目录

**第1级 1**

第1.1级 2

第1.2级 3

**第2级 4**

第2.1级 5

第2.2级 6

目录

第3级 1

第3.1级 2

第3.1.1级 3

第4级 4

第4.1级 5

第4.1.1级 6

IO多路复用与分时复用。

**select、poll、epoll都是IO多路复用机制，可以监视多个描述符的读/写等事件，一旦某个描述符就绪（一般是读或者写事件发生了），就能够将发生的事件通知给关心的应用程序去处理该事件。 像斥候的角色，监视着敌人的一举一动。**

**关于MessageQueue**

**Java层和Native层的MessageQueue通过JNI建立关联，从而使得MessageQueue成为Java层和Native层的枢纽，既能处理上层消息，也能处理native层消息，而Handler/Looper/Message这三大类在Java层与Native层之间没有任何的关联，只是分别在Java层和Native层的消息模型中具有相似的功能，都是彼此独立的，各自实现相应的逻辑。**

**这里我们可以回答为什么java层的loop方法是死循环但却不会消耗性能这个问题：**

**因为java层的消息机制是依赖native层的消息机制来实现的，而native层的消息机制是通过Linux的管道和epoll机制实现的，epoll机制是一种高效的IO多路复用机制【单进程/线程同时监测若干个文件描述符是否可以执行IO操作的能力】， 它使用一个文件描述符管理多个描述符，java层通过mPtr指针也就共享了native层的epoll机制的高效性，当loop方法中取不到消息时，便**

**阻塞在MessageQueue的next方法，**

**而next方法阻塞在nativePollOnce方法，nativePollOnce方法**

**通过JNI调用进入到native层中去，最终nativePollOnce方法阻塞在epoll\_wait方法中，**

**epoll\_wait方法会让当前线程释放CPU资源进入休眠状态，等到下一个消息到达(mWakeEventFd【唤醒文件描述符】会往管道写入字符)或监听的其他事件发生时就会唤醒线程，然后处理消息，所以就算loop方法是死循环，当线程空闲时，它会进入休眠状态，不会消耗大量的CPU资源。**