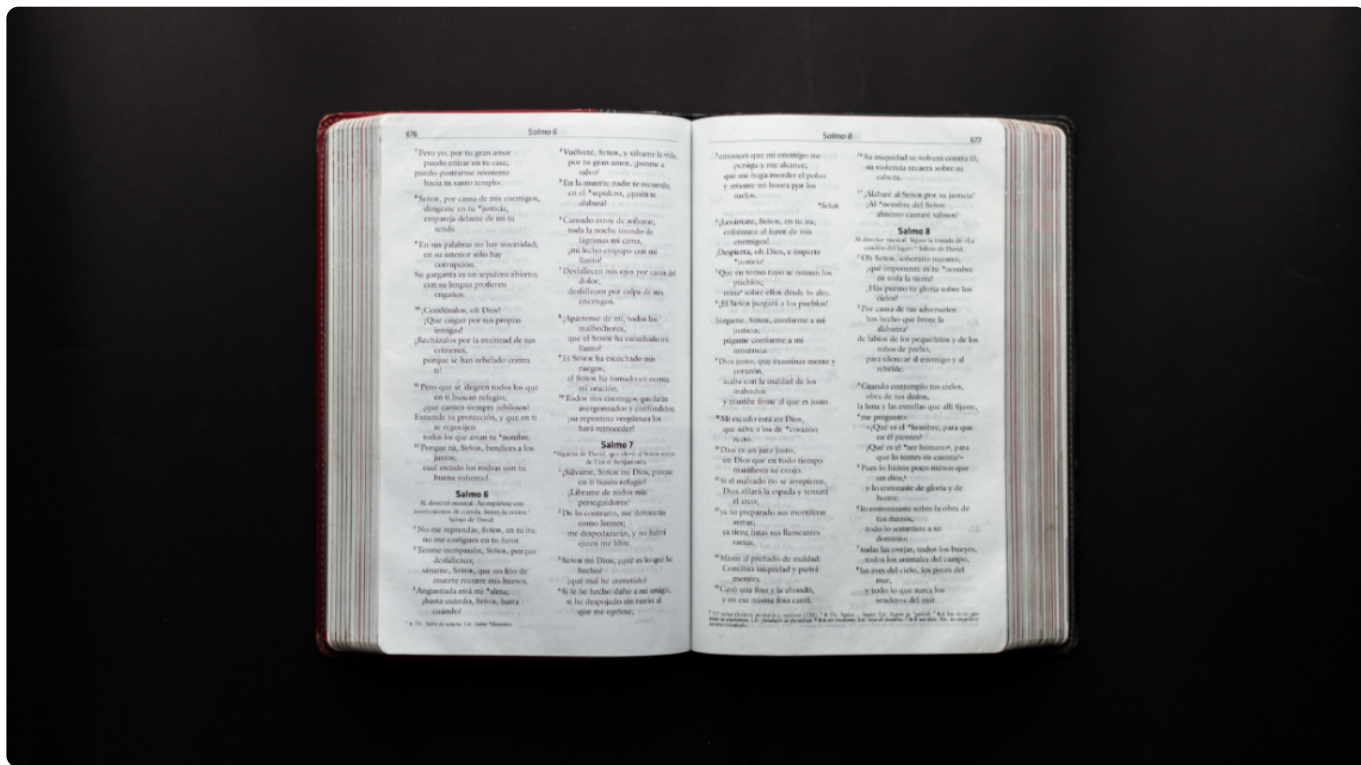


16 | AprEndpoint组件：Tomcat APR提高I/O性能的秘密

2019-06-15 李号双

深入拆解Tomcat & Jetty

[进入课程 >](#)



讲述：李号双

时长 11:22 大小 9.12M



我们在使用 Tomcat 时，会在启动日志里看到这样的提示信息：

```
The APR based Apache Tomcat Native library which allows optimal performance in production environments was not found on the java.library.path: ***
```

这句话的意思就是推荐你去安装 APR 库，可以提高系统性能。那什么是 APR 呢？

APR (Apache Portable Runtime Libraries) 是 Apache 可移植运行时库，它是用 C 语言实现的，其目的是向上层应用程序提供一个跨平台的操作系统接口库。Tomcat 可以用它来处理包括文件和网络 I/O，从而提升性能。我在专栏前面提到过，Tomcat 支持的连接器有 NIO、NIO.2 和 APR。跟 NioEndpoint 一样，AprEndpoint 也实现了非阻塞 I/O，它们

