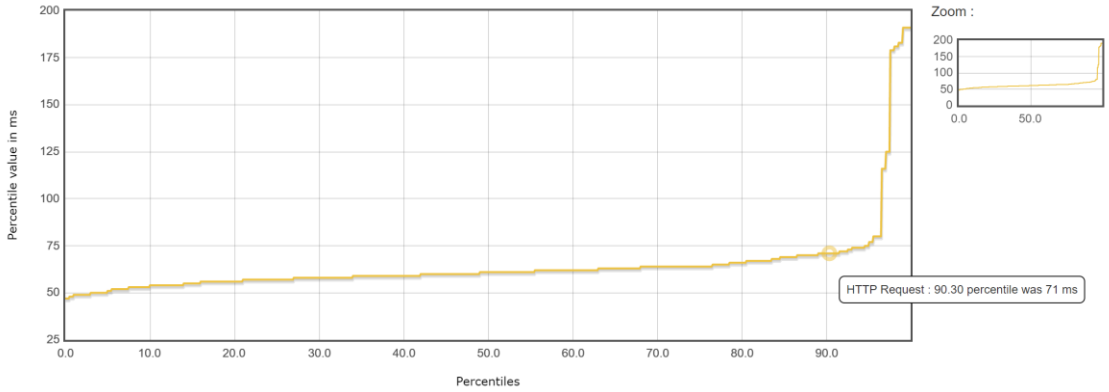
此外，由于是探究缓存的影响，所以我们设置只访问一个{id}。此外，我们测试有效速率的边界，得到结果如下

使用 redis 缓存：

300 线程: 响应时间取 90.3%线程响应 71ms

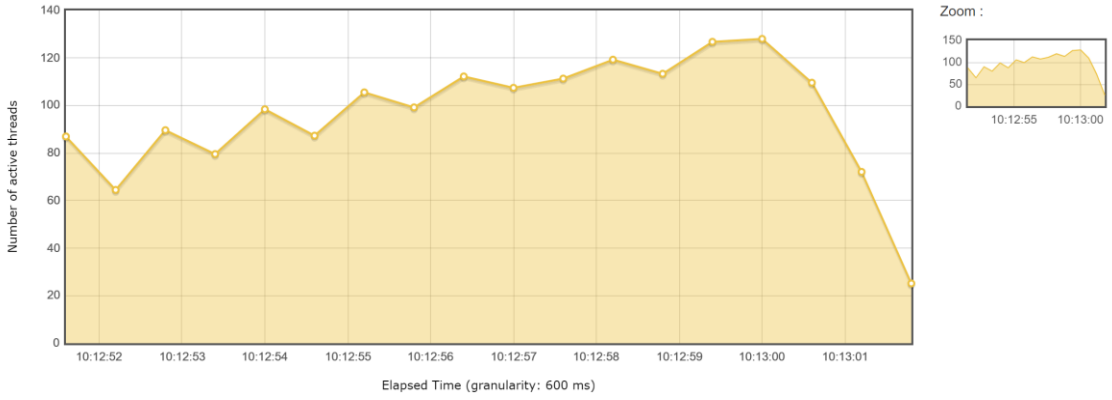
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	200	0	0.00%	64.54	47	191	61.00	71.00	76.90	190.92	20.84	15.06	2.65
HTTP Request	200	0	0.00%	64.54	47	191	61.00	71.00	76.90	190.92	20.84	15.06	2.65

