Detailed Explanation of the Java IO Model

Guide Java Java IO, Java Basics About 2031 words About 7 minutes

The IO model is truly difficult to understand, requiring a lot of fundamental computer knowledge. Writing this article took quite a while, and I'm eager to share what I know! I hope you'll find it helpful! To prepare this article, I even took a look at the book "UNIX Network Programming," but it was incredibly difficult. Oh my goodness! It's heartbreaking!

My personal ability is limited. If there is anything that needs to be supplemented/improved/modified in the article, please point it out in the comments section. Let's make progress together!

Preface

I/O has always been a knowledge point that many friends find difficult to understand. In this article, I will tell you what I understand about I/O, and I hope it will be helpful to you.

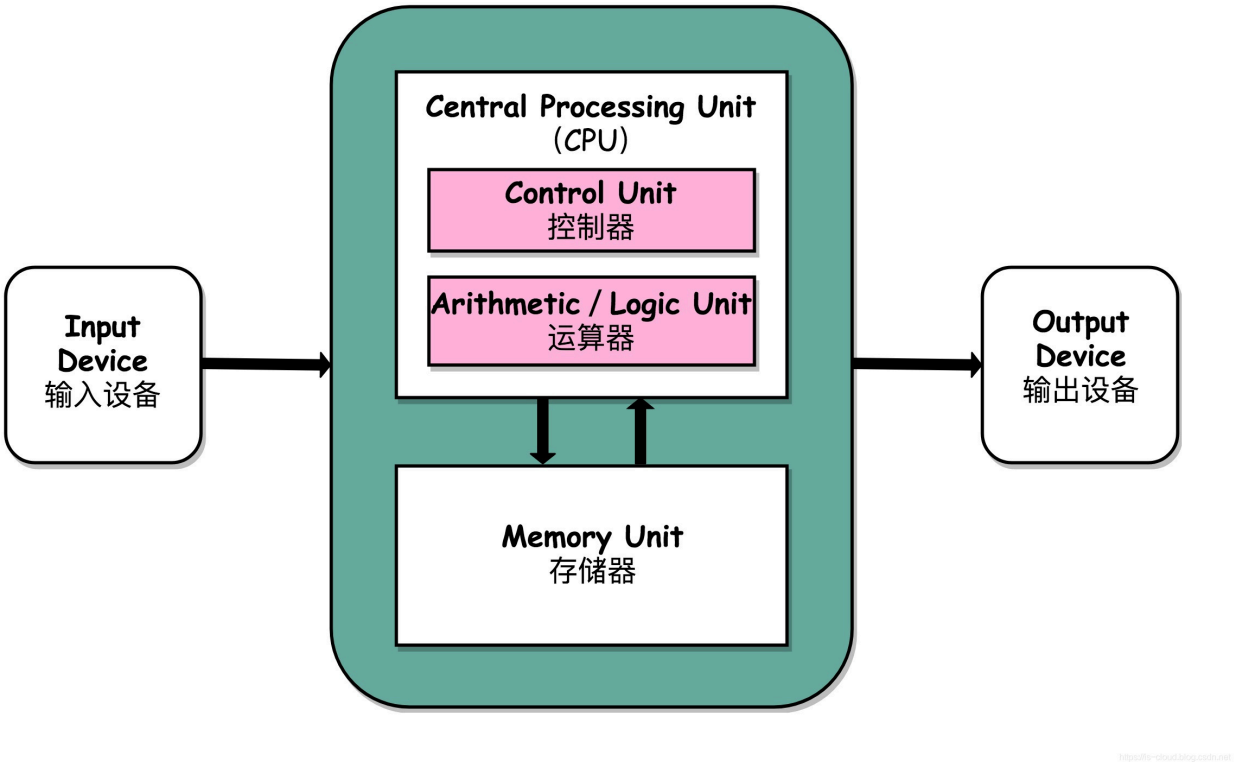
I/O

What is I/O?

I/O (**Input** / **Output**) means **input/output** .