的区别是：NioEndpoint 通过调用 Java 的 NIO API 来实现非阻塞 I/O，而 AprEndpoint 是通过 JNI 调用 APR 本地库而实现非阻塞 I/O 的。

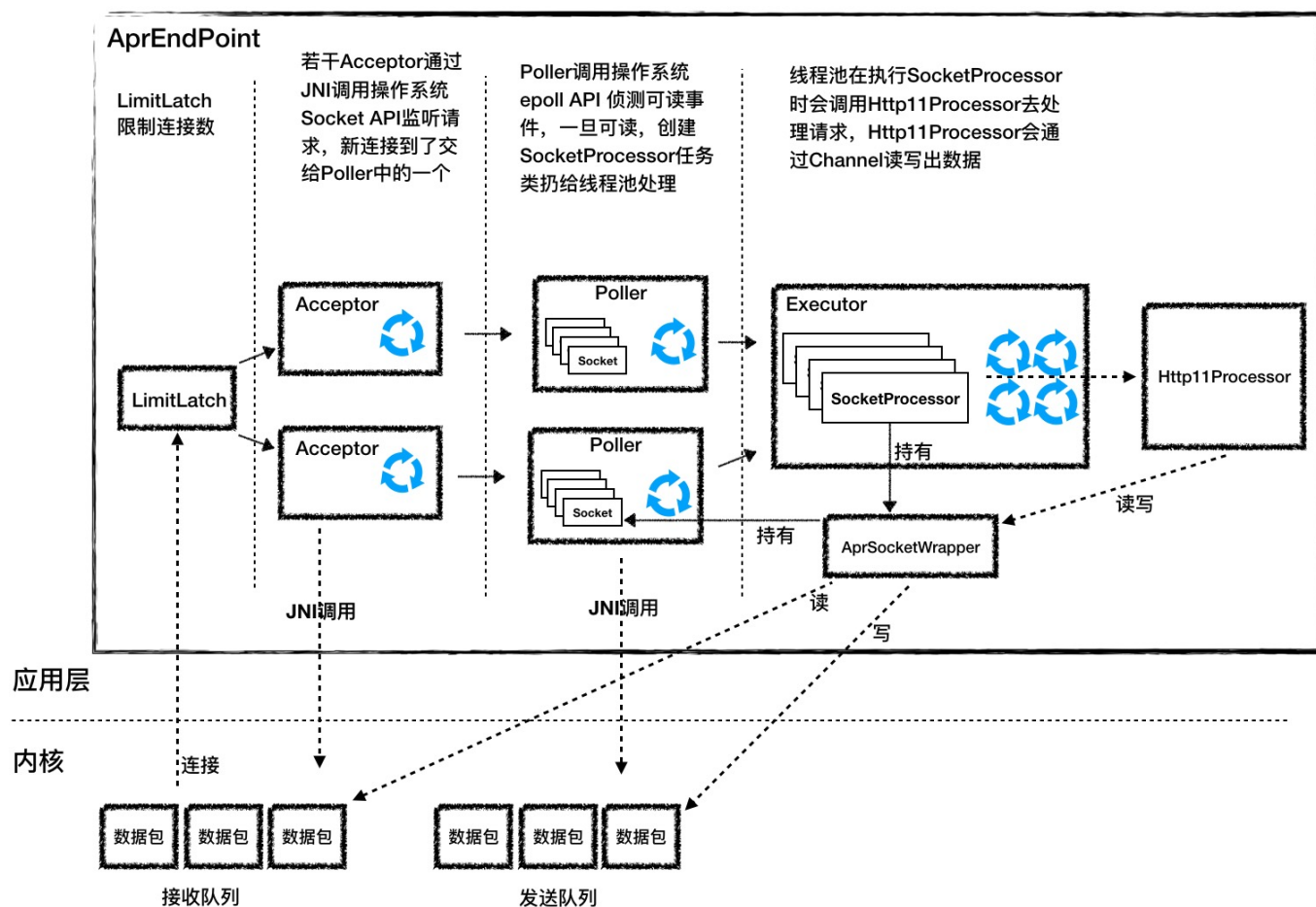
那同样是非阻塞 I/O，为什么 Tomcat 会提示使用 APR 本地库的性能会更好呢？这是因为在某些场景下，比如需要频繁与操作系统进行交互，Socket 网络通信就是这样一个场景，特别是如果你的 Web 应用使用了 TLS 来加密传输，我们知道 TLS 协议在握手过程中有多次网络交互，在这种情况下 Java 跟 C 语言程序相比还是有一定的差距，而这正是 APR 的强项。

