[**轻量级压测工具hey介绍与实践**](https://www.cnblogs.com/wintersun/p/12492096.html)

**轻量级压测工具hey介绍与实践**

<https://github.com/rakyll/hey>

**二进制安装**

Linux 64-bit: <https://storage.googleapis.com/hey-release/hey_linux_amd64>

Mac 64-bit: <https://storage.googleapis.com/hey-release/hey_darwin_amd64>

Windows 64-bit: <https://storage.googleapis.com/hey-release/hey_windows_amd64>

**源码安装**

**需要提交安装GO语言环境**[**https://www.runoob.com/go/go-environment.html**](https://www.runoob.com/go/go-environment.html)

git clone <https://github.com/rakyll/hey>.git

go get -u github.com/rakyll/hey

go install github.com/rakyll/hey

go build

**hey压测简明示例**

指定时长的get请求：客户端(-c)并发为2， 持续发送请求2s (-c)

hey -z 5s -c 2 https://www.baidu.com/

指定请求总数的get请求：运行2000次(-n)，客户端并发为50(-c)

hey -n 2000 -c 50  https://www.baidu.com/

指定host的get请求：使用的cpu核数为2 (-cpus), 压测时长为5s(-z), 并发数为2

hey -z 5s -c 2 -cpus 2 -host "baidu.com" https://220.181.38.148

请求带header的get接口：压测时长为5s (-z), 客户端发送请求的速度为128QPS, 请求头用-H添加

hey -z 5s -q 128 -H "client-ip:0.0.0.0" -H "X-Up-Calling-Line-Id:X.L.Xia" https://www.baidu.com/

请求post请求

hey -z 5s -c 50 -m POST -H "info:firstname=xiuli; familyname=xia" -d "year=2020&month=1&day=21" https://www.baidu.com/

代理模式，需额外配置proxy：因部分ip频繁发请求有风险，故可用-x设置白名单代理向服务器发请求

hey -z 5s -c 10 -x "http://127.0.0.1:8001" <http://baidu.com/>

shell for循环实现压测

for i in `seq 10`; do curl -v http://baidu.com; done

**使用方法**

Usage: hey [options...] <url>

Options:

  // 指定运行的总请求数。默认值为200。

  -n  Number of requests to run. Default is 200.

  // 客户端并发执行的请求数，默认为50。总请求数不能小于并发数。

  -c  Number of workers to run concurrently. Total number of requests cannot

      be smaller than the concurrency level. Default is 50.

  // 客户端发送请求的速度限制，以每秒响应数QPS为单位，默认没有限制。

  -q  Rate limit, in queries per second (QPS) per worker. Default is no rate limit.

  // 发送请求的持续时长，超时后程序停止并退出。若指定了持续时间，则忽略总请求数(-n)，例如-z 10s，-z 3m

  -z  Duration of application to send requests. When duration is reached,

      application stops and exits. If duration is specified, n is ignored.

      Examples: -z 10s -z 3m.

  // 输出类型。若没有提供，则打印摘要。CSV是唯一支持的格式，结果以逗号分隔各个指标项。

  -o  Output type. If none provided, a summary is printed.

      "csv" is the only supported alternative. Dumps the response

      metrics in comma-separated values format.

  // HTTP方法，例如GET，POST，PUT，DELETE，HEAD，OPTIONS方法

  -m  HTTP method, one of GET, POST, PUT, DELETE, HEAD, OPTIONS.

  // HTTP请求头，可以用-H连续添加多个请求头。

  -H  Custom HTTP header. You can specify as many as needed by repeating the flag.

      For example, -H "Accept: text/html" -H "Content-Type: application/xml" .

  // 每个请求的超时时间（以秒为单位）。默认值为20s，数值0代表永不超时。

  -t  Timeout for each request in seconds. Default is 20, use 0 for infinite.

  // HTTP响应头

  -A  HTTP Accept header.

  // HTTP请求正文

  -d  HTTP request body.

  // HTTP请求正文文件

  -D  HTTP request body from file. For example, /home/user/file.txt or ./file.txt.