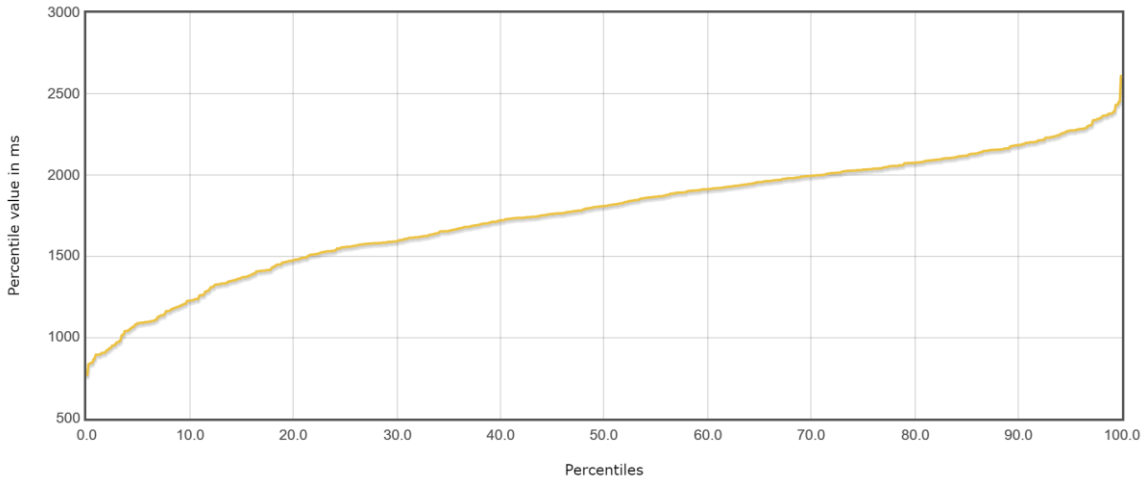




600 线程：没有稳定的活跃线程数，说明到达了数据可行的边界

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**600**	**0**	**0.00%**	**1764.71**	**769**	**2611**	**1809.50**	**2185.00**	**2275.90**	**2383.95**	**54.76**	**39.57**	**6.95**
HTTP Request	600	0	0.00%	1764.71	769	2611	1809.50	2185.00	2275.90	2383.95	54.76	39.57	6.95