Let's first interpret I/O from the perspective of computer structure.

According to the von Neumann structure, the computer structure is divided into five parts: arithmetic unit, controller, memory, input device, and output device.



Input devices (such as keyboards) and output devices (such as monitors) are both external devices. Network cards and hard drives can be considered both input and output devices.

Input devices input data into the computer, and output devices receive data from the computer.

From the perspective of computer architecture, I/O describes the process of communication between a computer system and external devices.

Let's first interpret I/O from the perspective of the application.

According to the operating system knowledge learned in college: In order to ensure the stability and security of the operating system, the address space of a process is divided into **user space** and **kernel space** .

The applications we normally run all run in user space. Only kernel space can perform system-level resource-related operations, such as file management, process communication, memory management, etc. In other words, if we want to perform IO operations, we must rely on the capabilities of kernel space.

Furthermore, user space programs cannot directly access kernel space.

When you want to perform IO operations, since you do not have the permission to perform these operations, you can only initiate a system call to request the operating system to help you complete it.

Therefore, if a user process wants to perform IO operations, it must indirectly access the kernel space through **system calls.**

The things we come into contact with most in the normal development process are **disk IO (reading and writing files)** and **network IO (network requests and responses)** .

From the application's perspective, our application initiates an IO call (system call) to the operating system's kernel, which then performs the specific IO operations. In other words, our application actually only initiates the IO operation call, and the specific IO execution is completed by the operating system's kernel.

When an application initiates an I/O call, it goes through two steps:

1. The kernel waits for the I/O device to prepare data
2. The kernel copies data from kernel space to user space.

What are the common IO models?

Under the UNIX system, there are five IO models: **synchronous blocking I/O** , **synchronous non-blocking I/O** , **I/O multiplexing** , **signal-driven I/O** and **asynchronous I/O** .

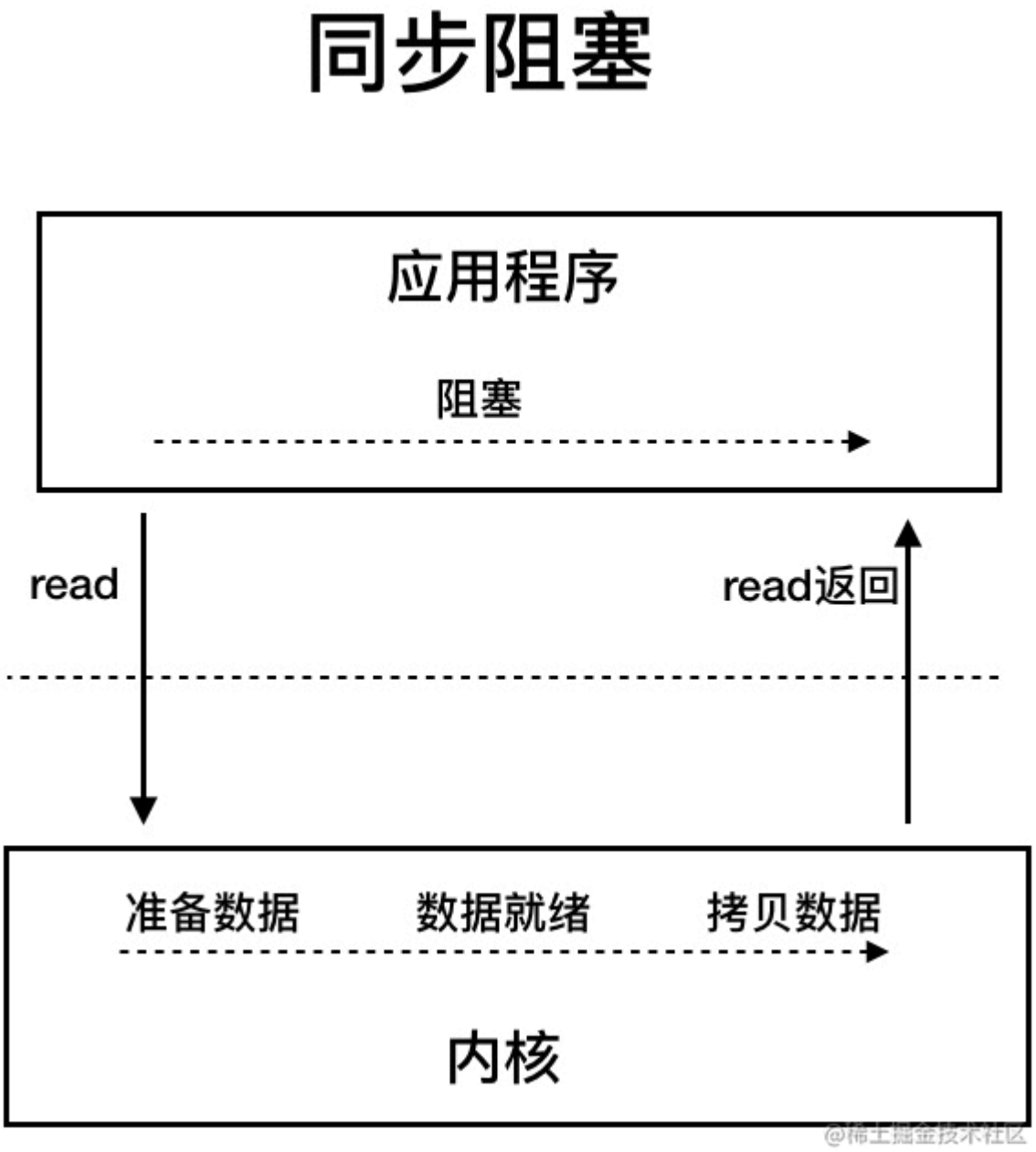
These are also the 5 IO models we often mention.

