Sockets en Java — Resumen completo con marcadores, mapa mental y diagramas

Autora: Siham Ennahal
Fecha: 16/12/2025
Este documento sigue el estilo de apuntes solicitado: definición clara, tipos, sintaxis de métodos, ejemplos completos y buenas prácticas. Incluye un mapa mental textual y diagramas de secuencia en formato ASCII.

¿Qué es un socket?

Un socket es un canal de comunicación bidireccional, usado por dos procesos para enviarse datos por la red.
Se comporta como un descriptor de fichero: ➡ se puede leer y escribir igual que si fuese un archivo.

Tipos de sockets en Java

- Sockets Stream (TCP): orientados a conexión; fiabilidad, orden y control de errores; datos como flujo continuo de bytes.
- Sockets Datagrama (UDP): no orientados a conexión; entrega no garantizada (pueden perderse/duplicarse); datos en paquetes; más rápidos.
- Raw Sockets (IP): acceso directo a nivel IP (debugging); **no disponibles en Java**.

Sockets TCP — Conceptos y flujo

La comunicación se compone de 3 fases: 1) apertura de sockets, 2) uso de streams, 3) cierre.

Apertura

- Servidor: new ServerSocket(puerto) → accept() devuelve Socket para el cliente.
- Cliente: new Socket(host, puerto) → establece la conexión.

Streams

- InputStream / OutputStream: lectura/escritura de bytes.
- BufferedReader / PrintWriter: texto por líneas.
- DataInputStream / DataOutputStream: tipos primitivos.
- ObjectInputStream / ObjectOutputStream: objetos (TCP).

Cierre

- Cada extremo llama a socket.close(); el ServerSocket sigue aceptando nuevas conexiones.

Sintaxis rápida — Clase Socket

- Socket(String host, int port)
- InetAddress getInetAddress() / getLocalAddress()
- int getPort() / getLocalPort()
- InputStream getInputStream() / OutputStream getOutputStream()
- void close()

Sintaxis rápida — Clase ServerSocket

- ServerSocket(int port)
- Socket accept()
- void close()

Ejemplo Cliente TCP (eco)

```
// Cliente TCP (eco)
try (Socket client = new Socket("127.0.0.1", 4321);
     PrintWriter out = new PrintWriter(client.getOutputStream(), true);
     BufferedReader in = new BufferedReader(new InputStreamReader(client.getInputStream()));
     BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in))) {
    String input;
    while ((input = stdIn.readLine()) != null) {
        out.println(input);
        System.out.println("echo: " + in.readLine());
    }
} catch (UnknownHostException e) {
    System.err.println("Host desconocido: " + e.getMessage());
} catch (IOException e) {
    System.err.println("Error de E/S: " + e.getMessage());
}
```

Ejemplo Servidor TCP (eco)

```
// Servidor TCP (eco)
try (ServerSocket serverSocket = new ServerSocket(4321)) {
    System.out.println("Servidor inicializado");
    try (Socket clientSocket = serverSocket.accept();
```

```
         PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
         BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStr
        System.out.println("Aceptada nueva conexión");
        String inputLine;
        while ((inputLine = in.readLine()) != null) {
            out.println(inputLine);
            System.out.println("recibido: " + inputLine);
        }
    }
} catch (IOException e) {
    System.out.println("Excepción: " + e.getMessage());
}
```

Diagrama de secuencia — Eco TCP

```
Cliente              Servidor
  |    connect()  ->   |
  |------------------->|
  |    "hola"          |
  |==== send ========> |
  | <==== echo ======= |
  |    "hola"          |
  |                    |
  |---- close() ---->  |
```

Petición HTTP por Socket

```
try (Socket s = new Socket(InetAddress.getByName("www.uma.es"), 80)) {
    InputStream in = s.getInputStream();
    OutputStream os = s.getOutputStream();
    PrintWriter out = new PrintWriter(new OutputStreamWriter(os));
    out.println("GET / HTTP/1.1■");
    out.println("Host: www.uma.es■");
    out.println("■");
    out.flush();
    BufferedReader br = new BufferedReader(new InputStreamReader(in));
    String line;
    while ((line = br.readLine()) != null) System.out.println(line);
}
```

Buenas prácticas TCP

- Usar try-with-resources para cerrar automáticamente.
- Definir un protocolo de mensajes claro (delimitadores/tamaños).
- Configurar SO_TIMEOUT para evitar bloqueos indefinidos en read().
- Usar UTF-8 al convertir texto↔bytes.

