zedosu

昵称：zedosu
园龄：1年5个月
粉丝：16
关注：0
+加关注

< 2018年8月 >

日 一 二 三 四 五 六
29 30 31 1 2 3 4
5 6 7 8 9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31 1
2 3 4 5 6 7 8

搜索

找找看
谷歌搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签
更多链接

我的标签

关于**BIO**和**NIO**的理解

摘要: 关于BIO和NIO的理解

最近大概看了ZooKeeper和Mina的源码发现都是用Java NIO实现的，所以有必要搞清楚什么是NIO。下面是我结合网络资料自己总结的，为了节约时间图示随便画的，能达意就行。

简介：

BIO：同步阻塞式IO，服务器实现模式为一个连接一个线程，即客户端有连接请求时服务器端就需要启动一个线程进行处理，如果这个连接不做任何事情会造成不必要的线程开销，当然可以通过线程池机制改善。
NIO：同步非阻塞式IO，服务器实现模式为一个请求一个线程，即客户端发送的连接请求都会注册到多路复用器上，多路复用器轮询到连接有I/O请求时才启动一个线程进行处理。
AIO(NIO.2)：异步非阻塞式IO，服务器实现模式为一个有效请求一个线程，客户端的I/O请求都是由OS先完成了再通知服务器应用去启动线程进行处理。

BIO

同步阻塞式IO，相信每一个学习过操作系统网络编程或者任何语言的网络编程的人都很熟悉，在while循环中服务端会调用accept方法等待接收客户端的连接请求，一旦接收到一个连接请求，就可以建立通信套接字在这个通信套接字上进行读写操作，此时不能再接收其他客户端连接请求，只能等待同当前连接的客户端的操作执行完成。
如果BIO要能够同时处理多个客户端请求，就必须使用多线程，即每次accept阻塞等待来自客户端请求，一旦受到连接请求就建立通信套接字同时开启一个新的线程来处理这个套接字的数据读写请求，然后立

java(61)
Java多线程(19)
JavaWeb(17)
计算机网络(10)
数据库(9)
JVM(8)
数据结构与算法(7)
Spring(6)
文章(5)
mysql(4)
更多

随笔档案(121)

2017年4月 (55)
2017年3月 (66)

最新评论

1. Re:关于BIO和NIO的理解
好.支持.收藏了.
--jsdsh
2. Re:多线程在javaweb中的应用
@
2
--一路上友你
3. Re:多线程在javaweb中的应用
1
--一路上友你

阅读排行榜

1. mysql 如何优化left join(11199)
2. Java集合类框架的基本接口有哪些？(10549)
3. 关于BIO和NIO的理解(8802)
4. 多线程在javaweb中的应用(5164)
5. ThreadPoolExecutor使用详解(4134)

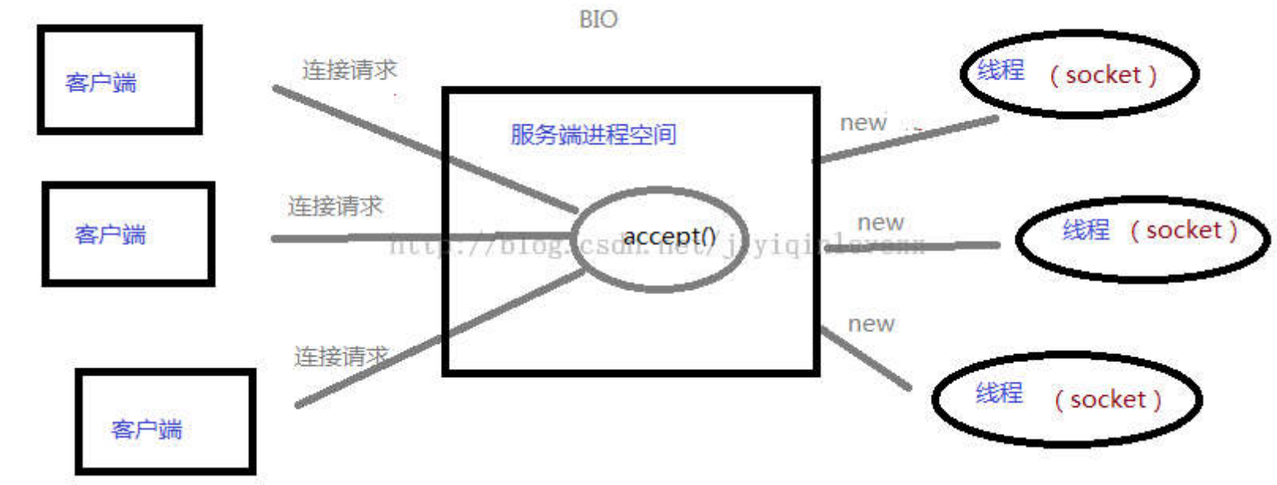
评论排行榜

1. 多线程在javaweb中的应用(2)
2. 关于BIO和NIO的理解(1)