3 common IO models in Java

BIO (Blocking I/O)

BIO belongs to the synchronous blocking IO model .

In the synchronous blocking IO model, after the application initiates a read call, it will be blocked until the kernel copies the data to the user space.



This is fine when the number of client connections is low. However, when faced with hundreds of thousands or even millions of connections, the traditional BIO model is powerless. Therefore, we need a more efficient I/O processing model to handle higher concurrency.

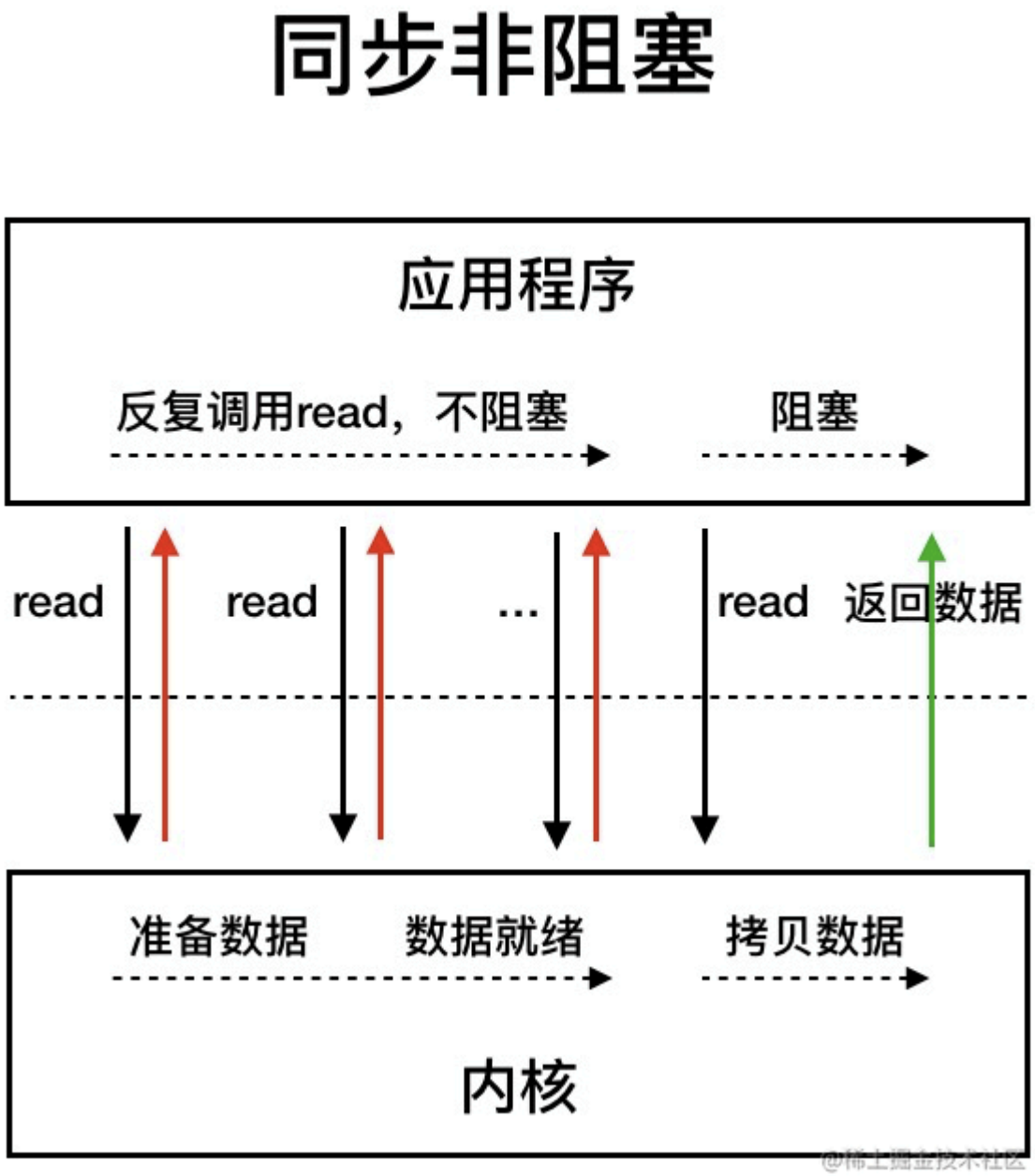
NIO (Non-blocking/New I/O)

NIO in Java was introduced in Java 1.4, corresponding to `java.nio` the package, and provides abstractions such as `Channel` , `Selector` , and `Buffer` . The "N" in NIO s be understood as non-blocking, not simply "New." It supports buffered, channel-based I/O operations. NIO should be used for high-load, high-concurrency (network) applications.

NIO in Java can be seen as **an I/O multiplexing model** . Many people also believe that NIO in Java belongs to the synchronous non-blocking IO model.

Follow my thoughts and read on, I believe you will get the answer!

Let's first look at **the synchronous non-blocking IO model** .