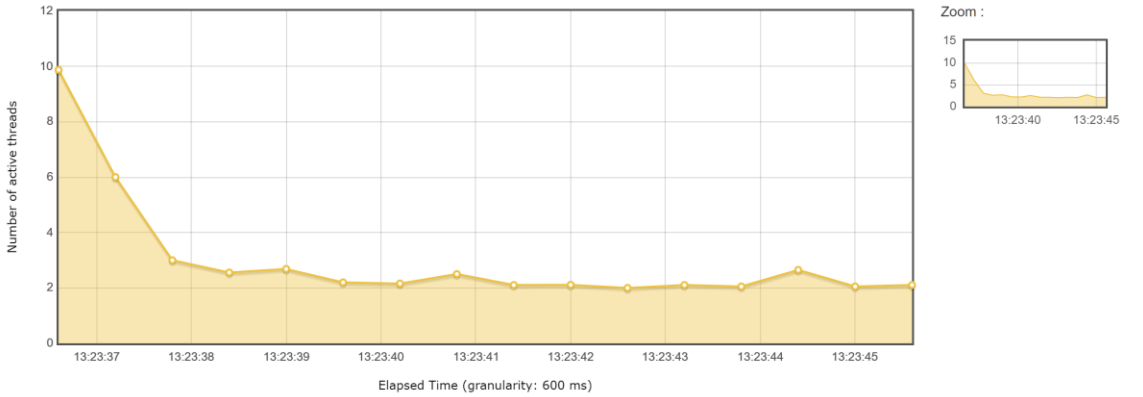


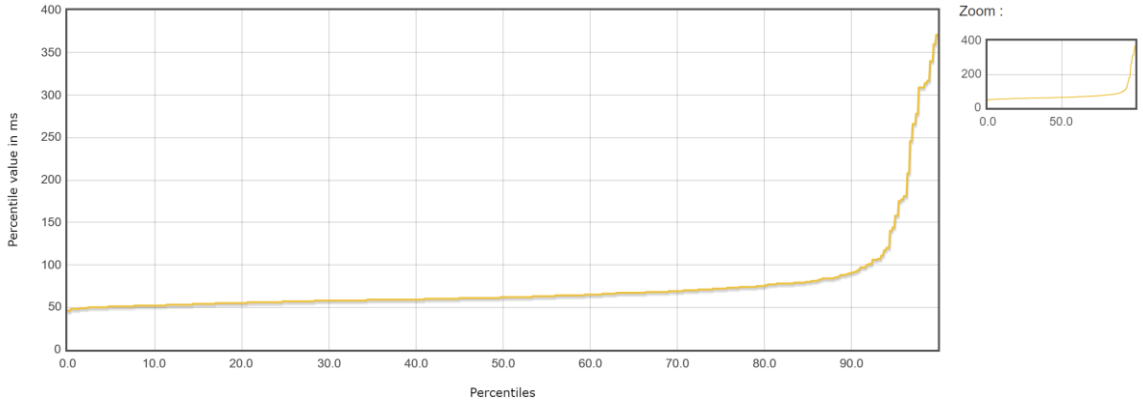


注释掉 redis 缓存的代码段：

300 线程: 响应时间取 90.3%线程响应 92ms

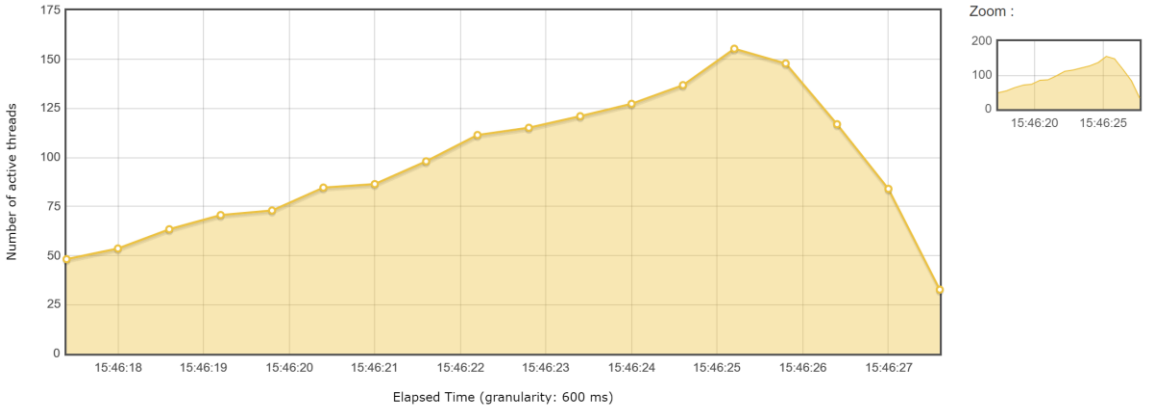
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**300**	**0**	**0.00%**	**75.11**	**46**	**371**	**62.00**	**90.90**	**157.30**	**339.77**	**32.47**	**23.47**	**4.12**
HTTP Request	300	0	0.00%	75.11	46	371	62.00	90.90	157.30	339.77	32.47	23.47	4.12

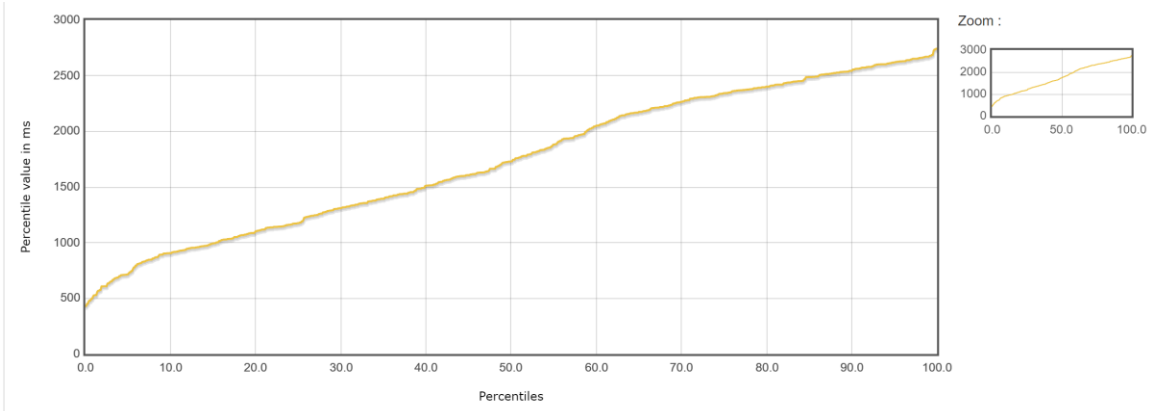




600 线程：没有稳定的活跃线程数，说明到达了数据可行的边界

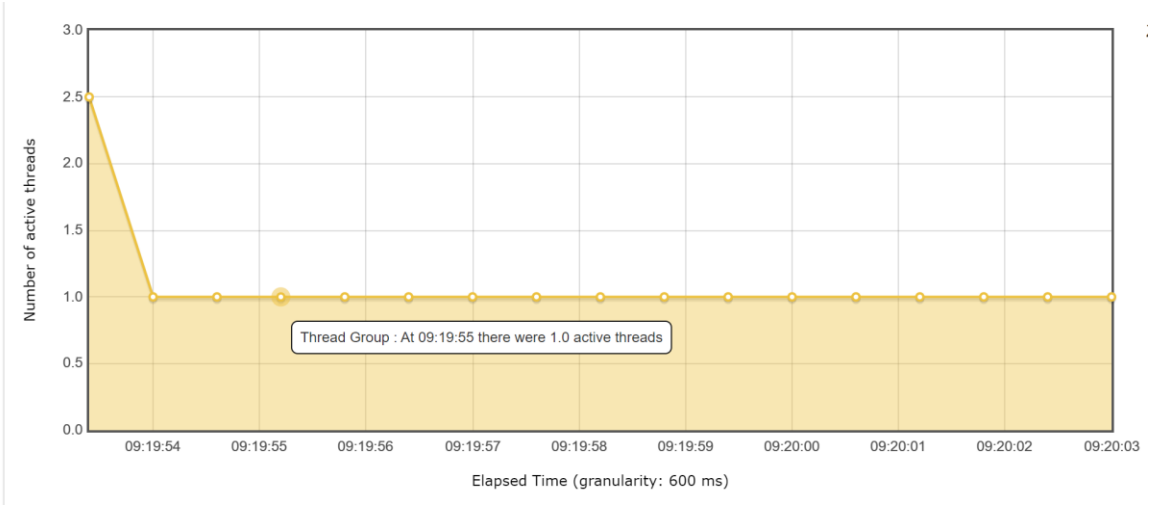
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	600	0	0.00%	1742.11	431	2742	1729.50	2553.90	2621.85	2674.95	54.17	39.15	6.88
HTTP Request	600	0	0.00%	1742.11	431	2742	1729.50	2553.90	2621.85	2674.95	54.17	39.15	6.88