Tomcat 本身是 Java 编写的，为了调用 C 语言编写的 APR，需要通过 JNI 方式来调用。JNI (Java Native Interface) 是 JDK 提供的一个编程接口，它允许 Java 程序调用其他语言编写的程序或者代码库，其实 JDK 本身的实现也大量用到 JNI 技术来调用本地 C 程序库。

在今天这一期文章，首先我会讲 AprEndpoint 组件的工作过程，接着我会在原理的基础上分析 APR 提升性能的一些秘密。在今天的学习过程中会涉及到一些操作系统的底层原理，毫无疑问掌握这些底层知识对于提高你的内功非常有帮助。

AprEndpoint 工作过程

下面我还是通过一张图来帮你理解 AprEndpoint 的工作过程。



你会发现它跟 NioEndpoint 的图很像，从左到右有 LimitLatch、Acceptor、Poller、SocketProcessor 和 Http11Processor，只是 Acceptor 和 Poller 的实现和 NioEndpoint 不同。接下来我分别来讲讲这两个组件。

Acceptor

Accpetor 的功能就是监听连接，接收并建立连接。它的本质就是调用了四个操作系统 API：socket、bind、listen 和 accept。那 Java 语言如何直接调用 C 语言 API 呢？答案就是通过 JNI。具体来说就是两步：先封装一个 Java 类，在里面定义一堆用 **native 关键字** 修饰的方法，像下面这样。

[复制代码](#)


```

1 public class Socket {
2     ...
3     // 用 native 修饰这个方法，表明这个函数是 C 语言实现
4     public static native long create(int family, int type,
5                                     int protocol, long cont)
6
7     public static native int bind(long sock, long sa);
8
9     public static native int listen(long sock, int backlog);

```

```
10
11 public static native long accept(long sock)
12 }
```

接着用 C 代码实现这些方法，比如 bind 函数就是这样实现的：

 复制代码

```
1 // 注意函数的名字要符合 JNI 规范的要求
2 JNIEXPORT jint JNICALL
3 Java_org_apache_tomcat_jni_Socket_bind(JNIEnv *e, jlong sock, jlong sa)
4 {
5     jint rv = APR_SUCCESS;
6     tcn_socket_t *s = (tcn_socket_t *) sock;
7     apr_sockaddr_t *a = (apr_sockaddr_t *) sa;
8
9     // 调用 APR 库自己实现的 bind 函数
10    rv = (jint)apr_socket_bind(s->sock, a);
11    return rv;
12 }
```

专栏里我就不展开 JNI 的细节了，你可以[扩展阅读](#)获得更多信息和例子。我们要注意的是函数名字要符合 JNI 的规范，以及 Java 和 C 语言如何互相传递参数，比如在 C 语言有指针，Java 没有指针的概念，所以在 Java 中用 long 类型来表示指针。AprEndpoint 的 Acceptor 组件就是调用了 APR 实现的四个 API。

Poller

Acceptor 接收到一个新的 Socket 连接后，按照 NioEndpoint 的实现，它会把这个 Socket 交给 Poller 去查询 I/O 事件。AprEndpoint 也是这样做的，不过 AprEndpoint 的 Poller 并不是调用 Java NIO 里的 Selector 来查询 Socket 的状态，而是通过 JNI 调用 APR 中的 poll 方法，而 APR 又是调用了操作系统的 epoll API 来实现的。

这里有个特别的地方是在 AprEndpoint 中，我们可以配置一个叫 deferAccept 的参数，它对应的是 TCP 协议中的 TCP_DEFER_ACCEPT，设置这个参数后，当 TCP 客户端有新的连接请求到达时，TCP 服务端先不建立连接，而是再等等，直到客户端有请求数据发过来时再建立连接。这样的好处是服务端不需要用 Selector 去反复查询请求数据是否就绪。

这是一种 TCP 协议层的优化，不是每个操作系统内核都支持，因为 Java 作为一种跨平台语言，需要屏蔽各种操作系统的差异，因此并没有把这个参数提供给用户；但是对于 APR 来说，它的目的就是尽可能提升性能，因此它向用户暴露了这个参数。