  // 内容类型，默认为“ text / html”。

  -T  Content-type, defaults to "text/html".

  // 基本身份验证，用户名：密码。

  -a  Basic authentication, username:password.

  // HTTP代理地址作, 使用host:port格式。

  -x  HTTP Proxy address as host:port.

  // 启用HTTP / 2

  -h2 Enable HTTP/2.

  // HTTP主机头

  -host HTTP Host header.

  // 禁用压缩。

  -disable-compression  Disable compression.

  // 禁用保持活动状态，防止重新使用不同的HTTP请求之间的TCP连接。

  -disable-keepalive    Disable keep-alive, prevents re-use of TCP

                        connections between different HTTP requests.

  // 禁用HTTP重定向

  -disable-redirects    Disable following of HTTP redirects

  // 使用的cpu内核数。当前计算机的默认值为8核。

  -cpus                 Number of used cpu cores.

                        (default for current machine is 8 cores)

**Windows下命令行**

hey\_windows\_amd64.exe -n 9000 -c 100 -m "POST" -T "application/json" -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1LzA1L2lkZW50aXR5L2NsYWltcy9uYW1laWRlbnRpZmllciI6IjEzNzI1MTUyNTI1Iiwic3ViIjoiMTM3MjUxNTI1MjUiLCJqdGkiOiIxZTU2MzlhNS02YjQzLTRmNTctYWFkYi0yN2UyNzcyOTliNTgiLCJpYXQiOjE1ODM3OTg0NzAsIm5iZiI6MTU4Mzc5ODQ3MCwiZXhwIjoxNTgzODg0ODcwLCJpc3MiOiJHREg1IiwiYXVkIjoiR0RINSJ9.n5sUWANx28LXmvipMMl0cAtorMCei5NxJXUDEnHcd7E"

<http://localhost:8881/api/services/app/GZGoddessFestival/TurnOn>

**hey实战**

对于多个实例的应用，如并发锁。 实际业务场景中分布式锁。

**nginx配置**

upstream backend  {

       server 127.0.0.1:5000;

        server 127.0.0.1:6001;

    }

    server {

        listen       8881;

        server\_name  localhost;

        #charset koi8-r;

        #access\_log  logs/host.access.log  main;

        location / {

          proxy\_pass  http://backend;

        }

}

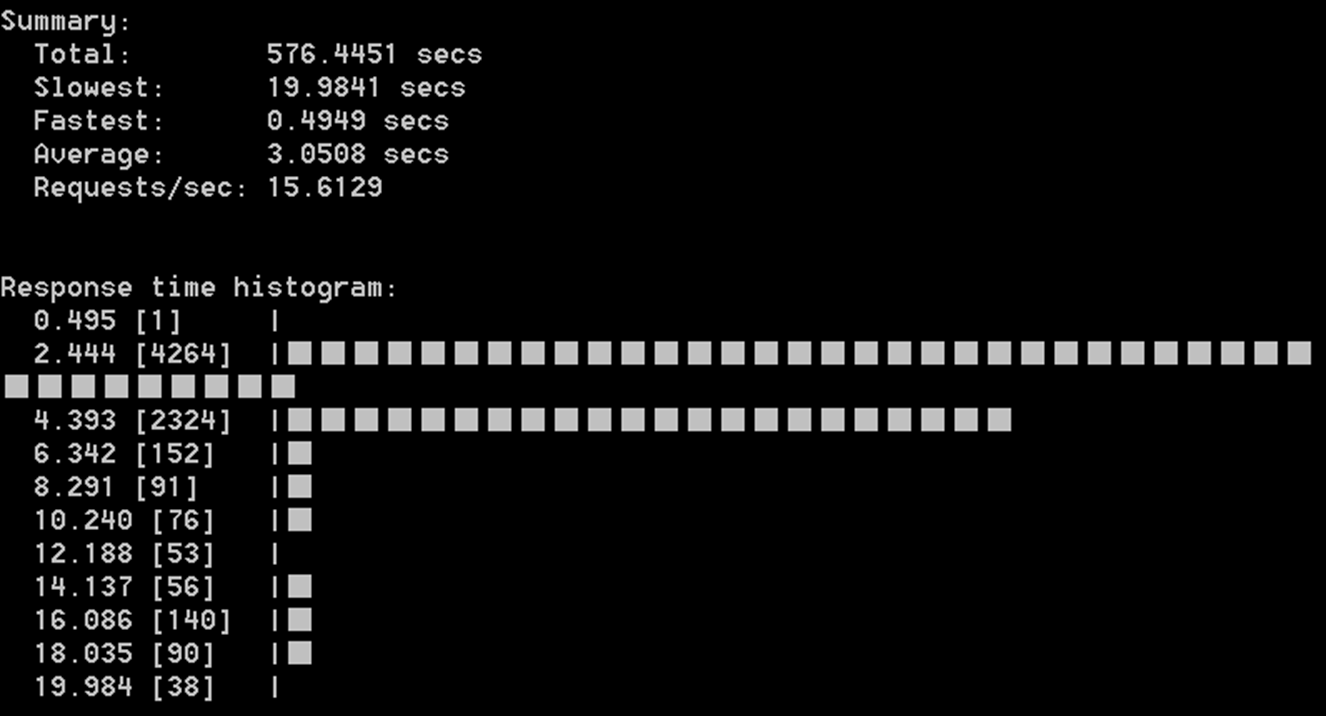
我们分别在本地部署两个相同版本的实例，分别端口是5000,6001，如上是我们反向代理的配置。对外监听的8881端口。