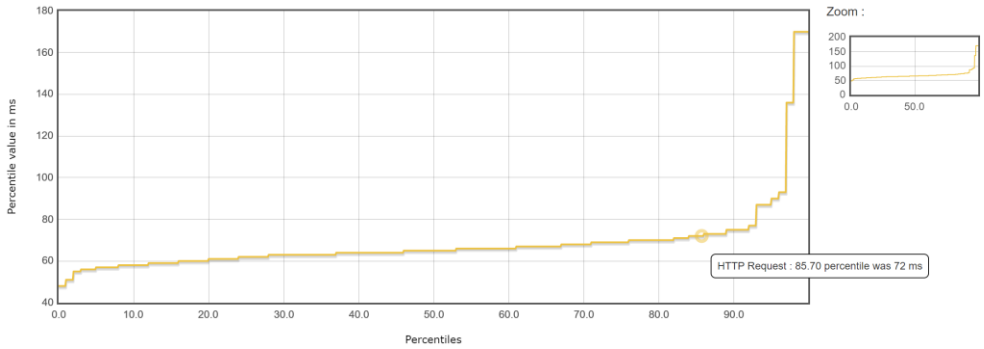




然后我们对比了 100，200，300，400 线程数的情况下，有无 redis 缓存的代码的效率：
结果如下
有 redis 缓存的 100 线程：响应时间取 92%线程响应 78ms

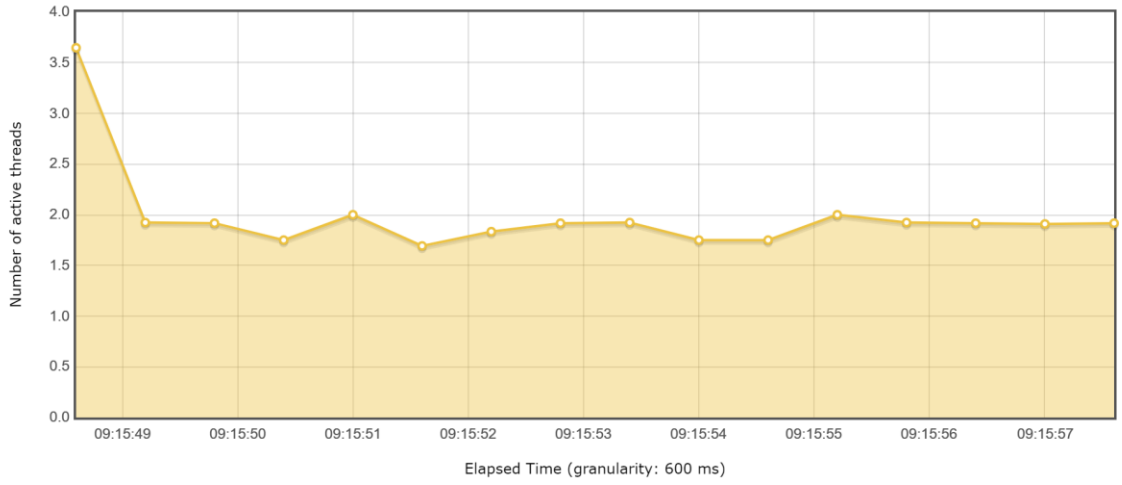
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	100	0	0.00%	68.42	48	170	65.00	75.00	89.85	170.00	10.33	7.46	1.31
HTTP Request	100	0	0.00%	68.42	48	170	65.00	75.00	89.85	170.00	10.33	7.46	1.31