APR 提升性能的秘密

APR 连接器之所以能提高 Tomcat 的性能，除了 APR 本身是 C 程序库之外，还有哪些提速的秘密呢？

JVM 堆 VS 本地内存

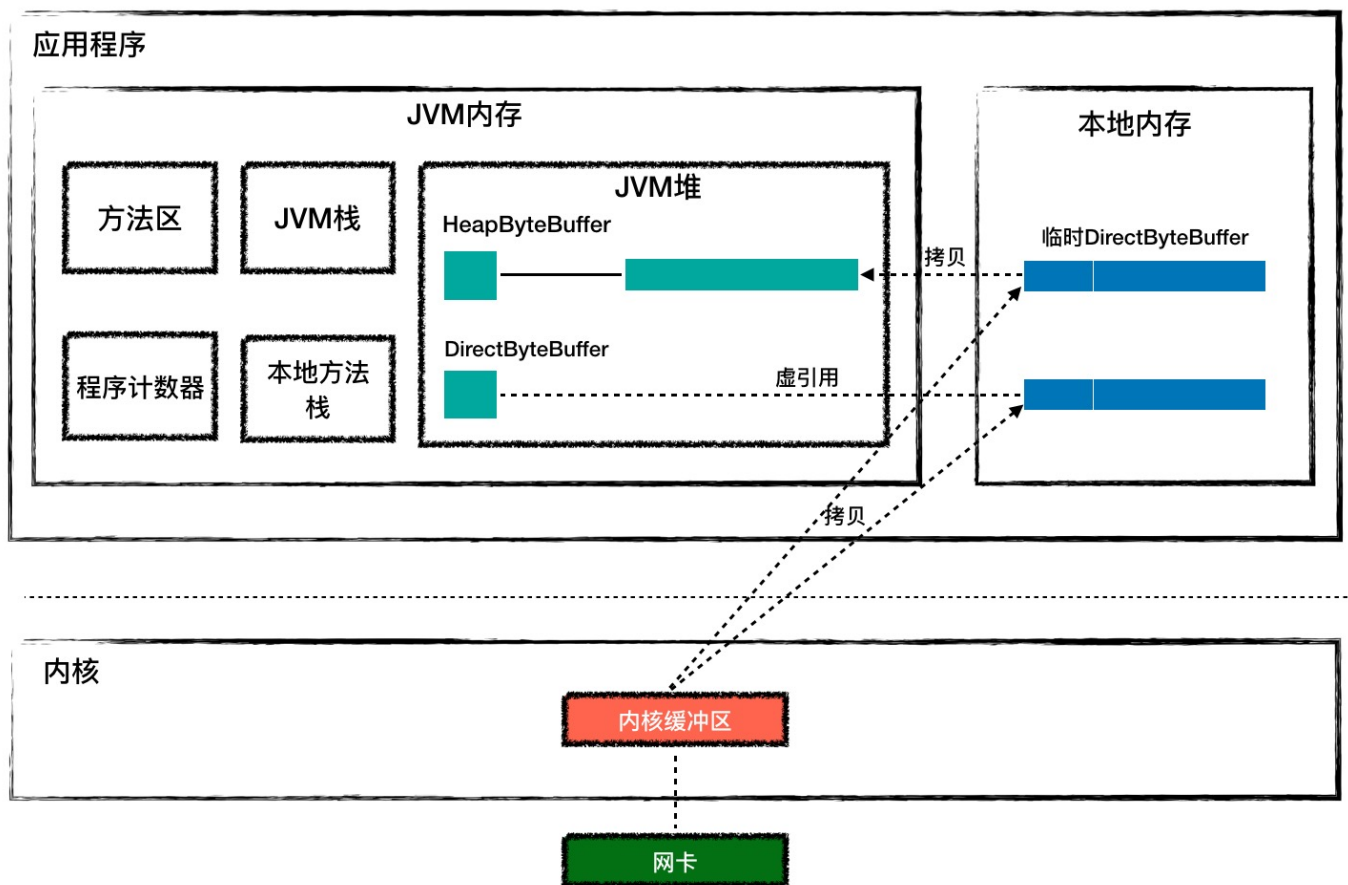
我们知道 Java 的类实例一般在 JVM 堆上分配，而 Java 是通过 JNI 调用 C 代码来实现 Socket 通信的，那么 C 代码在运行过程中需要的内存又是从哪里分配的呢？C 代码能否直接操作 Java 堆？