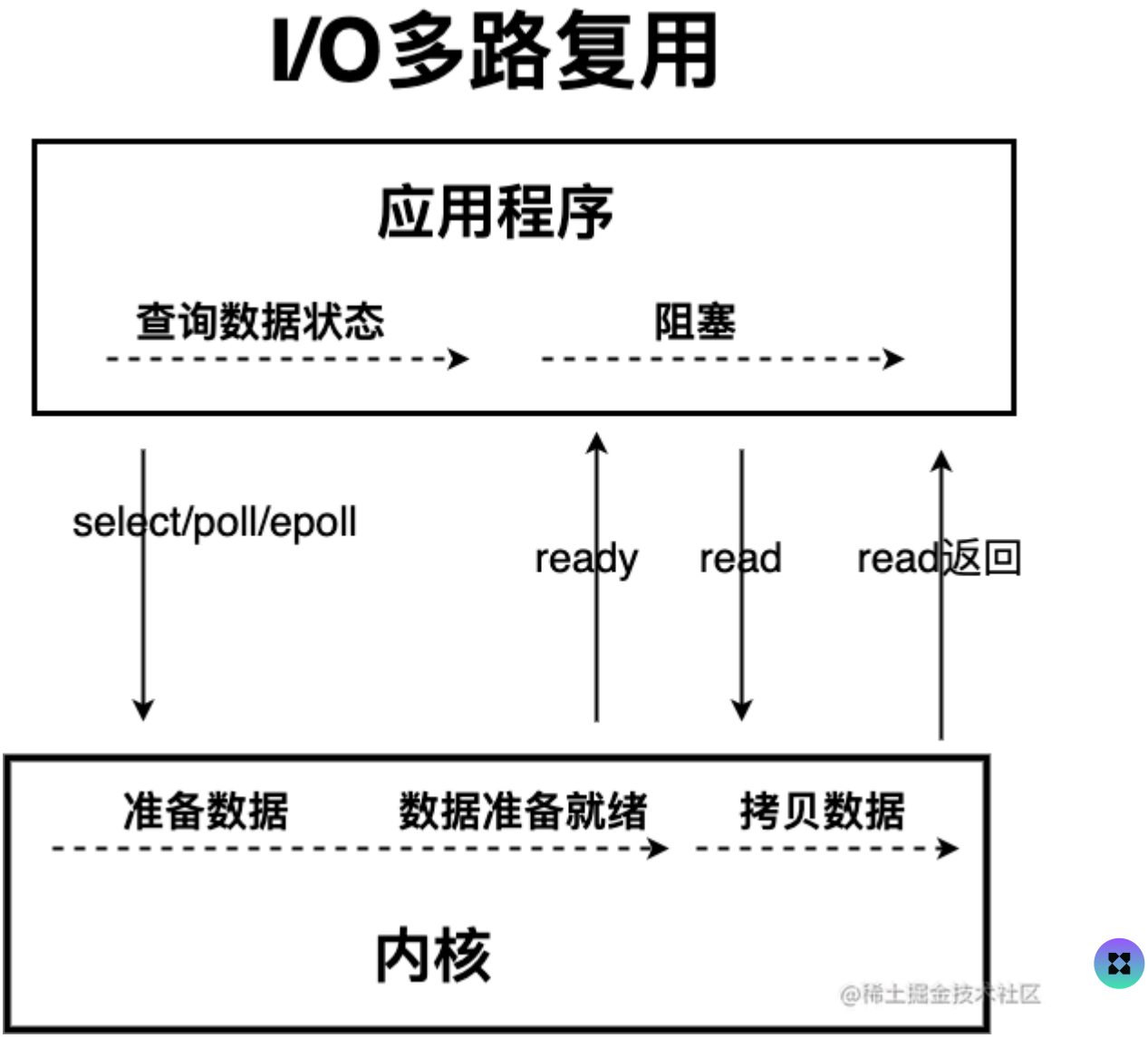
In the synchronous non-blocking IO model, the application will continue to initiate read calls, waiting for the data to be copied from the kernel space to the user space. During this time, the thread is still blocked until the kernel copies the data to the user space.

Compared with the synchronous blocking IO model, the synchronous non-blocking IO model is indeed a great improvement. Through polling operations, it avoids continuous blocking.

> With synchronous non-blocking IO, if a read call is initiated and the data isn't ready, the application can switch to performing some small computations rather than waiting, then quickly return to issuing another read call. This is known as polling. This polling isn't continuous; there are gaps. Exploiting these gaps is what makes synchronous non-blocking IO more efficient than synchronous blocking IO.

However, this IO model also has problems: **the process of applications constantly making I/O system calls to poll whether data is ready consumes a lot of CPU resources.**

At this time, **the I/O multiplexing model** comes into play.