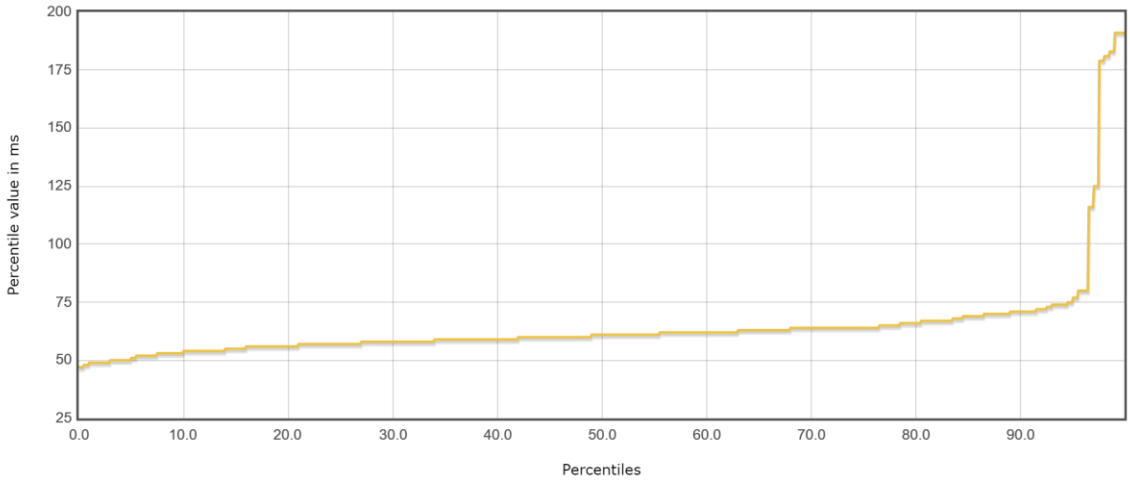




有 redis 缓存的 200 线程：响应时间取 95%线程响应 76.9ms

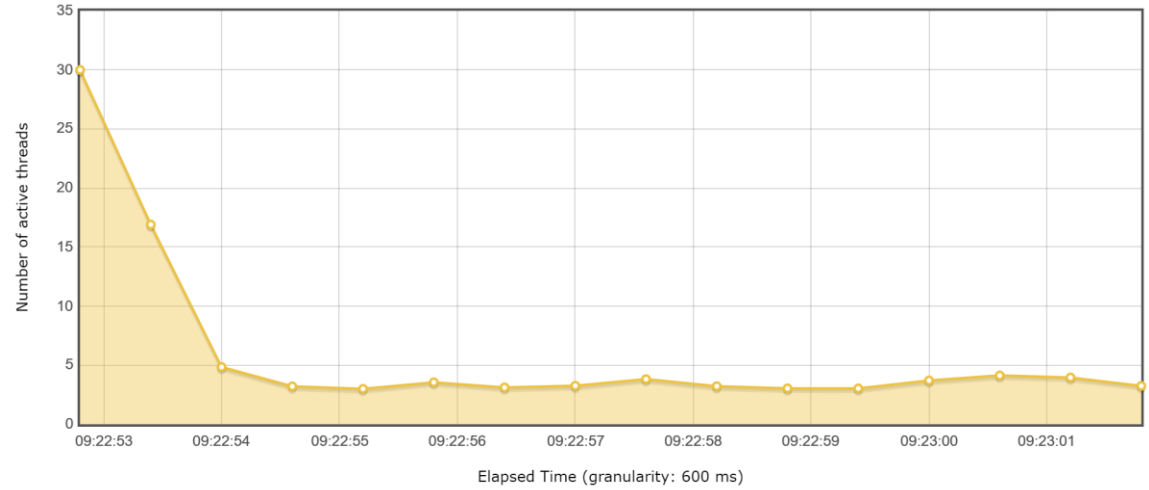
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**200**	**0**	**0.00%**	**64.54**	**47**	**191**	**61.00**	**71.00**	**76.90**	**190.92**	**20.84**	**15.06**	**2.65**
HTTP Request	200	0	0.00%	64.54	47	191	61.00	71.00	76.90	190.92	20.84	15.06	2.65