为了回答这些问题，我先来说说 JVM 和用户进程的关系。如果你想运行一个 Java 类文件，可以用下面的 Java 命令来执行。

 复制代码

```
1 java my.class
```

这个命令行中的 `java` 其实是一个可执行程序，这个程序会创建 JVM 来加载和运行你的 **Java 类**。操作系统会创建一个进程来执行这个 `java` 可执行程序，而每个进程都有自己的虚拟地址空间，JVM 用到的内存（包括堆、栈和方法区）就是从进程的虚拟地址空间上分配的。请你注意的是，JVM 内存只是进程空间的一部分，除此之外进程空间内还有代码段、数据段、内存映射区、内核空间等。从 JVM 的角度看，JVM 内存之外的部分叫作本地内存，C 程序代码在运行过程中用到的内存就是本地内存中分配的。下面我们通过一张图来了解一下。



Tomcat 的 Endpoint 组件在接收网络数据时需要预先分配好一块 Buffer，所谓的 Buffer 就是字节数组 `byte[]`，Java 通过 JNI 调用把这块 Buffer 的地址传给 C 代码，C 代码通过操作系统 API 读取 Socket 并把数据填充到这块 Buffer。Java NIO API 提供了两种 Buffer 来接收数据：HeapByteBuffer 和 DirectByteBuffer，下面的代码演示了如何创建两种 Buffer。

[复制代码](#)

```
1 // 分配 HeapByteBuffer
2 ByteBuffer buf = ByteBuffer.allocate(1024);
3
4 // 分配 DirectByteBuffer
5 ByteBuffer buf = ByteBuffer.allocateDirect(1024);
```

创建好 Buffer 后直接传给 Channel 的 read 或者 write 函数，最终这块 Buffer 会通过 JNI 调用传递给 C 程序。

[复制代码](#)

```
1 // 将 buf 作为 read 函数的参数
2 int bytesRead = socketChannel.read(buf);
```

那 `HeapByteBuffer` 和 `DirectByteBuffer` 有什么区别呢？`HeapByteBuffer` 对象本身在 JVM 堆上分配，并且它持有的字节数组 `byte[]` 也是在 JVM 堆上分配。但是如果用 **`HeapByteBuffer`** 来接收网络数据，**需要把数据从内核先拷贝到一个临时的本地内存，再从临时本地内存拷贝到 JVM 堆**，而不是直接从内核拷贝到 JVM 堆上。这是为什么呢？这是因为数据从内核拷贝到 JVM 堆的过程中，JVM 可能会发生 GC，GC 过程中对象可能会被移动，也就是说 JVM 堆上的字节数组可能会被移动，这样的话 Buffer 地址就失效了。如果这中间经过本地内存中转，从本地内存到 JVM 堆的拷贝过程中 JVM 可以保证不做 GC。

如果使用 `HeapByteBuffer`，你会发现 JVM 堆和内核之间多了一层中转，而 `DirectByteBuffer` 用来解决这个问题，`DirectByteBuffer` 对象本身在 JVM 堆上，但是它持有的字节数组不是从 JVM 堆上分配的，而是从本地内存分配的。`DirectByteBuffer` 对象中有个 `long` 类型字段 `address`，记录着本地内存的地址，这样在接收数据的时候，直接把这个本地内存地址传递给 C 程序，C 程序会将网络数据从内核拷贝到这个本地内存，JVM 可以直接读取这个本地内存，这种方式比 `HeapByteBuffer` 少了一次拷贝，因此一般来说它的速度会比 `HeapByteBuffer` 快好几倍。你可以通过上面的图加深理解。