Errores comunes TCP

- Olvidar flush() en PrintWriter.
- Suponer que readLine() nunca devuelve null (cierre remoto).
- Ignorar que write() puede necesitar múltiples llamadas en binario.

Clase InetAddress

- static InetAddress getByName(String host)
- static InetAddress[] getAllByName(String host)
- static InetAddress getLocalHost()
- boolean isReachable(int timeout)
- String getHostAddress()

Nota: si no se especifica IP al crear un socket en el servidor, se usa 0.0.0.0 (wildcard). Es buena práctica limitar la IP de escucha.

TCP Multihilo (Thread/ThreadPool)

Worker Runnable

```
class ServidorWorker implements Runnable {
    private final Socket cliente;
    public ServidorWorker(Socket so) { this.cliente = so; }
    @Override public void run() {
        try {
            // ... operar con streams ...
        } catch (IOException e) {
            System.out.println(e);
        } finally {
            try { cliente.close(); } catch (IOException ignore) {}
        }
    }
}
```

Servidor: una hebra por cliente

```
try (ServerSocket s = new ServerSocket(4321)) {
    while (true) {
        Socket clientSocket = s.accept();
        new Thread(new ServidorWorker(clientSocket)).start();
    }
}
```

Servidor con ThreadPool

```
ExecutorService pool = Executors.newFixedThreadPool(10);
try (ServerSocket ss = new ServerSocket(4321)) {
    while (true) {
        Socket clientSocket = ss.accept();
        pool.execute(new ServidorWorker(clientSocket));
    }
} finally { pool.shutdown(); }
```

Buenas prácticas Multihilo

- Cerrar el socket en finally.
- Limitar el número de hilos con un pool.
- Sincronizar accesos a recursos compartidos (ficheros/contadores).

Errores comunes Multihilo

- Crear hilos ilimitados (riesgo de OOM).
- Escritura concurrente sin synchronized (p. ej., chat).
- Olvidar pool.shutdown().

TCP Asíncrono (java.nio.channels)

Modelo no bloqueante con AsynchronousServerSocketChannel/AsynchronousSocketChannel.
Dos enfoques: Future y CompletionHandler. Se trabaja con ByteBuffer.

Servidor asíncrono (CompletionHandler)

```
try (AsynchronousServerSocketChannel server = AsynchronousServerSocketChannel.open()) {
    server.bind(new InetSocketAddress("127.0.0.1", 2345));
    server.accept(null, new CompletionHandler<AsynchronousSocketChannel, Object>() {
        @Override public void completed(AsynchronousSocketChannel ch, Object att) {
            if (server.isOpen()) server.accept(null, this);
            if (ch != null && ch.isOpen()) {
                ByteBuffer buffer = ByteBuffer.allocate(1024);
                try {
                    Future<Integer> f = ch.read(buffer);
                    f.get();
                    buffer.flip();
                    ch.write(buffer).get();
                    buffer.clear();
                } catch (Exception e) { e.printStackTrace(); }
            }
        }
        @Override public void failed(Throwable exc, Object att) { exc.printStackTrace(); }
    });
}
```

Cliente asíncrono (Future)

```
try (AsynchronousSocketChannel client = AsynchronousSocketChannel.open()) {
    client.connect(new InetSocketAddress("127.0.0.1", 2345)).get();
    String str = "Hola servidor Asíncrono!!";
    ByteBuffer buffer = ByteBuffer.wrap(str.getBytes());
    client.write(buffer).get();
    buffer.flip();
    client.read(buffer).get();
    System.out.println(new String(buffer.array()).trim());
    buffer.clear();
}
```

Diagrama de secuencia — NIO

```
Cliente                      Servidor (Asíncrono)
  open()                        open()
  connect() ---- Future --->    accept() -- CompletionHandler --> channel
  write(ByteBuffer) --F-->      read(ByteBuffer) --F--> flip(); write()
  read(ByteBuffer)  --F-->
```

Buenas prácticas NIO

- Llamar a flip() antes de leer del ByteBuffer y clear() al terminar.
- Re-llamar a accept() dentro de completed() para seguir aceptando clientes.
- Evitar get() bloqueante si se busca paralelismo máximo: usar CompletionHandler.