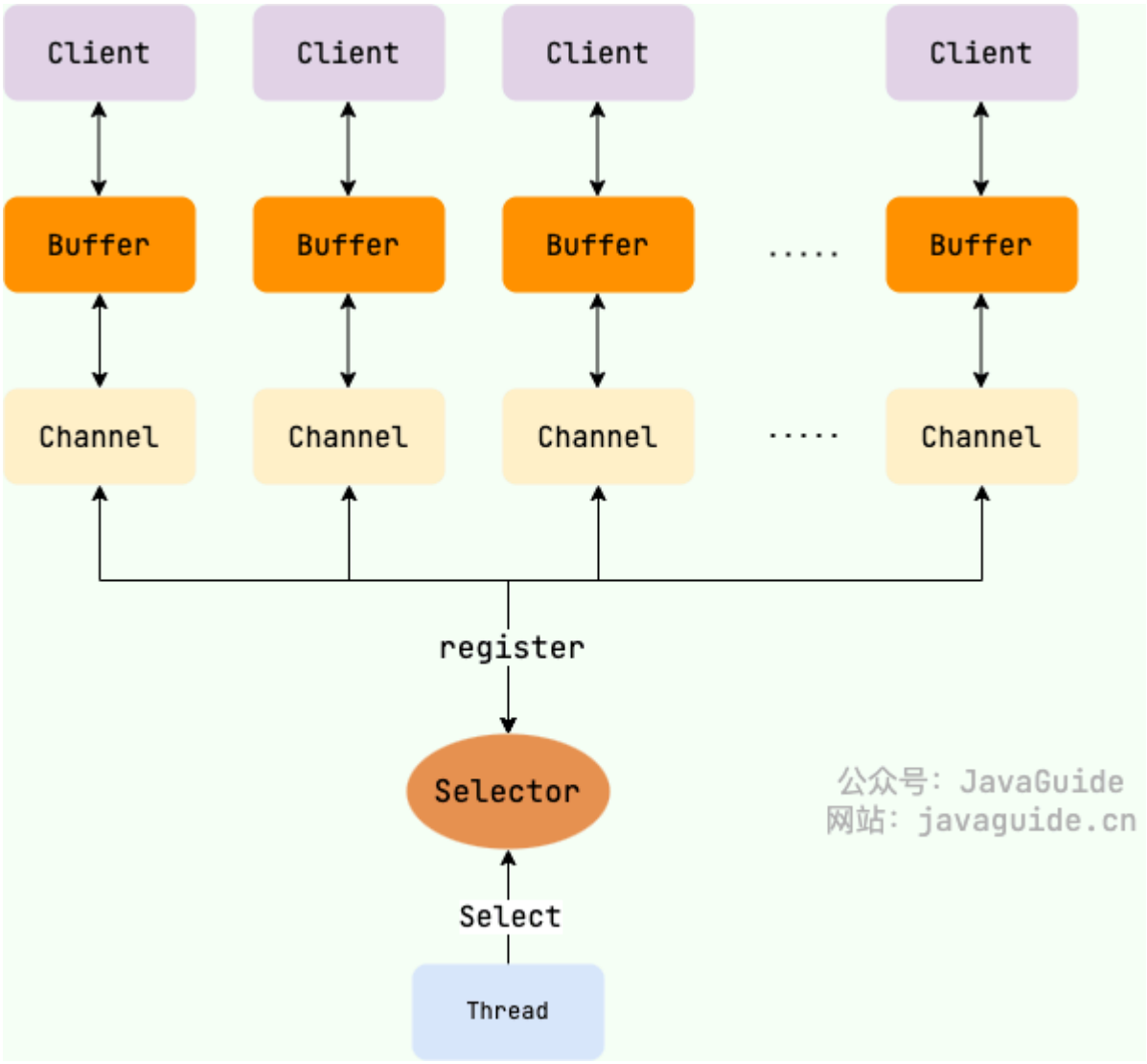


In the I/O multiplexing model, a thread first initiates a select call to query the kernel whether the data is ready. Once the kernel has the data ready, the user thread initiates a read call. The read call process (data transfer from kernel space to user space) is still blocking.