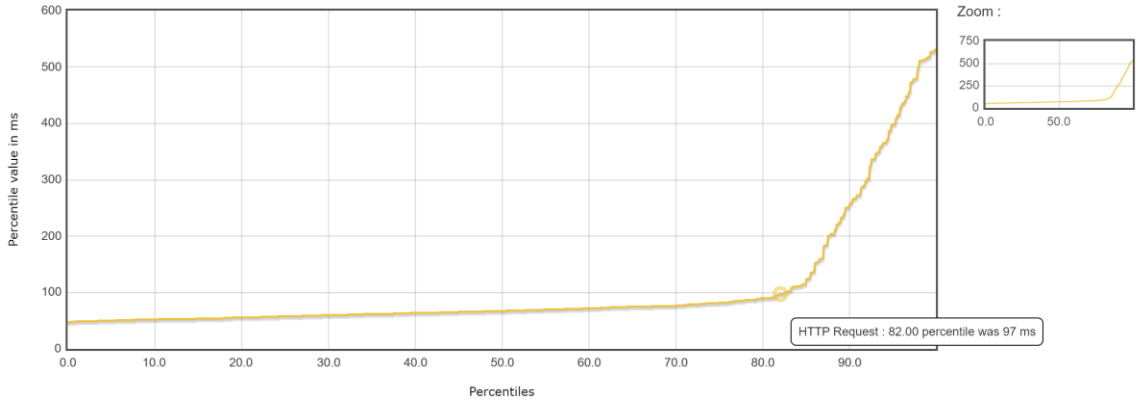




有 redis 缓存的 400 线程：响应时间取 82%线程响应 97ms

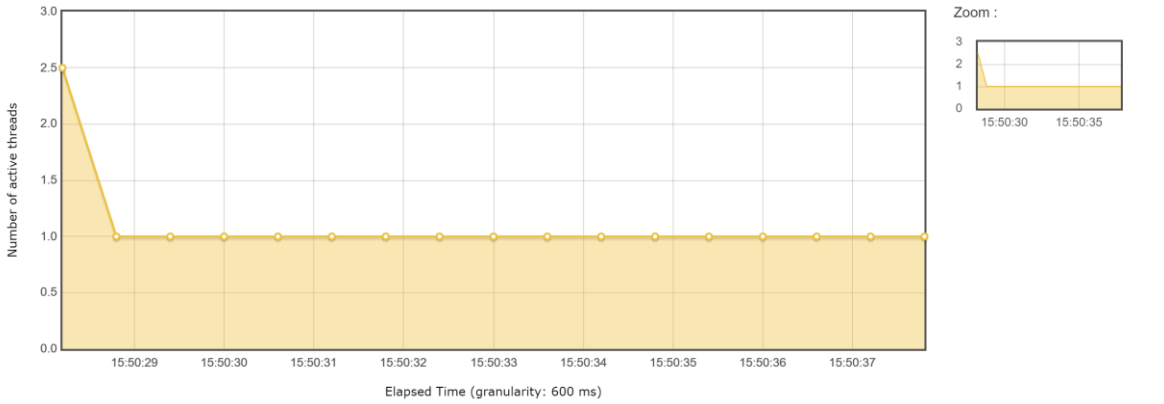
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**400**	**0**	**0.00%**	**105.84**	**47**	**531**	**67.00**	**257.40**	**398.00**	**517.97**	**43.90**	**31.73**	**5.57**
HTTP Request	400	0	0.00%	105.84	47	531	67.00	257.40	398.00	517.97	43.90	31.73	5.57

