**Nginx 并发模型**

Nginx 的并发模型是一个多进程并发模型，它的 Master 进程在绑定监听地址端口后 fork 出了多个 Slave 进程共同竞争处理这个服务端套接字接收到的很多客户端连接。

**NIO和传统的IO有什么区别呢？**

IO是面向流的，NIO是面向块（缓冲区）的。

IO面向流的操作一次一个字节地处理数据。一个输入流产生一个字节的数据，一个输出流消费一个字节的数据。，导致了数据的读取和写入效率不佳。

NIO面向块的操作在一步中产生或者消费一个数据块。按块处理数据比按(流式的)字节处理数据要快得多，同时数据读取到一个它稍后处理的缓冲区，需要时可在缓冲区中前后移动。这就增加了处理过程中的灵活性。通俗来说，NIO采取了“预读”的方式，当你读取某一部分数据时，他就会猜测你下一步可能会读取的数据而预先缓冲下来。

**IO是阻塞的，NIO是非阻塞的**

对于传统的IO，当一个线程调用read() 或 write()时，该线程被阻塞，直到有一些数据被读取，或数据完全写入。该线程在此期间不能再干任何事情了。

而对于NIO，使用一个线程发送读取数据请求，没有得到响应之前，线程是空闲的，此时线程可以去执行别的任务，而不是像IO中那样只能等待响应完成。

**NIO和IO适用场景**

NIO是为弥补传统IO的不足而诞生的，但是尺有所短寸有所长，NIO也有缺点，因为NIO是面向缓冲区的操作，每一次的数据处理都是对缓冲区进行的，那么就会有一个问题，在数据处理之前必须要判断缓冲区的数据是否完整或者已经读取完毕，如果没有，假设数据只读取了一部分，那么对不完整的数据处理没有任何意义。所以每次数据处理之前都要检测缓冲区数据。

**那么NIO和IO各适用的场景是什么呢？**

如果需要管理同时打开的成千上万个连接，这些连接每次只是发送少量的数据，例如聊天服务器，这时候用NIO处理数据可能是个很好的选择。

而如果只有少量的连接，而这些连接每次要发送大量的数据，这时候传统的IO更合适。使用哪种处理数据，需要在数据的响应等待时间和检查缓冲区数据的时间上作比较来权衡选择。