> Currently, the system calls that support IO multiplexing include select, epoll, etc. The select system call is currently supported on almost all operating systems.
>
> - **select call** : A system call provided by the kernel that supports querying the availability of multiple system calls at once. Almost all operating systems support it.
> - **epoll call** : Linux 2.6 kernel, an enhanced version of select call, optimizes IO execution efficiency.

The IO multiplexing model reduces CPU resource consumption by reducing invalid system calls.

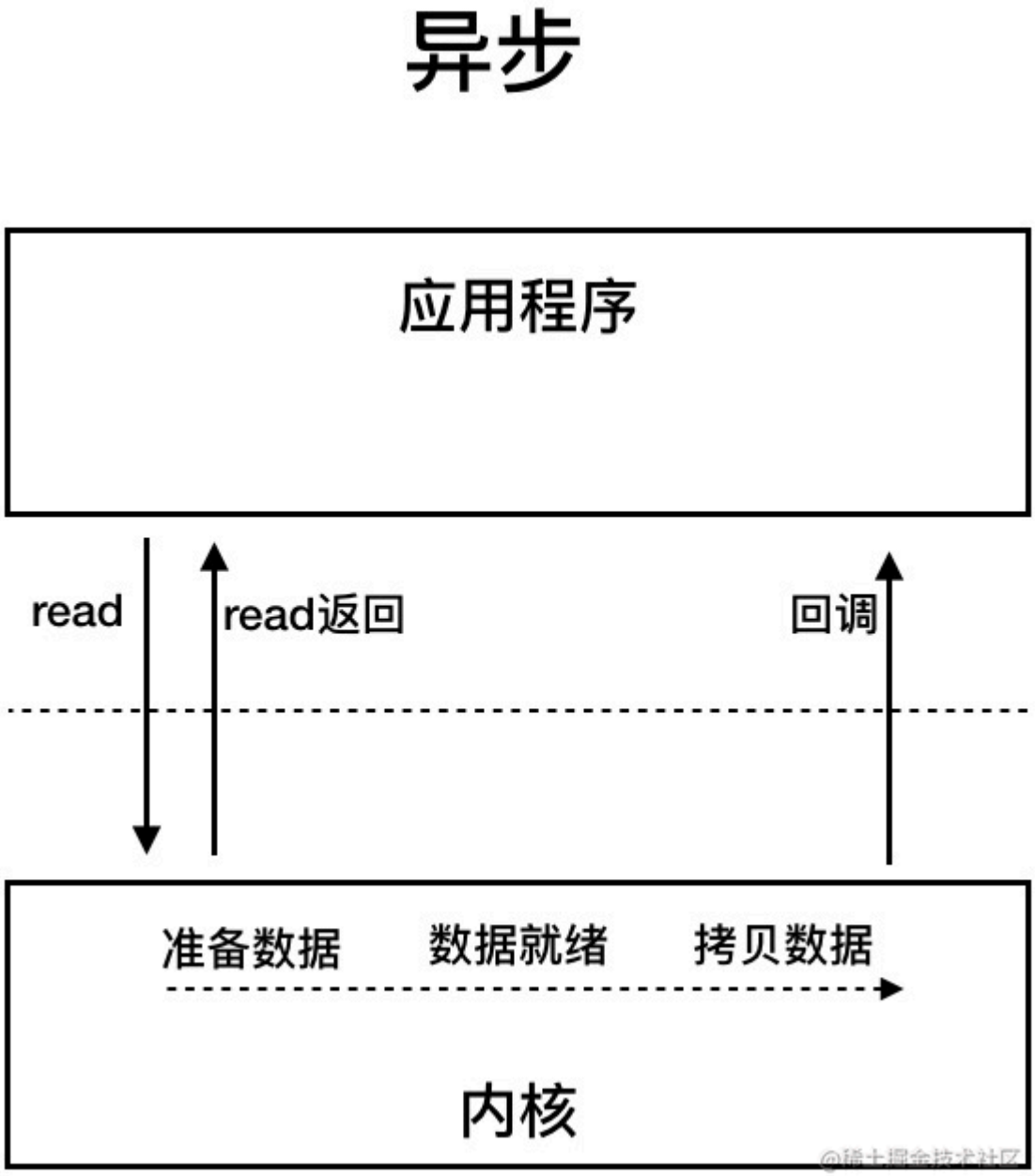
In Java's NIO, the concept of a selector , also known as **a multiplexer** , is crucial . It allows a single thread to manage multiple client connections. Client data is served only when it arrives.