推荐排行榜

1. java垃圾回收 - 为什么要进行垃圾回收(1)

刻又继续accept等待其他客户端连接请求，即为每一个客户端连接请求都创建一个线程来单独处理，大概原理图就像这样：



虽然此时服务器具备了高并发能力，即能够同时处理多个客户端请求了，但是却带来了一个问题，随着开启的线程数目增多，将会消耗过多的内存资源，导致服务器变慢甚至崩溃，NIO可以一定程度解决这个问题。

NIO

同步非阻塞式IO，关键是采用了事件驱动的思想来实现了一个多路转换器。

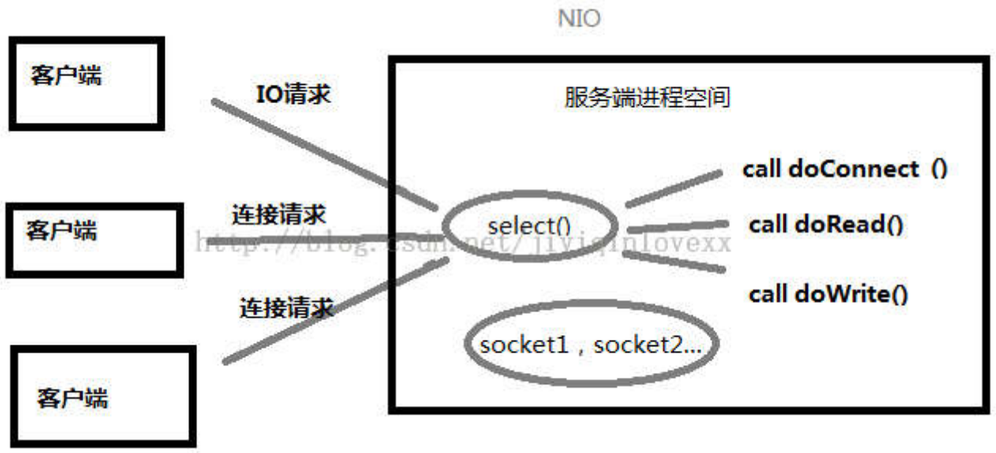
NIO与BIO最大的区别就是只需要开启一个线程就可以处理来自多个客户端的IO事件，这是怎么做到的呢？

