第四课第一题

题目：**1.（选做）**运行课上的例子，以及 Netty 的例子，分析相关现象。

运行HttpServer1， HttpServer2， HttpServer3， 三个server都不在service中休眠

HttpServer1为单线程处理请求， 压测结果：

```
PS C:\Users\xiaowenhou> sb -u http://localhost:8801 -c 50 -N 30
Starting at 2021/1/19 22:17:22
[Press C to stop the test]
---------------Finished!----------------
Finished at 2021/1/19 22:17:57 (took 00:00:34.5886329)
Status 200:    106372
Status 303:    13552

RPS: 3844.9 (requests/second)
Min: 0ms
Avg: 2ms

  50%   below 0ms
  60%   below 0ms
  70%   below 1ms
  80%   below 3ms
  90%   below 6ms
  95%   below 10ms
  98%   below 15ms
  99%   below 20ms
99.9%   below 39ms
```

HttpServer2为来一个请求创建一个线程，压测结果：

```
PS C:\Users\xiaowenhou> sb -u http://localhost:8802 -c 50 -N 30
Starting at 2021/1/19 22:19:51
[Press C to stop the test]
---------------Finished!----------------
Finished at 2021/1/19 22:20:25 (took 00:00:34.5291831)
Status 200:    108216
Status 303:    1826

RPS: 3536.3 (requests/second)
Max: 88ms
Min: 0ms
Avg: 2.9ms

  50%   below 1ms
  60%   below 2ms
  70%   below 2ms
  80%   below 5ms
  90%   below 8ms
  95%   below 12ms
  98%   below 18ms
  99%   below 23ms
99.9%   below 43ms
```

由于service方法中没有休眠，请求进来之后能够很快返回，这时候**创建线程以及线程间切换的开销已经大于程序执行的开销， 所以在这种情况下，HttpServer2的吞吐量还不如单线程响应请求的HttpServer1**.

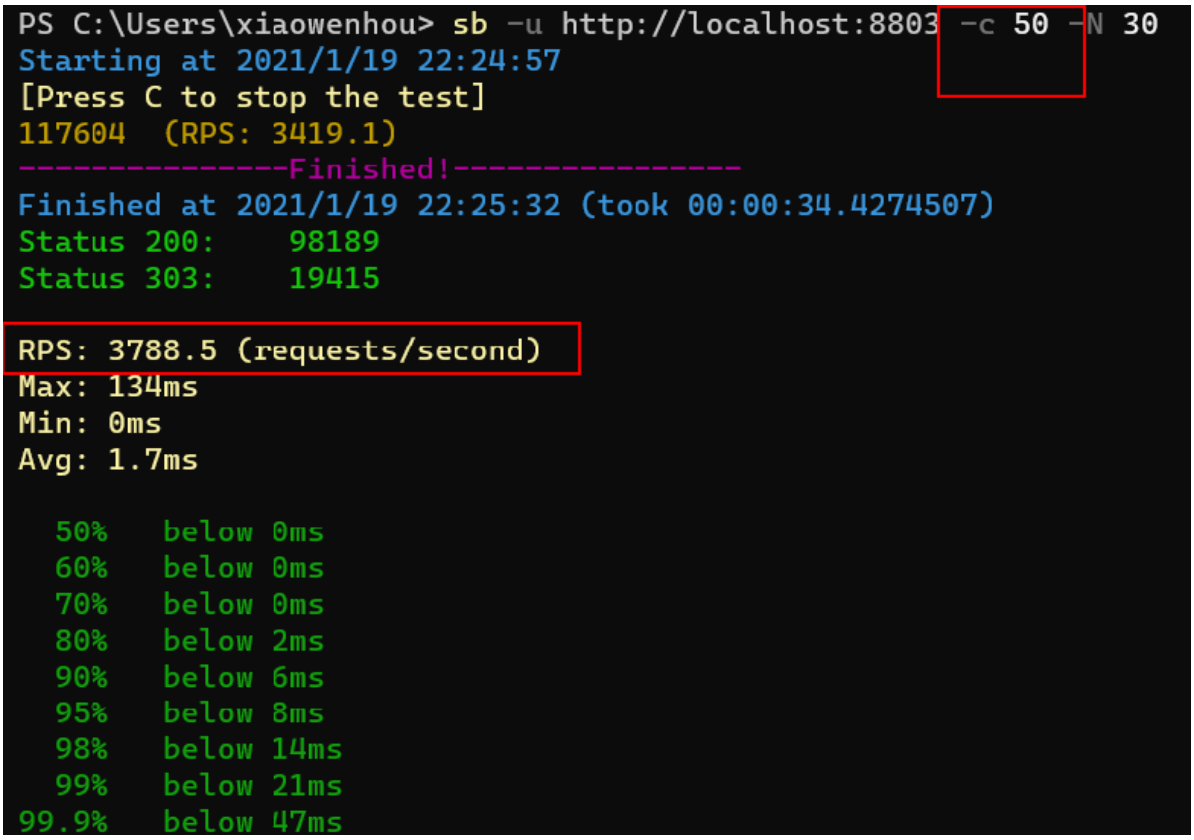
HttpServer3采用线程池：线程池中线程的数量为可用的核心数 + 2，避免线程数过大， 引起线程切换频繁，影响性能

