javanio开发实例计算机等级考试 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/645/2021_2022_javanio_E5_BC_80_c97_645436.htm 首先了解下所谓的java nio是个什么东西！ 传统的并发型服务器设计是利用阻塞型网络I/O 以多线程的模式来实现的，然而由于系统常常在进行网络读写时处于阻塞状态，会大大影响系统的性能；自Java1. 4 开始引入了NIO(新I/O) API，通过使用非阻塞型I/O，实现流畅的网络读写操作，为开发高性能并发型服务器程序提供了一个很好的解决方案。这就是java nio,首先来看下传统的阻塞型网络I/O的不足. Java 平台传统的I/O 系统都是基于Byte（字节）和Stream（数据流）的，相应的I/O 操作都是阻塞型的，所以服务器程序也采用阻塞型I/O 进行数据的读、写操作。本文以TCP长连接模式来讨论并发型服务器的相关设计，为了实现服务器程序的并发性要求，系统由一个单独的主线程来监听用户发起的连接请求，一直处于阻塞状态；当有用户连接请求到来时，程序都会启一个新的线程来统一处理用户数据的读、写操作。 这种模式的优点是简单、实用、易管理；然而缺点也是显而易见的：由于是为每一个客户端分配一个线程来处理输入、输出数据，其线程与客户机的比例近似为1：1，随着线程数量的不断增加，服务器启动了大量的并发线程，会大大加大系统对线程的管理开销，这将成为吞吐量瓶颈的主要原因；其次由于底层的I/O 操作采用的同步模式，I/O 操作的阻塞管理粒度是以服务于请求的线程为单位的,有可能大量的线程会闲置,处于盲等状态，造成I/O资源利用率不高，影响整个系统的性能。 对于并发型服务器，系统用

在阻塞型I/O 等待和线程间切换的时间远远多于CPU 在内存中处理数据的时间，因此传统的阻塞型I/O 已经成为制约系统性能的瓶颈。Java1.4 版本后推出的NIO 工具包，提供了非阻塞型I/O 的异步输入输出机制，为提高系统的性能提供了可实现的基础机制。 NIO 包及工作原理 针对传统I/O 工作模式的不足，NIO 工具包提出了基于Buffer（缓冲区）、Channel（通道）、Selector（选择器）的新模式；Selector（选择器）、可选择的Channel（通道）和SelectionKey（选择键）配合起来使用，可以实现并发的非阻塞型I/O 能力。 NIO 工具包的成员 Buffer（缓冲器） Buffer 类是一个抽象类，它有7 个子类分别对应于七种基本的数据类型：ByteBuffer、CharBuffer、DoubleBuffer、FloatBuffer、IntBuffer、LongBuffer和ShortBuffer。每一个Buffer对象相当于一个数据容器，可以把它看作内存中的一个大的数组，用来存储和提取所有基本类型(boolean 型除外)的数据。Buffer 类的核心是一块内存区，可以直接对其执行与内存有关的操作，利用操作系统特性和能力提高和改善Java 传统I/O 的性能。 Channel（通道） Channel 被认为是NIO 工具包的一大创新点，是(Buffer)缓冲器和I/O 服务之间的通道，具有双向性，既可以读入也可以写出，可以更高效的传递数据。我们这里主要讨论ServerSocketChannel 和SocketChannel，它们都继承了SelectableChannel，是可选择的通道，分别可以工作在同步和异步两种方式下（这里的可选择不是指可以选择两种工作方式，而是指可以有选择的注册自己感兴趣的事件）。当通道工作在同步方式时，它的功能和编程方法与传统的ServerSocket、Socket 对象相似；当通道工作在异步工作方

式时，进行输入输出处理不必等到输入输出完毕才返回，并且可以将其感兴趣的（如：接受操作、连接操作、读出操作、写入操作）事件注册到Selector 对象上，与Selector 对象协同工作可以更有效率的支持和管理并发的网络套接字连接。Selector（选择器）和SelectionKey（选择键） 各类 Buffer 是数据的容器对象；各类Channel 实现在各类Buffer 与各类I/O 服务间传输数据。Selector 是实现并发型非阻塞I/O 的核心，各种可选择的通道将其感兴趣的事件注册Selector 对象上，Selector 在一个循环中不断轮循监视这各些注册在其上的Socket 通道。 SelectionKey 类则封装了SelectableChannel 对象在Selector 中的注册信息。当Selector 监测到在某个注册的SelectableChannel 上发生了感兴趣的事件时,自动激活产生一个SelectionKey对象,在这个对象中记录了哪一个SelectableChannel 上发生了哪种事件，通过对被激活的SelectionKey 的分析,外界可以知道每个SelectableChannel 发生的具体事件类型,进行相应的处理。 NIO 工作原理 通过上面的讨论，我们可以看出在并发型服务器程序中使用NIO，实际上是通过网络事件驱动模型实现的。我们应用Select 机制，不用为每一个客户端连接新启线程处理，而是将其注册到特定的Selector 对象上，这就可以在单线程中利用Selector 对象管理大量并发的网络连接，更好的利用了系统资源；采用非阻塞I/O 的通信方式，不要求阻塞等待I/O 操作完成即可返回，从而减少了管理I/O 连接导致的系统开销，大幅度提高了系统性能。 当有读或写等任何注册的事件发生时，可以从Selector 中获得相应的SelectionKey ， 从SelectionKey 中可以找到发生的事件和该事件所发生的具体的SelectableChannel，

以获得客户端发送过来的数据。由于在非阻塞网络I/O 中采用了事件触发机制，处理程序可以得到系统的主动通知，从而可以实现底层网络I/O 无阻塞、流畅地读写，而不像在原来的阻塞模式下处理程序需要不断循环等待。使用NIO，可以编写出性能更好、更易扩展的并发型服务器程序。 并发型服务器程序的实现代码应用 NIO 工具包，基于非阻塞网络I/O 设计的并发型服务器程序与以往基于阻塞I/O 的实现程序有很大不同，在使用非阻塞网络I/O 的情况下，程序读取数据和写入数据的时机不是由程序员控制的，而是Selector 决定的。下面便给出基于非阻塞网络I/O 的并发型服务器程序的核心代码片段： import java.io.*； //引入Java.io包 import java.net.*； //引入Java.net包 import java.nio.channels.*； //引入Java.nio.channels包 import java.util.*； //引入Java.util包 public class TestServer implements Runnable { /** * 服务器Channel对象，负责接受用户连接 */ 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com