就是多路复用器，可以监听来自多个客户端的IO事件：

A. 若服务端监听到客户端连接请求，便为其建立通信套接字(java中就是通道)，然后返回继续监听，若同时有多个客户端连接请求到来也可以全部收到，依次为它们都建立通信套接字。

B. 若服务端监听到来自已经创建了通信套接字的客户端发送来的数据，就会调用对应接口处理接收到的数据，若同时有多个客户端发来数据也可以依次进行处理。

C. 监听多个客户端的连接请求和接收数据请求同时还能监听自己时候有数据要发送。



总之就是在一个线程中就可以调用多路复用接口（java中是select）阻塞同时监听来自多个客户端的IO请求，一旦有收到IO请求就调用对应函数处理。

各自应用场景

到这里你也许已经发现，一旦有请求到来(不管是几个同时到还是只有一个到)，都会调用对应IO处理函数处理，所以：

（1）NIO适合处理连接数目特别多，但是连接比较短（轻操作）的场景，Jetty，Mina，ZooKeeper等都是基于java nio实现。

（2）BIO方式适用于连接数目比较小且固定的场景，这种方式对服务器资源要求比较高，并发局限于应用中。

附录：下面附上一个别人写的**java NIO**的例子。

服务端：

```
1. package cn.nio;
2.
3. import java.io.IOException;
4. import java.net.InetSocketAddress;
5. import java.nio.ByteBuffer;
6. import java.nio.channels.SelectionKey;
7. import java.nio.channels.Selector;
8. import java.nio.channels.ServerSocketChannel;
9. import java.nio.channels.SocketChannel;
10. import java.util.Iterator;
11.
12. /**
13. * NIO服务端
14. *
15. */
16. public class NIOServer {
17.    //通道管理器
18.    private Selector selector;
19.
20.    /**
21.     * 获得一个ServerSocket通道，并对该通道做一些初始化的工作
22.     * @param port  绑定的端口号
23.     * @throws IOException
24.     */
25.    public void initServer(int port) throws IOException {
26.        // 获得一个ServerSocket通道
27.        ServerSocketChannel serverChannel = ServerSocketChannel.open();
```

```
28. 28.      // 设置通道为非阻塞
29. 29.      serverChannel.configureBlocking(false);
30. 30.      // 将该通道对应的ServerSocket绑定到port端口
31. 31.      serverChannel.socket().bind(new InetSocketAddress(port));
32. 32.      // 获得一个通道管理器
33. 33.      this.selector = Selector.open();
34. 34.      //将通道管理器和该通道绑定，并为该通道注册SelectionKey.OP_ACCEPT事件,注册该事件后，
35. 35.      //当该事件到达时，selector.select()会返回，如果该事件没到达selector.select()会一直阻塞。
36. 36.      serverChannel.register(selector, SelectionKey.OP_ACCEPT);
37. 37.  }
38. 38.
39. 39.  /**
40. 40.   * 采用轮询的方式监听selector上是否有需要处理的事件，如果有，则进行处理
41. 41.   * @throws IOException
42. 42.   */
43. 43.  @SuppressWarnings("unchecked")
44. 44.  public void listen() throws IOException {
45. 45.      System.out.println("服务端启动成功！");
46. 46.      // 轮询访问selector
47. 47.      while (true) {
48. 48.          //当注册的事件到达时，方法返回；否则,该方法会一直阻塞
49. 49.          selector.select();
50. 50.          // 获得selector中选中的项的迭代器，选中的项为注册的事件
51. 51.          Iterator ite = this.selector.selectedKeys().iterator();
52. 52.          while (ite.hasNext()) {
53. 53.              SelectionKey key = (SelectionKey) ite.next();
54. 54.              // 删除已选的key,以防重复处理
55. 55.              ite.remove();
56. 56.              // 客户端请求连接事件
57. 57.              if (key.isAcceptable()) {
58. 58.                  ServerSocketChannel server = (ServerSocketChannel) key
```

```
59. 59.                 .channel();
60. 60.             // 获得和客户端连接的通道
61. 61.             SocketChannel channel = server.accept();
62. 62.             // 设置成非阻塞
63. 63.             channel.configureBlocking(false);
64. 64.
65. 65.             //在这里可以给客户端发送信息哦
66. 66.             channel.write(ByteBuffer.wrap(new String("向客户端发送了一条信息").getBytes()));
67. 67.             //在和客户端连接成功之后，为了可以接收到客户端的信息，需要给通道设置读的权限。
68. 68.             channel.register(this.selector, SelectionKey.OP_READ);
69. 69.
70. 70.             // 获得了可读的事件
71. 71.         } else if (key.isReadable()) {
72. 72.                 read(key);
73. 73.         }
74. 74.
75. 75.         }
76. 76.
77. 77.     }
78. 78. }
79. 79. /**
80. 80.  * 处理读取客户端发来的信息 的事件
81. 81.  * @param key
82. 82.  * @throws IOException
83. 83.  */
84. 84. public void read(SelectionKey key) throws IOException{
85. 85.     // 服务器可读取消息:得到事件发生的Socket通道
86. 86.     SocketChannel channel = (SocketChannel) key.channel();
87. 87.     // 创建读取的缓冲区
88. 88.     ByteBuffer buffer = ByteBuffer.allocate(10);
89. 89.     channel.read(buffer);
```

```
90. 90.         byte[] data = buffer.array();
91. 91.         String msg = new String(data).trim();
92. 92.         System.out.println("服务端收到信息："+msg);
93. 93.         ByteBuffer outBuffer = ByteBuffer.wrap(msg.getBytes());
94. 94.         channel.write(outBuffer);// 将消息回送给客户端
95. 95.     }
96. 96.
97. 97.     /**
98. 98.      * 启动服务端测试
99. 99.      * @throws IOException
100. 100.     */
101. 101.    public static void main(String[] args) throws IOException {
102. 102.        NIOServer server = new NIOServer();
103. 103.        server.initServer(8000);
104. 104.        server.listen();
105. 105.    }
106. 106.
107. 107. }
```