Tomcat 中的 `AprEndpoint` 就是通过 `DirectByteBuffer` 来接收数据的，而 `NioEndpoint` 和 `Nio2Endpoint` 是通过 `HeapByteBuffer` 来接收数据的。你可能会问，`NioEndpoint` 和 `Nio2Endpoint` 为什么不用 `DirectByteBuffer` 呢？这是因为本地内存不好管理，发生内存泄漏难以定位，从稳定性考虑，`NioEndpoint` 和 `Nio2Endpoint` 没有去冒这个险。

sendfile

我们再来考虑另一个网络通信的场景，也就是静态文件的处理。浏览器通过 Tomcat 来获取一个 HTML 文件，而 Tomcat 的处理逻辑无非是两步：

1. 从磁盘读取 HTML 到内存。
2. 将这段内存的内容通过 Socket 发送出去。

但是在传统方式下，有很多次的内存拷贝：

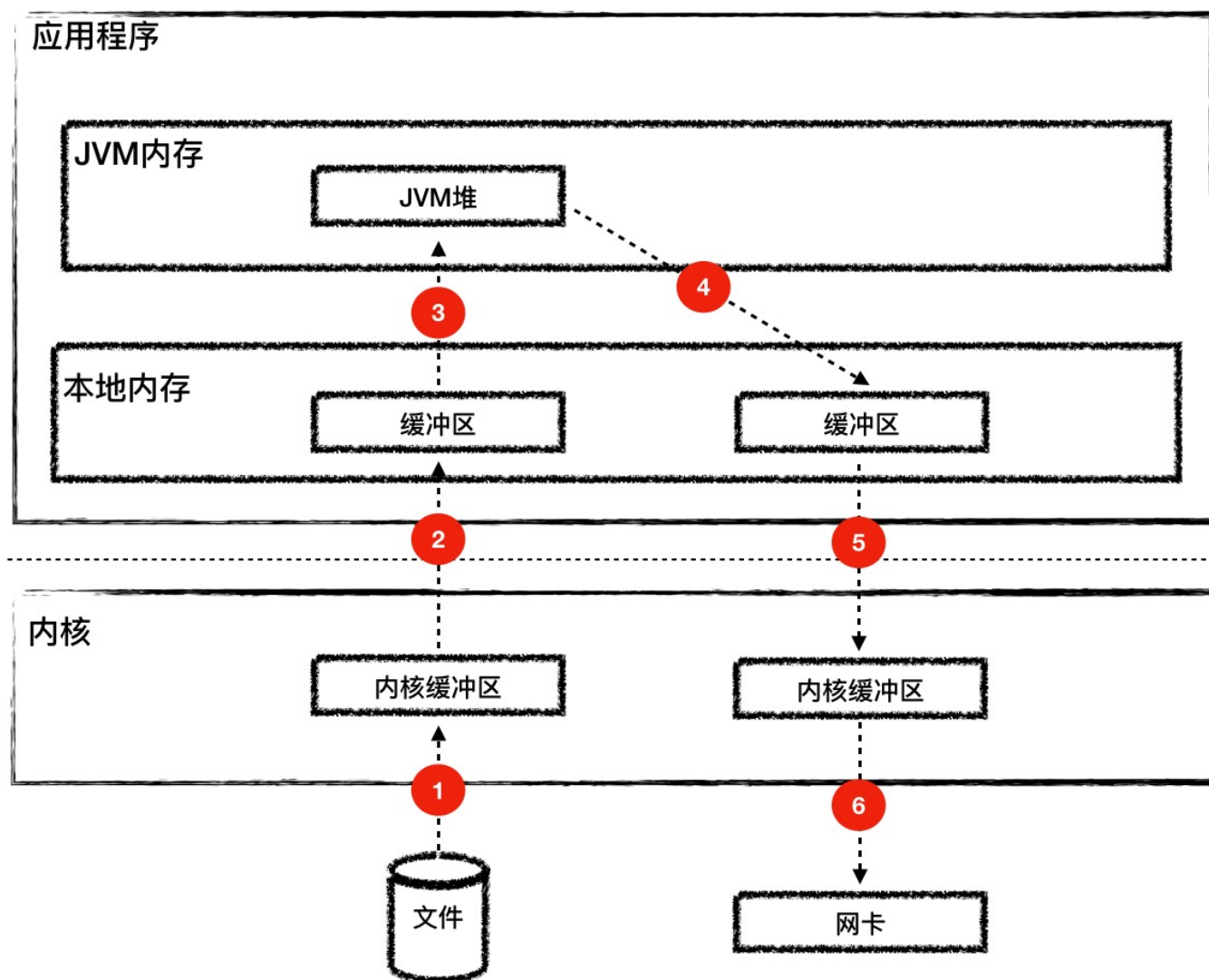
读取文件时，首先是内核把文件内容读取到内核缓冲区。

如果使用 HeapByteBuffer，文件数据从内核到 JVM 堆内存需要经过本地内存中转。

同样在将文件内容推入网络时，从 JVM 堆到内核缓冲区需要经过本地内存中转。

最后还需要把文件从内核缓冲区拷贝到网卡缓冲区。

从下面的图你会发现这个过程有 6 次内存拷贝，并且 read 和 write 等系统调用将导致进程从用户态到内核态的切换，会耗费大量的 CPU 和内存资源。



而 Tomcat 的 AprEndpoint 通过操作系统层面的 sendfile 特性解决了这个问题，sendfile 系统调用方式非常简洁。

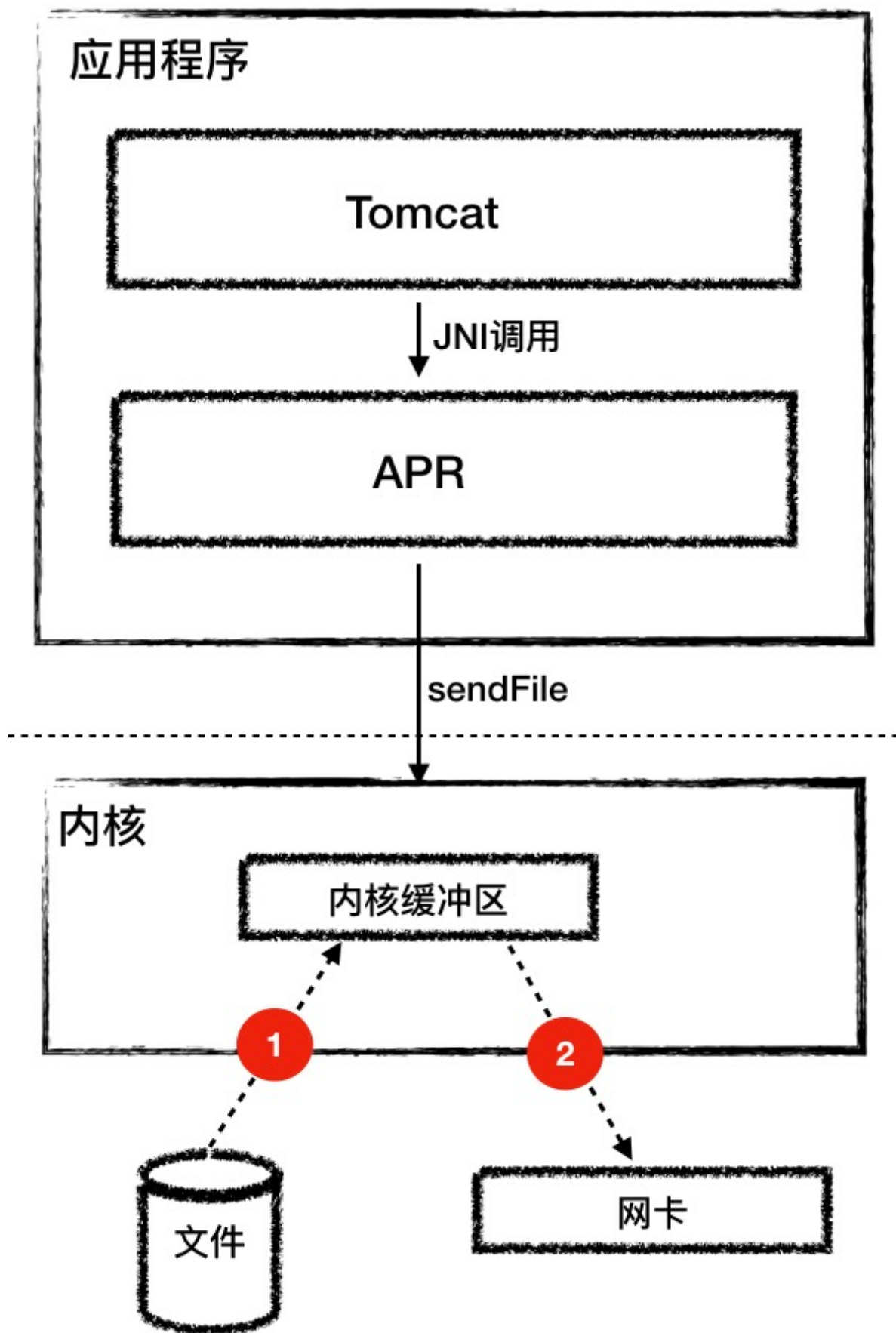
复制代码

```
1 sendfile(socket, file, len);
```