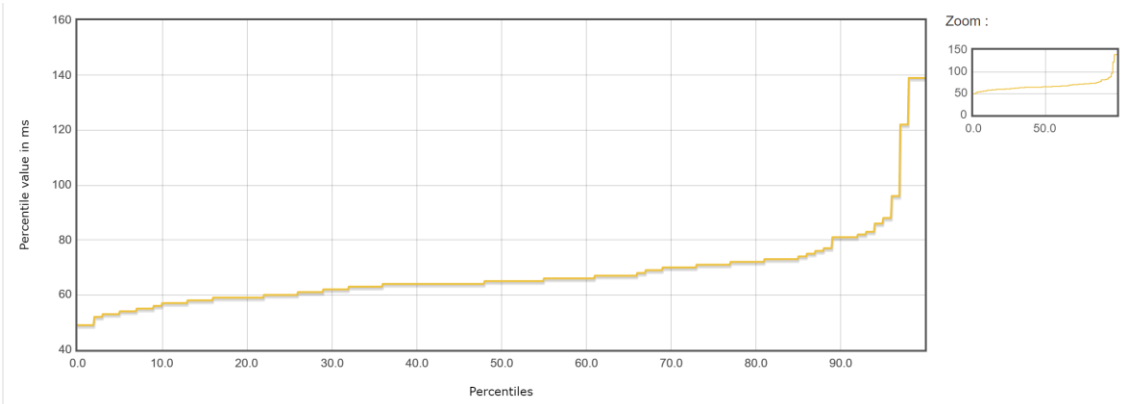




注释掉 redis 缓存的代码段的 100 线程：响应时间取 90%线程响应 80ms

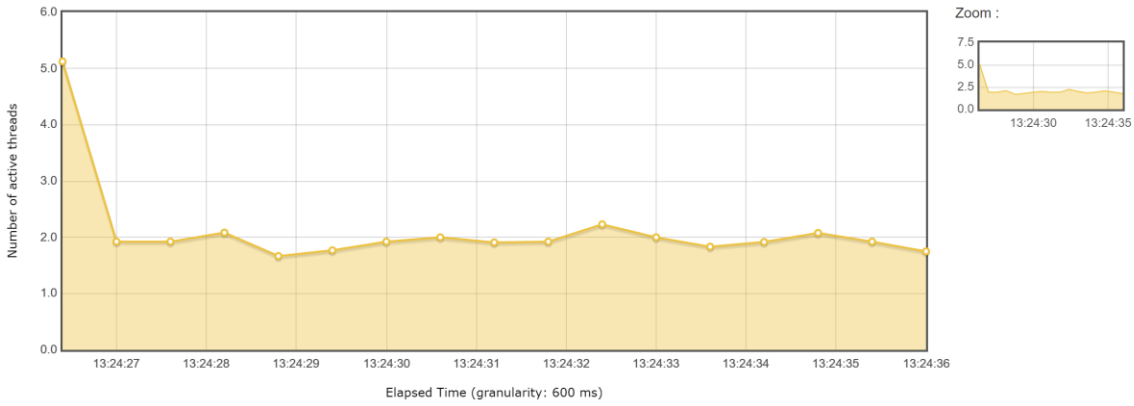
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**100**	**0**	**0.00%**	**67.74**	**49**	**139**	**65.00**	**81.00**	**87.90**	**139.00**	**10.35**	**7.48**	**1.31**
HTTP Request	100	0	0.00%	67.74	49	139	65.00	81.00	87.90	139.00	10.35	7.48	1.31

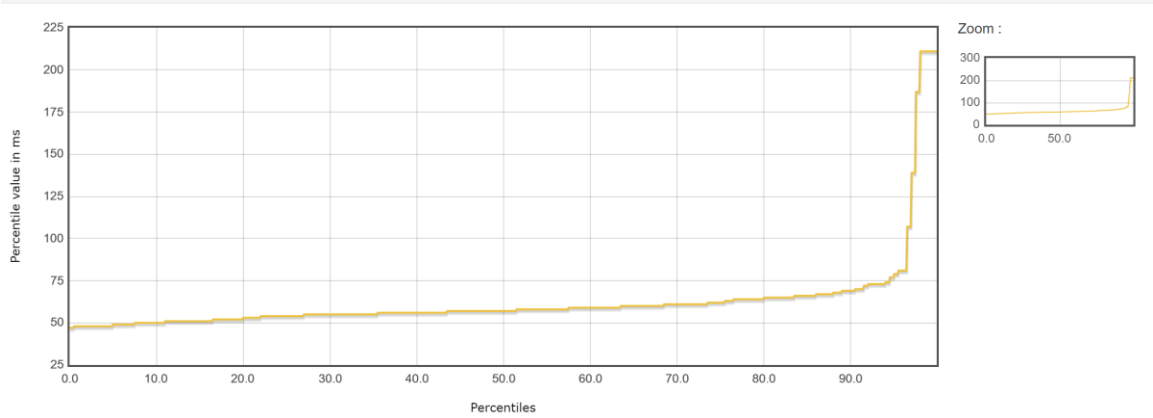




注释掉 redis 缓存的代码段的 200 线程：响应时间取 89%线程响应 69ms

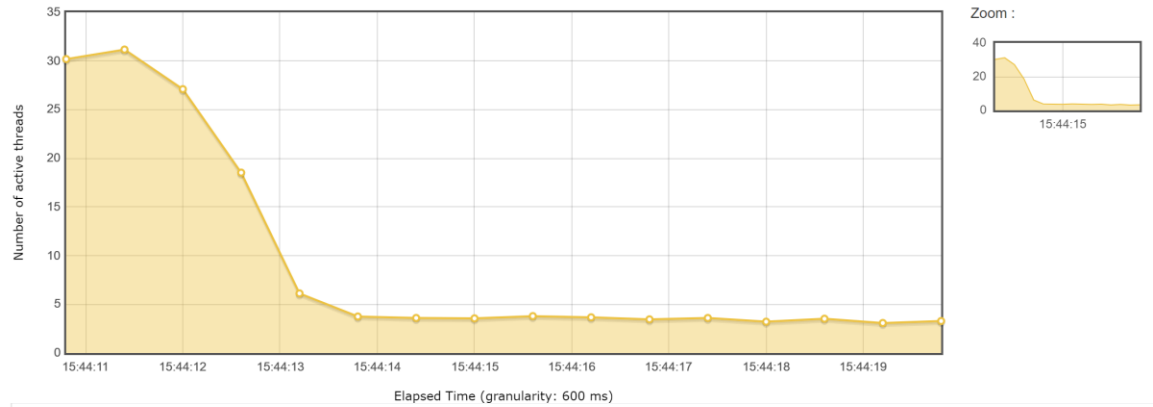
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**200**	**0**	**0.00%**	**62.50**	**47**	**211**	**57.00**	**69.00**	**78.90**	**211.00**	**20.78**	**15.02**	**2.64**
HTTP Request	200	0	0.00%	62.50	47	211	57.00	69.00	78.90	211.00	20.78	15.02	2.64