AIO (Asynchronous I/O)

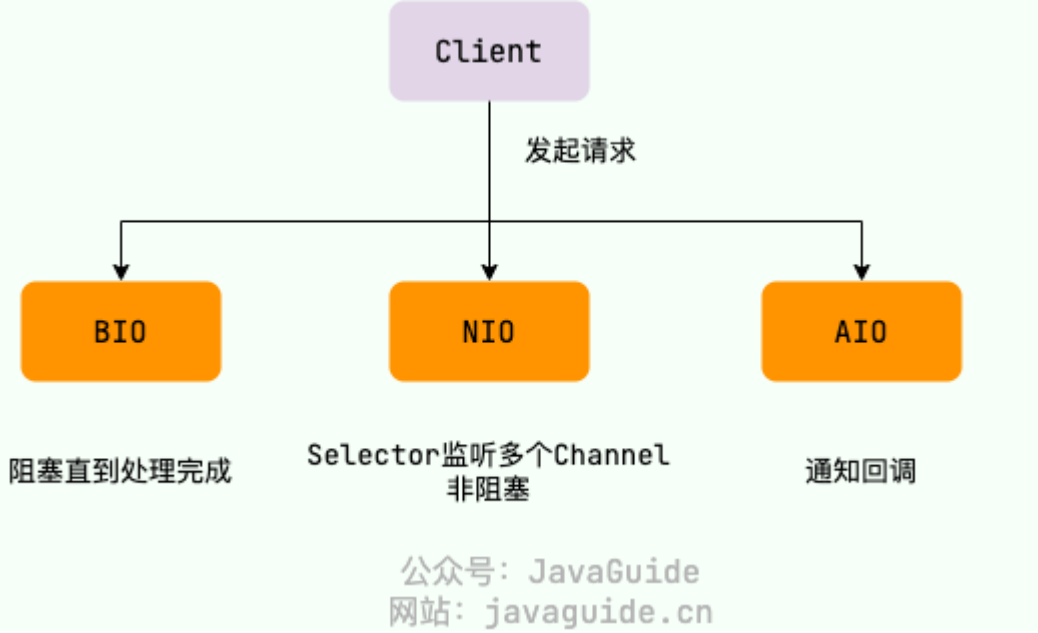
AIO is NIO 2. Java 7 introduced an improved version of NIO, NIO 2, which is an asynchronous IO model.

Asynchronous IO is implemented based on events and callback mechanisms, which means that the application will return directly after the operation and will not be blocked there. When the background processing is completed, the operating system will notify the corresponding thread to perform subsequent operations.



Currently, AIO isn't widely used. Netty previously experimented with using AIO but abandoned it. This was because using AIO didn't significantly improve Netty's performance on Linux systems.

Finally, here is a picture that briefly summarizes BIO, NIO, and AIO in Java.



refer to

- In-depth analysis of Tomcat & Jetty
- How to complete an IO: https://llc687.top/126.html
- Programmers should understand IO like this: https://www.jianshu.com/p/fa7bdc4f3de7
- Understand the underlying principles of Java NIO in 10 minutes: https://www.cnblogs.com/crazymakercircle/p/10225159.html
- How Much You Know About IO Models | Theory: https://www.cnblogs.com/sheng-jie/p/how-much-you-know-about-io-models.html
- "UNIX Network Programming Volume 1; Socket Networking API" Section 6.2 IO Model

Recently Updated2025/7/13 17:36
Contributors: guide , afon , Aaron Ge , anaer , sam , Guide , Mr.Hope , yanjunlin , seaflower