客户端：

```
1.  1. package cn.nio;
2.  2.
3.  3. import java.io.IOException;
4.  4. import java.net.InetSocketAddress;
5.  5. import java.nio.ByteBuffer;
6.  6. import java.nio.channels.SelectionKey;
7.  7. import java.nio.channels.Selector;
8.  8. import java.nio.channels.SocketChannel;
9.  9. import java.util.Iterator;
10. 10.
11. 11. /**
```

```
12. 12. * NIO客户端
13. 13. *
14. 14. */
15. 15. public class NIOClient {
16. 16.    //通道管理器
17. 17.    private Selector selector;
18. 18.
19. 19.    /**
20. 20.     * 获得一个Socket通道，并对该通道做一些初始化的工作
21. 21.     * @param ip 连接的服务器的ip
22. 22.     * @param port  连接的服务器的端口号
23. 23.     * @throws IOException
24. 24.     */
25. 25.    public void initClient(String ip,int port) throws IOException {
26. 26.        // 获得一个Socket通道
27. 27.        SocketChannel channel = SocketChannel.open();
28. 28.        // 设置通道为非阻塞
29. 29.        channel.configureBlocking(false);
30. 30.        // 获得一个通道管理器
31. 31.        this.selector = Selector.open();
32. 32.
33. 33.        // 客户端连接服务器,其实方法执行并没有实现连接，需要在listen（）方法中调
34. 34.        //用channel.finishConnect();才能完成连接
35. 35.        channel.connect(new InetSocketAddress(ip,port));
36. 36.        //将通道管理器和该通道绑定，并为该通道注册SelectionKey.OP_CONNECT事件。
37. 37.        channel.register(selector, SelectionKey.OP_CONNECT);
38. 38.    }
39. 39.
40. 40.    /**
41. 41.     * 采用轮询的方式监听selector上是否有需要处理的事件，如果有，则进行处理
42. 42.     * @throws IOException
```

```
43. 43.     */
44. 44.    @SuppressWarnings("unchecked")
45. 45.    public void listen() throws IOException {
46. 46.        // 轮询访问selector
47. 47.        while (true) {
48. 48.            selector.select();
49. 49.            // 获得selector中选中的项的迭代器
50. 50.            Iterator ite = this.selector.selectedKeys().iterator();
51. 51.            while (ite.hasNext()) {
52. 52.                SelectionKey key = (SelectionKey) ite.next();
53. 53.                // 删除已选的key,以防重复处理
54. 54.                ite.remove();
55. 55.                // 连接事件发生
56. 56.                if (key.isConnectable()) {
57. 57.                    SocketChannel channel = (SocketChannel) key
58. 58.                            .channel();
59. 59.                // 如果正在连接，则完成连接
60. 60.                if(channel.isConnectionPending()){
61. 61.                    channel.finishConnect();
62. 62.
63. 63.                }
64. 64.                // 设置成非阻塞
65. 65.                channel.configureBlocking(false);
66. 66.
67. 67.                //在这里可以给服务端发送信息哦
68. 68.                channel.write(ByteBuffer.wrap(new String("向服务端发送了一条信息").getBytes()));
69. 69.                //在和服务端连接成功之后，为了可以接收到服务端的信息，需要给通道设置读的权限。
70. 70.                channel.register(this.selector, SelectionKey.OP_READ);
71. 71.
72. 72.                // 获得了可读的事件
73. 73.            } else if (key.isReadable()) {
```

```
74. 74.                 read(key);
75. 75.             }
76. 76.
77. 77.         }
78. 78.
79. 79.     }
80. 80. }
81. 81. /**
82. 82.  * 处理读取服务端发来的信息 的事件
83. 83.  * @param key
84. 84.  * @throws IOException
85. 85.  */
86. 86. public void read(SelectionKey key) throws IOException{
87. 87.     //和服务端的read方法一样
88. 88. }
89. 89.
90. 90.
91. 91. /**
92. 92.  * 启动客户端测试
93. 93.  * @throws IOException
94. 94.  */
95. 95. public static void main(String[] args) throws IOException {
96. 96.     NIOClient client = new NIOClient();
97. 97.     client.initClient("localhost",8000);
98. 98.     client.listen();
99. 99. }
100. 100.
101. 101. }
102.
```

http://blog.csdn.net/jiyiqinlovexx/article/details/42619097

标签： java， IO

好文要顶 关注我 收藏该文

zedosu
关注 - 0
粉丝 - 16
+加关注

0 0

« 上一篇：JVM调优及参数设置
» 下一篇：Java IO流学习总结 - BIO

posted @ 2017-04-04 23:36 zedosu 阅读(8802) 评论(1) 编辑 收藏

评论

#1楼 2018-04-02 15:11 | jsdsh

好.支持.收藏了.

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 登录 或 注册，访问网站首页。

最新IT新闻：
· 云存储公司Dropbox营收3.39亿美元 但股价下跌近10%
· 消息称亚马逊拟在西雅图总部开设内部诊所
· 谷歌推出视频问答应用Cameos：让明星出面回答问题
· 特斯拉都私有化了，恒大还救得活FF吗？
· 特斯拉私有化疑云 我们找特斯拉股东聊了聊
» 更多新闻...

最新知识库文章：
· 成为一个有目标的学习者
· 历史转折中的“杭派工程师”
· 如何提高代码质量？