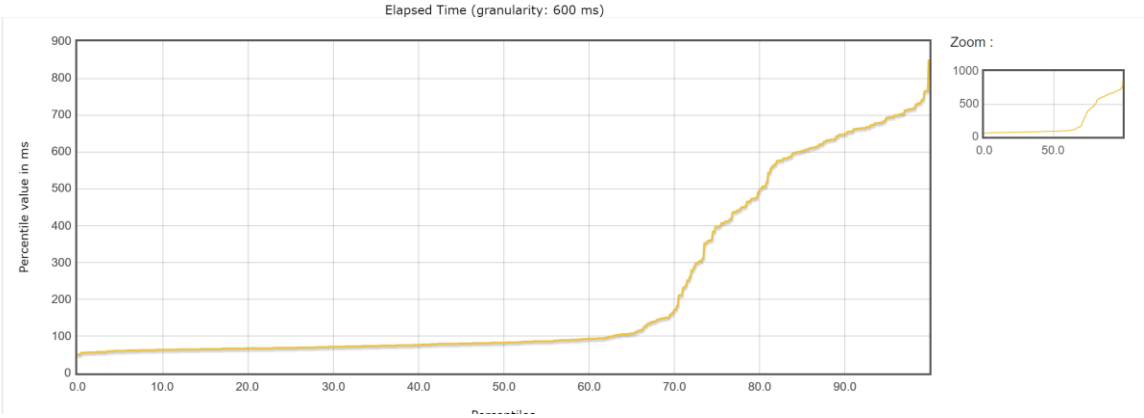




注释掉 redis 缓存的代码段的 400 线程：响应时间取 64%线程响应 106ms

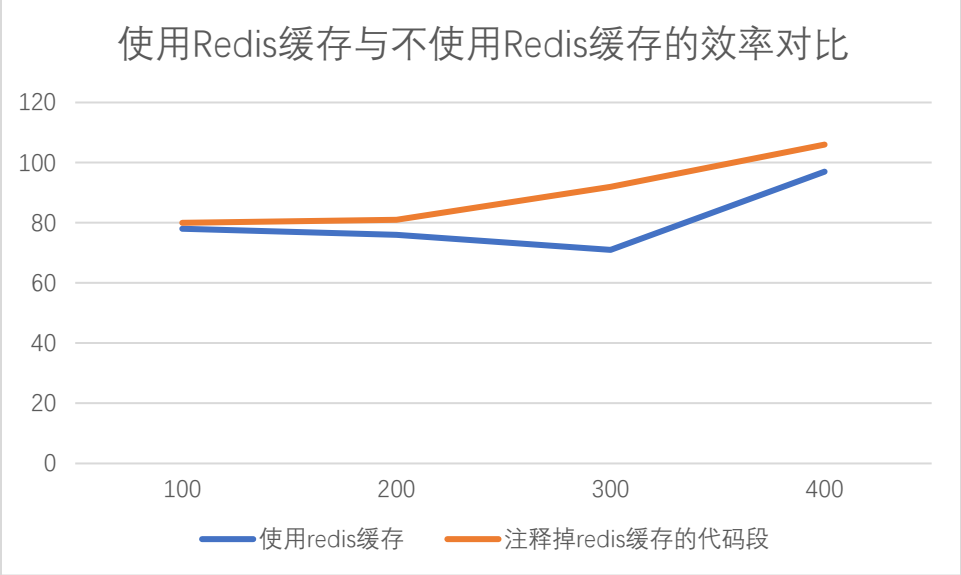
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**400**	**0**	**0.00%**	**221.26**	**51**	**848**	**83.00**	**649.80**	**693.90**	**741.92**	**42.76**	**30.90**	**5.43**
HTTP Request	400	0	0.00%	221.26	51	848	83.00	649.80	693.90	741.92	42.76	30.90	5.43





画出表格如下：

线程数	使用 redis 缓存	注释掉 redis 缓存的代码段
100	78	80
200	76	81
300	71	92
400	97	106



可以看到：把 redis 缓存的相关代码注释掉后,代码的运行效率没有明显变化，或是略有下降，而堵塞的边界也几乎一样。对此，我们小组怀疑这段 redis 的代码没有起作用，因此我们的最终结论是：要么 redis 的那段代码块没有起作用

六. 实验结论

1. 使用 cloneobj 与否效率几乎一样
2. OOMALL 的 Redis 模块在这个 API 中很可能没有发挥作用。

七. 附录

git 地址： https://github.com/Elaina-Kaslana/J2EE3.git