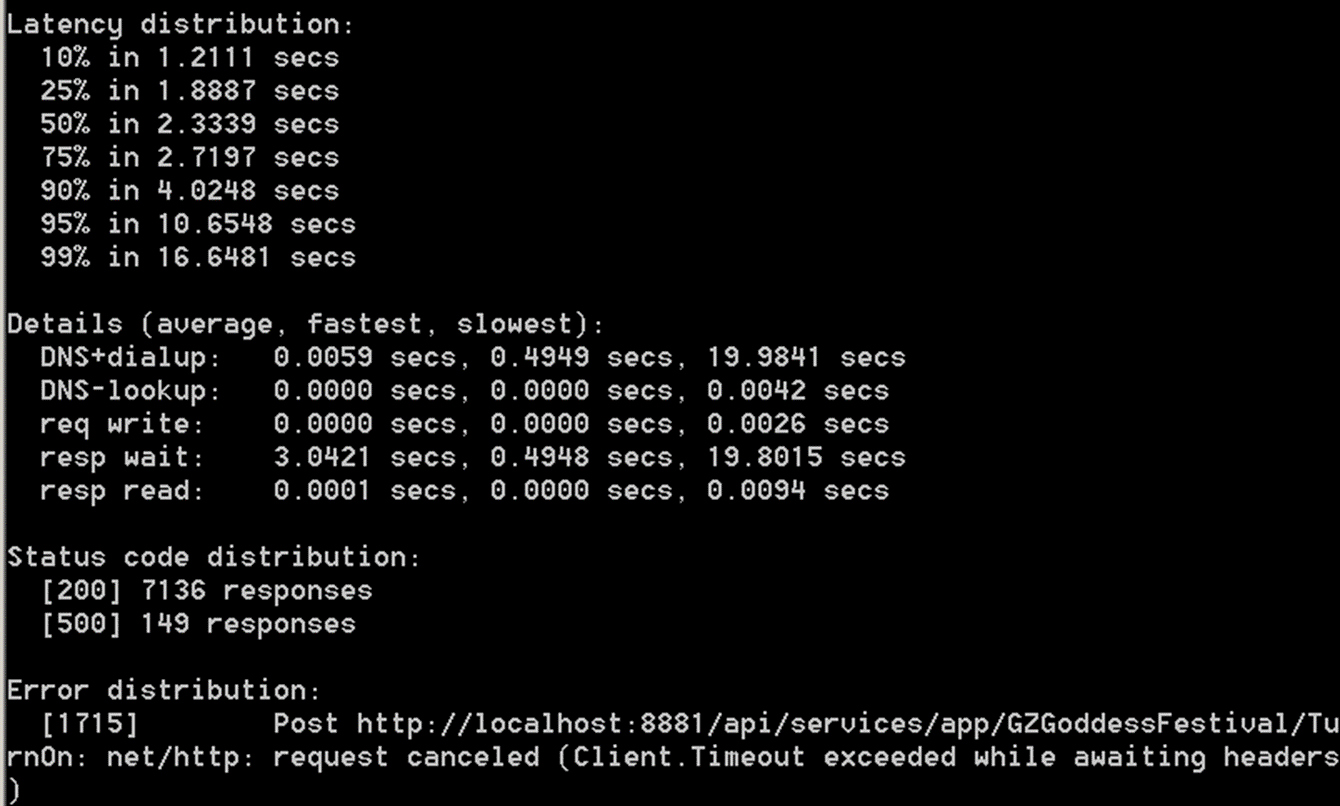
2个实例本地启动，dotnetcore应用默认是5000端口

dotnet Api.Web.Host.dll --urls=<http://localhost:6001/>

dotnet Api.Web.Host.dll

我们获取对应接口的URL，准备负载测试如下，9000请求，100并发， HTTP POST，自定义HTTP HEADER 用于认证：

[](https://img2018.cnblogs.com/blog/15172/202003/15172-20200314143852267-1695856471.gif)[](https://img2018.cnblogs.com/blog/15172/202003/15172-20200314143853388-1197032448.gif)

[](https://img2018.cnblogs.com/blog/15172/202003/15172-20200314143854236-1953406415.gif)

以上是简单的测试摘要，HTTP STATUS 200 共计 7136个，HTTP 500 149个，最慢的请求19.8秒

**Nginx的日志**

6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.132 Safari/537.36"

127.0.0.1 - - [10/Mar/2020:07:56:45 +0800] "POST /api/services/app/Ability/TestRsa HTTP/1.1" 200 455 "http://localhost:8881/swagger/index.html" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.132 Safari/537.36"

127.0.0.1 - - [10/Mar/2020:07:56:48 +0800] "POST /api/services/app/Ability/TestRsa HTTP/1.1" 200 455 "http://localhost:8881/swagger/index.html" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.132 Safari/537.36"