代码：

```
    public static void main(String[] args) throws IOException{

        ExecutorService executorService = Executors.newFixedThreadPool(
                Runtime.getRuntime().availableProcessors() + 2); //线程数为核心数 +
2
        final ServerSocket serverSocket = new ServerSocket(8803);
        while (true) {
            try {
                final Socket socket = serverSocket.accept();
                executorService.execute(() -> service(socket));
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
```

压测结果：



经过多次测试， 吞吐量相比HttpServer2要好一些， 但还是比不上HttpServer1，猜测**虽然创建线程的开销小了很多， 但是将请求放在等待队列， 然后线程执行完当前任务后，从任务队列中获取任务执行等过程消耗了一定的性能， 因为线程真正要执行的任务很简单很快，因此复杂的设计和操作所消耗的性能反而不如最简单的一个线程响应来的高效。**

Netty服务器压测结果

```
PS C:\Users\xiaowenhou> sb -u http://localhost:8808/test -c 50 -N 30
Starting at 2021/1/19 22:21:48
[Press C to stop the test]
197202  (RPS: 5721.5)
---------------Finished!----------------
Finished at 2021/1/19 22:22:23 (took 00:00:34.6898187)
Status 200:    197203

RPS: 6317.5 (requests/second)
Max:  116ms
Min:  0ms
Avg:  0.2ms

  50%   below 0ms
  60%   below 0ms
  70%   below 0ms
  80%   below 0ms
  90%   below 0ms
  95%   below 0ms
  98%   below 5ms
  99%   below 6ms
99.9%   below 14ms
```

Netty服务器的压测结果就很惊艳了， 完全碾压其他的几个服务器， **原因在于之前几个服务器，不管是单线程， 多线程， 还是线程池， 本质上都是BIO，当内核在准备数据的时候线程都要阻塞等待， 而Netty则采用了NIO，多个请求的线程在Netty中注册事件，Netty内部有select在关注这些事件，当内核准备好数据以后，就通知该请求的线程取数据，通过这种机制，可以连接到Server上的请求更多，因此吞吐量也更大。**

接下来测试在service和Handler中休眠20ms的情况（模拟IO密集型）

HttpServer1压测结果：

```
PS C:\Users\xiaowenhou> sb -u http://localhost:8801 -c 50 -N 30
Starting at 2021/1/19 22:48:12
[Press C to stop the test]
972     (RPS: 28.1)
---------------Finished!----------------
Finished at 2021/1/19 22:48:47 (took 00:00:34.6432304)
1005    (RPS: 29.9)                     Status 200:    1005
RPS: 32.2 (requests/second)
Max:  1591ms
Min:  78ms
Avg:  1514.5ms

  50%   below 1550ms
  60%   below 1551ms
  70%   below 1554ms
  80%   below 1555ms
  90%   below 1557ms
  95%   below 1559ms
  98%   below 1560ms
  99%   below 1561ms
99.9%   below 1576ms
1016    (RPS: 29.3)
```

