第2周 3. 各种网络IO模型的压测分析

笔记本： JVM进阶
创建时间： 2020/10/28 9:26 **更新时间：** 2020/10/28 9:40
作者： holybell@vip.qq.com
URL： about:blank

1.单线程处理Soket监听以及业务

```
public class HttpServer01 {
    public static void main(String[] args) throws IOException{
        ServerSocket serverSocket = new ServerSocket(8801);
        while (true) {
            try {
                Socket socket = serverSocket.accept();
                service(socket);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    private static void service(Socket socket) {
        try {
            Thread.sleep(20);
            PrintWriter printWriter = new PrintWriter(socket.getOutputStream(),
true);
            printWriter.println("HTTP/1.1 200 OK");
            printWriter.println("Content-Type:text/html;charset=utf-8");
            printWriter.println();
            printWriter.write("hello,nio");
            printWriter.close();
            socket.close();
        } catch (IOException | InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

压测结果：

```
2490    (RPS: 32.6)
---------------Finished!----------------
Finished at 2020/10/26 13:39:56 (took 00:01:16.7873920)
2505    (RPS: 32.8)                     Status 200:    2505

RPS: 40.9 (requests/second)
Max: 1627ms
Min: 40ms
Avg: 470ms

  50%   below 439ms
  60%   below 450ms
  70%   below 473ms
  80%   below 502ms
  90%   below 567ms
  95%   below 634ms
  98%   below 760ms
  99%   below 835ms
```

```
99.9%   below 1546ms
```

结论：由于仅仅使用一个线程处理socket的监听同时处理业务，导致大量并发请求阻塞在等候被accept的阶段，造成了吞吐量低下，延迟很高的情况。

2.利用现代多核CPU的优势，异步处理业务

```
public class HttpServer02 {
    public static void main(String[] args) throws IOException{
        ServerSocket serverSocket = new ServerSocket(8802);
        while (true) {
            try {
                final Socket socket = serverSocket.accept();
                new Thread(() -> {
                    service(socket);
                }).start();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    private static void service(Socket socket) {
        try {
            Thread.sleep(20);
            PrintWriter printWriter = new PrintWriter(socket.getOutputStream(),
true);
            printWriter.println("HTTP/1.1 200 OK");
            printWriter.println("Content-Type:text/html;charset=utf-8");
            printWriter.println();
            printWriter.write("hello,nio");
            printWriter.close();
            socket.close();
        } catch (IOException | InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

压测结果：

```
28890   (RPS: 413.2)
---------------Finished!----------------
Finished at 2020/10/26 13:42:58 (took 00:01:10.2760196)
Status 200:    28890
Status 303:    4

RPS: 472.8 (requests/second)
Max: 333ms
Min: 20ms
Avg: 31.2ms

  50%   below 26ms
  60%   below 28ms
  70%   below 31ms
  80%   below 35ms
  90%   below 45ms
  95%   below 59ms
  98%   below 83ms
  99%   below 108ms
99.9%   below 198ms
```

结论：每次accept一个socket，立即创建一个线程异步处理业务，解决了上一个代码中并发请求阻塞在被accept导致延迟高，吞吐量低的问题，但是大量的请求将会导致频繁创建、销毁线程，线程切换将会造成大量的计算资源浪费。

3.使用线程池解决频繁创建销毁线程开销

```
public class HttpServer03 {
    public static void main(String[] args) throws
IOException{
        ExecutorService executorService =
Executors.newFixedThreadPool(40);
        final ServerSocket serverSocket = new
ServerSocket(8803);
        while (true) {
            try {
                final Socket socket =
serverSocket.accept();
                executorService.execute(() ->
service(socket));
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    private static void service(Socket socket) {
        try {
            Thread.sleep(20);
            PrintWriter printWriter = new
PrintWriter(socket.getOutputStream(), true);
            printWriter.println("HTTP/1.1 200 OK");
            printWriter.println("Content-
Type:text/html;charset=utf-8");
            printWriter.println();
            printWriter.write("hello,nio");
            printWriter.close();
            socket.close();
        } catch (IOException | InterruptedException
e) {
            e.printStackTrace();
        }
    }
}
```

压测结果：

```
31626   (RPS: 461.9)
---------------Finished!----------------
Finished at 2020/10/26 13:44:51 (took
00:01:08.7659332)
Status 200:    31616
Status 303:    10

RPS: 516.5 (requests/second)
Max: 337ms
Min: 19ms
Avg: 26.4ms

  50%   below 22ms
  60%   below 23ms
  70%   below 25ms
  80%   below 28ms
  90%   below 35ms
  95%   below 45ms
  98%   below 66ms
  99%   below 87ms
99.9%   below 199ms
```

结论：由于使用线程池，解决了过多的线程导致无谓的线程切换开销，无法及时处理的任务进入线程池任务队列等待处理，进一步提升了吞吐量

4.使用Netty的reactor网络I/O模型

```
public class HttpServer {
    private static Logger logger = LoggerFactory.getLogger(HttpServer.class);

    private boolean ssl;
    private int port;

    public HttpServer(boolean ssl,int port) {
        this.port=port;
        this.ssl=ssl;
    }

    public void run() throws Exception {
        final SslContext sslCtx;
        if (ssl) {
            SelfSignedCertificate ssc = new SelfSignedCertificate();
            sslCtx = SslContext.newServerContext(ssc.certificate(),
ssc.privateKey());
        } else {
            sslCtx = null;
        }

        EventLoopGroup bossGroup = new NioEventLoopGroup(3);
        EventLoopGroup workerGroup = new NioEventLoopGroup(1000);
```

```
        try {
            ServerBootstrap b = new ServerBootstrap();
            b.option(ChannelOption.SO_BACKLOG, 128)             // 网络调优参数配置
                    .option(ChannelOption.TCP_NODELAY, true)
                    .option(ChannelOption.SO_KEEPALIVE, true)
                    .option(ChannelOption.SO_REUSEADDR, true)
                    .option(ChannelOption.SO_RCVBUF, 32 * 1024)
                    .option(ChannelOption.SO_SNDBUF, 32 * 1024)
                    .option(EpollChannelOption.SO_REUSEPORT, true)
                    .childOption(ChannelOption.SO_KEEPALIVE, true);
                    //.option(ChannelOption.ALLOCATOR,
PooledByteBufAllocator.DEFAULT);

            b.group(bossGroup, workerGroup).channel(NioServerSocketChannel.class)
                    .handler(new LoggingHandler(LogLevel.INFO)).childHandler(new
HttpInitializer(sslCtx));    // 将具体的业务操作封装到handler中

            Channel ch = b.bind(port).sync().channel();
            logger.info("开启netty http服务器，监听地址和端口为 " + (ssl ? "https" :
"http") + "://127.0.0.1:" + port + '/');
            ch.closeFuture().sync();
        } finally {
            bossGroup.shutdownGracefully();
            workerGroup.shutdownGracefully();
        }
    }
}
```

压测结果：

```
62349   (RPS: 907.5)
---------------Finished!----------------
Finished at 2020/10/26 13:54:57 (took
00:01:09.0939519)
Status 200:    62349

RPS: 1017.5 (requests/second)
Max: 469ms
Min: 0ms
Avg: 4.2ms

  50%   below 3ms
  60%   below 4ms
  70%   below 5ms
  80%   below 6ms
  90%   below 9ms
  95%   below 12ms
  98%   below 17ms
  99%   below 22ms
99.9%   below 55ms
```