127.0.0.1 - - [10/Mar/2020:07:58:09 +0800] "GET /api/services/app/GZGoddessFestival/TurnOn HTTP/1.1" 404 0 "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.132 Safari/537.36"

127.0.0.1 - - [10/Mar/2020:07:59:06 +0800] "POST /api/services/app/GZGoddessFestival/TurnOn HTTP/1.1" 401 206 "-" "Fiddler"

127.0.0.1 - - [10/Mar/2020:07:59:16 +0800] "POST /api/services/app/GZGoddessFestival/TurnOn HTTP/1.1" 401 206 "-" "hey/0.0.1"

127.0.0.1 - - [10/Mar/2020:07:59:16 +0800] "POST /api/services/app/GZGoddessFestival/TurnOn HTTP/1.1" 401 206 "-" "hey/0.0.1"

127.0.0.1 - - [10/Mar/2020:07:59:16 +0800] "POST /api/services/app/GZGoddessFestival/TurnOn HTTP/1.1" 401 206 "-" "hey/0.0.1"

127.0.0.1 - - [10/Mar/2020:07:59:16 +0800] "POST /api/services/app/GZGoddessFestival/TurnOn HTTP/1.1" 401 206 "-" "hey/0.0.1"

以上日志中我们也看到hey的请求，TurnOn这个接口是用于游戏抽奖，我们使用简单的工具实现负载测试，用于检测程序接口在并发的TPS，同时验证接口是否有奖品超发的情况。

**总结**

实际项目中，现在可能是微服务的，本地的微服务程序需要进行依赖隔离。上面演示的仅仅是单体应用。 Hey做为一个轻量级负载工具，同时也支持扩平台。Hey有开源，对于Go语言也是学习的案例。

但对于复杂场景的负载测试需要输出详尽报告则不太适用。