HttpServer2压测结果：

```
PS C:\Users\xiaowenhou> sb -u http://localhost:8802 -c 50 -N 30
Starting at 2021/1/19 22:49:06
[Press C to stop the test]
44810   (RPS: 1299.3)
---------------Finished!----------------
Finished at 2021/1/19 22:49:40 (took 00:00:34.5398788)
Status 200:    44179

RPS: 1441.7 (requests/second)
Max: 145ms
Min: 20ms
Avg: 29.4ms

  50%   below 28ms
  60%   below 28ms
  70%   below 29ms
  80%   below 30ms
  90%   below 33ms
  95%   below 56ms
  98%   below 59ms
  99%   below 61ms
99.9%   below 81ms
```

HttpServer3压测结果：

```
PS C:\Users\xiaowenhou> sb -u http://localhost:8803 -c 50 -N 30
Starting at 2021/1/19 22:49:46
[Press C to stop the test]
27679   (RPS: 804.4)
27691   (RPS: 804.5)                    ---------------Finished!----------------
27687   (RPS: 804.4)                    Finished at 2021/1/19 22:50:20 (took 00:00:34.4434639)
RPS: 892.1 (requests/second)
Max: 149ms
Min: 23ms
Avg: 51ms

  50%   below 45ms
  60%   below 51ms
  70%   below 57ms
  80%   below 58ms
  90%   below 60ms
  95%   below 63ms
  98%   below 92ms
  99%   below 104ms
99.9%   below 109ms
```

可见，**当模拟IO密集型时， HttpServer1的表现最差，因为只有一个线程， 该线程阻塞在等待IO上的时候， 其他请求也必须等待，此时CPU使用率极低，导致吞吐量也非常低，而由于每个请求阻塞在IO上的时间比较长， 因此在一定量的请求并发下， 额外的多创建一些线程反而能够获得比较不错的吞吐量，因为当某个线程阻塞在IO上时， 此时CPU能够切换到其他线程上执行，创建线程和切换线程的开销都被线程阻塞在IO上的时间给覆盖掉了， 多一些线程反而能够提高CPU的利用率，所以此时HttpServer2的表现要优于HttpServer3的表现。**

默认配置下Netty压测结果：

```
PS C:\Users\xiaowenhou> sb -u http://localhost:8808/test -c 50 -N 30
Starting at 2021/1/19 23:00:49
[Press C to stop the test]
15409   (RPS: 446)1)
---------------Finished!----------------
Finished at 2021/1/19 23:01:23 (took 00:00:34.6473188)
Status 200:    15410
RPS: 494.2 (requests/second)
Max: 277ms
Min: 51ms
Avg: 93.9ms

  50%   below 91ms
  60%   below 119ms
  70%   below 120ms
  80%   below 121ms
  90%   below 123ms
  95%   below 126ms
  98%   below 183ms
  99%   below 185ms
99.9%   below 226ms
```

再来看看Netty的表现，默认配置下的表现就差强人意了，吞吐量高于HttpServer1， 但是小于HttpServer2和HttpServer3，此时的配置如下：

```
    EventLoopGroup bossGroup = new NioEventLoopGroup(2);
    EventLoopGroup workerGroup = new NioEventLoopGroup(16);
```

因为Netty可用的线程在workerGroup中配置，对于IO密集型的任务，16个线程太少了，CPU存在大量空闲，没有发挥出性能。

修改Netty的workerGroup参数， 调大线程数后的压测结果：

```
PS C:\Users\xiaowenhou> sb -u http://localhost:8808/test -c 50 -N 30
Starting at 2021/1/19 23:15:57
[Press C to stop the test]
47911   (RPS: 1387.7)
---------------Finished!----------------
Finished at 2021/1/19 23:16:31 (took 00:00:34.6578572)
Status 200:    47918

RPS: 1538.2 (requests/second)
Max: 206ms
Min: 19ms
Avg: 26.6ms

  50%   below 27ms
  60%   below 27ms
  70%   below 28ms
  80%   below 28ms
  90%   below 29ms
  95%   below 30ms
  98%   below 32ms
  99%   below 33ms
99.9%   below 161ms
```

效果立竿见影，修改后的配置如下：

```
   EventLoopGroup bossGroup = new NioEventLoopGroup(2);
   EventLoopGroup workerGroup = new NioEventLoopGroup(50);
```

可见，**增大workerGroup中的线程数，就能立刻提升吞吐量，而且此时Netty的吞吐量依然是最好的，这也说明NIO相比BIO，不论在什么场景下，性能都有显著的优势。**