它带有两个关键参数：Socket 和文件句柄。将文件从磁盘写入 Socket 的过程只有两步：

第一步：将文件内容读取到内核缓冲区。

第二步：数据并没有从内核缓冲区复制到 Socket 关联的缓冲区，只有记录数据位置和长度的描述符被添加到 Socket 缓冲区中；接着把数据直接从内核缓冲区传递给网卡。这个过程你可以看下面的图。



本期精华

对于一些需要频繁与操作系统进行交互的场景，比如网络通信，Java 的效率没有 C 语言高，特别是 TLS 协议握手过程中需要多次网络交互，这种情况下使用 APR 本地库能够显著

提升性能。

除此之外，APR 提升性能的秘密还有：通过 DirectByteBuffer 避免了 JVM 堆与本地内存之间的内存拷贝；通过 sendfile 特性避免了内核与应用之间的内存拷贝以及用户态和内核态的切换。其实很多高性能网络通信组件，比如 Netty，都是通过 DirectByteBuffer 来收发网络数据的。由于本地内存难于管理，Netty 采用了本地内存池技术，感兴趣的同学可以深入了解一下。

课后思考

为什么不同的操作系统，比如 Linux 和 Windows，都有自己的 Java 虚拟机？

不知道今天的内容你消化得如何？如果还有疑问，请大胆的在留言区提问，也欢迎你把你的课后思考和心得记录下来，与我和其他同学一起讨论。如果你觉得今天有所收获，欢迎你把它分享给你的朋友。



深入拆解 Tomcat & Jetty

从源码角度深度探索 Java 中间件

李号双

eBay 技术主管



新版升级：点击「👤请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 15 | Nio2Endpoint组件：Tomcat如何实现异步I/O？

精选留言 (7)

写留言



YF

2019-06-15

1

老师，为什么从本地内存到 JVM 堆的拷贝过程中 JVM 可以保证不做 GC呢？那为什么从内核拷贝到JVM堆中就不能保证做GC呢？

展开

作者回复: 这是HotSpot VM层面保证的，具体来说就是HotSpot不是什么时候都能GC的，需要JVM中各个线程都处在“safepoint”时才能GC，本地内存到JVM堆的拷贝过程中是没有“safepoint”的，所以不会GC，至于什么是“safepoint”，你可以搜索一下。



nihil

2019-06-15

1

学到了，之前还纳闷，我在使用高版本的SpringBoot的时候总是会有arp的警告，今天终于知道是干嘛的了。感谢老师。

展开



刘章周

2019-06-15

1

java语言有个特性，一次编译到处执行，而class文件需要通过虚拟机编译成操作系统指令。不同操作系统的指令不一样，所以有对应的虚拟机来简化代码开发。c语言好像是直接调用操作系统指令，代码开发中调用一个方法，不同的操作系统可能不一样，还得准备两份。老师，说的对吗？

展开

作者回复: 对的



nightmare

2019-06-15

1

老师，arp模式更加适合https，或者文件传输的业务场景，相对于nio会有更快的io速度



粉条炒肉

2019-06-15

1

老师，请教一个问题，每次服务的访问高峰期重启或者发布tomcat服务，都会出现阻塞线程过多 这种情况是不是容器初始化的时候需要预热很多资源，从而响应不过来请求而导致阻塞，关于这种情况，有什么好的解决办法。

作者回复: 服务没有启动完成的时候，不要把流量送过来。



z.l

2019-06-15

1

请教老师, DirectByteBuffer和常说的mmap是什么关系？



陆离

2019-06-15

1

虚拟机只是抽象了操作系统，但并不是虚拟机也是一套的。
jvm的解释器在不同操作系统下是不同的，因为要将字节码解释生成本地机器码。
这么简单个问题，感觉还是解释不清楚，这个要和操作系统原理联系起来了。
烦请老师做一个详细的简单！

展开 ∨

作者回复: JVM为了跟操作系统打交道，必须调用操作系统提供的API，而每个操作系统提供的API都不同，所以必须针对不同的操作系统实现不同的JVM，这样位于JVM之上的Java字节码才可以实现跨平台。