

目录	
第1章 基础	
01 开篇词：为什么学习本专栏	
02 String、Long 源码解析和面试题	
03 Java 常用关键字理解	
04 Arrays、Collections、Objects 常用方法源码解析	
第2章 集合	
05 ArrayList 源码解析和设计思路	
06 LinkedList 源码解析	
07 List 源码会问哪些面试题	
08 HashMap 源码解析	
09 TreeMap 和 LinkedHashMap 核心源码解析	
10 Map源码会问哪些面试题	
11 HashSet、TreeSet 源码解析	
12 彰显细节：看集合源码对我们实际工作的帮助和应用	
13 差异对比：集合在 Java 7 和 8 有何不同和改进	
14 简化工作：Guava Lists Maps 实际工作运用和源码	
第3章 并发集合类	
15 CopyOnWriteArrayList 源码解析和设计思路	
16 ConcurrentHashMap 源码解析和设计思路	
17 并发 List、Map源码面试题	
18 场景集合：并发 List、Map的应用	

46 ServerSocket 源码及面试题

更新时间：2019-12-02 09:48:40



“既然我已经踏上这条道路，那么，任何东西都不应妨碍我沿着这条路走下去。”

果断更，请联系QQ/微信64260066康德

上一小节我们学习了 Socket，本文我们来看看服务端套接字 API：ServerSocket，本文学习完毕之后，我们就可以把客服端 Socket 和服务端 ServerSocket 串联起来，做一个真实的网络通信的 demo 了。

1 类属性

ServerSocket 的主要作用，是作为服务端的套接字，接受客户端套接字传递过来的信息，并把响应回传给客户端，其属性非常简单，如下：

```
private boolean created = false;// 已创建
private boolean bound = false;// 绑定
private boolean closed = false;// 已关闭
// 底层的功能都依靠 SocketImpl 来实现
private SocketImpl impl;
```

ServerSocket 和 Socket 一样，底层都是依靠 SocketImpl 的能力，而 SocketImpl 底层能力的实现基本上都是 native 方法实现的。

2 初始化

初始化大概可以分成两类：无参构造器和有参构造器。

1. 无参构造器做的事情比较简单，只指定了 SocketImpl 为 SocksSocketImpl 类；

目录	<pre>public ServerSocket(int port, int backlog, InetAddress bindAddr) throws IOException { // 默认是 SocksSocketImpl 实现 setImpl(); // 端口必须大于 0, 小于 65535 if (port < 0 port > 0xFFFF) throw new IllegalArgumentException("Port value out of range: " + port); // 最大可连接数如果小于1, 那么采取默认的 50 if (backlog < 1) backlog = 50; try { // 底层 native 方法 bind(new InetSocketAddress(bindAddr, port), backlog); } catch (SecurityException e) { close(); throw e; } catch (IOException e) { close(); throw e; } }</pre>
----	--

入参 port 指的是 ServerSocket 需要绑定本地那个端口。

入参 backlog 指的是服务端接受客户端连接队列的最大长度，这里需要注意的是，这里并不是限制客户端连接的个数，我们在 JDK8 版本下做过实验，我们把服务端的 backlog 设置成 1，并且变慢服务端的处理速度，当服务端并发请求过来时，并不是第二个请求过来就拒绝连接，我们在实际工作中，最好也不要 用 backlog 来限制客户端连接的个数。

还有一点需要注意的是 backlog 小于 1 时，backlog 会被设置成默认的 50。

入参 InetAddress 表示 ip 地址。

3 bind

bind 方法主要作用是把 ServerSocket 绑定到本地的端口上，只有当我们使用无参构造器初始化 ServerSocket 时，才会用到这个方法，如果使用有参构造器的话，在初始化时就已经绑定到本地的端口上了。

配合无参构造器，一般我们这么用：

```
// 进行初始化  
ServerSocket serverSocket = new ServerSocket();  
// 进行绑定  
serverSocket.bind(new InetSocketAddress("localhost", 7007));
```

4 accept

accept 方法主要是用来 ServerSocket 接受来自客户端的套接字的，如果此时没有来自客户端的请求时，该方法就会一直阻塞，如果有通过 setSoTimeout 方法设置超时时间，那么 accept 只会在超时间内阻塞，过了超时时间就会抛出异常。

bind 和 accept 方法底层都是 native 方法实现，我们就不看源码了。

目录

5.1 说说你对 Socket 和 ServerSocket 的理解？

答：两者我们都可以称为套接字，底层基于 TCP/UDP 协议，套接字对底层协议进行了封装，让我们使用时更加方便，Socket 常被使用在客户端，用于向服务端请求数据和接受响应，ServerSocket 常用于在服务端，用于接受客户端的请求并进行处理，两者其底层使用都是依靠 SocketImpl 的子类的 native 方法。

5.2 说说对 SocketOptions 中的 SO_TIMEOUT 的理解？

答：SocketOptions 类有很多属性设置，比如 SO_TIMEOUT、SO_LINGER 等等，这些问题说一下自己的理解即可，可以参考《Socket 源码及面试题》中对各种属性的解析。

5.3 在构造 Socket 的时候，我可以选择 TCP 或 UDP 么？应该如何选择？

答：可以的，Socket 有三个参数的构造器，第三个参数表示你想使用 TCP 还是 UDP。

5.4 TCP 有自动检测服务端是否存活的机制么？有没有更好的办法？

答：有的，我们可以通过 setKeepAlive 方法来激活该功能，如果两小时内，客户端和服务端的套接字之间没有任何通信，TCP 会自动发送 keepalive 探测给服务端，预测服务端有三种情况：

1. 服务端使用预期的 ACK 回复，说明一切正常；
2. 服务端回复 RST，表示服务端处于死机或者重启状态，终止连接；
3. 没有得到服务端的响应（会尝试多次），表示套接字已经关闭了。

但我们并不建议使用这种方式，我们可以自己起一个定时任务，定时的访问服务端的特殊接口，如果服务端返回的数据和预期一致，说明服务端是存活的。

总结

Socket 和 ServerSocket 在源码方面没啥特别可说的地方，基本都是一些设置，底层实现都是 native 的方法，但面试官会从此延伸到一些网络协议方面的知识，因为这已经超出本专栏的范畴了，感兴趣的同学可以自行百度。

精选留言 0

欢迎在这里发表留言，作者筛选后可公开显示



目前暂无任何讨论

果断更， 请联系QQ/微信64260063