Errores comunes NIO

- Olvidar flip()/clear() (datos corruptos).
- No manejar excepciones dentro del handler.

Sockets UDP

Sin conexión; se usan DatagramSocket/DatagramPacket; se trabaja con byte[].

Sintaxis rápida

- DatagramSocket() / DatagramSocket(int port)
- void send(DatagramPacket p) / void receive(DatagramPacket p)
- DatagramPacket(byte[] buf, int len[, InetAddress addr, int port])
- int getLength(), byte[] getData(), InetAddress getAddress(), int getPort()

Cliente UDP (PING)

```
try (DatagramSocket socket = new DatagramSocket()) {
    byte[] data = "PING!".getBytes(StandardCharsets.UTF_8);
    DatagramPacket p = new DatagramPacket(data, data.length, InetAddress.getLocalHost(), 4321)
    socket.send(p);
    byte[] buf = new byte[256];
    DatagramPacket resp = new DatagramPacket(buf, buf.length);
    socket.receive(resp);
    String res = new String(resp.getData(), 0, resp.getLength(), StandardCharsets.UTF_8);
    System.out.println("Servidor: " + res);
}
```

Servidor UDP (PONG)

```
try (DatagramSocket socket = new DatagramSocket(4321)) {
    byte[] buffer = new byte[256];
    while (true) {
        DatagramPacket p = new DatagramPacket(buffer, buffer.length);
        socket.receive(p);
        String req = new String(p.getData(), 0, p.getLength(), StandardCharsets.UTF_8);
        System.out.println("Cliente dijo: " + req);
        byte[] reply = "PONG!".getBytes(StandardCharsets.UTF_8);
        DatagramPacket r = new DatagramPacket(reply, reply.length, p.getAddress(), p.getPort()
        socket.send(r);
    }
}
```

Diagrama de secuencia — UDP PING/PONG

```
Cliente                Servidor
 send("PING")  --->      receive()
               <---      send("PONG")
 receive()
```

Buenas prácticas UDP

- Usar UTF-8 para conversión texto↔bytes.
- Usar getLength() para truncar al tamaño real del datagrama.
- Evitar paquetes grandes (riesgo de fragmentación/pérdida).

Errores comunes UDP

- Suponer entrega garantizada.
- Ignorar dirección/puerto remitente al responder.

UDP Multicast

Envía a múltiples destinos (IPs clase D: 224.0.0.1–239.255.255.255). Clase MulticastSocket.

Ejemplo Multicast

```
try (MulticastSocket socket = new MulticastSocket(4446)) {
    InetAddress group = InetAddress.getByName("224.0.0.3");
    socket.joinGroup(group);
    String msg = "Hola grupo!";
    DatagramPacket hello = new DatagramPacket(msg.getBytes(), msg.length(), group, 4446);
    socket.send(hello);
    byte[] buf = new byte[1000];
    DatagramPacket recv = new DatagramPacket(buf, buf.length);
    socket.receive(recv);
    String res = new String(recv.getData(), 0, recv.getLength(), StandardCharsets.UTF_8);
    System.out.println(res);
    socket.leaveGroup(group);
}
```

Sockets multiplexados (Selector)

Solución intermedia entre síncrono y asíncrono. Usa canales no bloqueantes y Selector para detectar operaciones listas.

Ejemplo con Selector

```
Selector sel = Selector.open();
ServerSocketChannel ssc = ServerSocketChannel.open();
ssc.bind(new InetSocketAddress(4321));
ssc.configureBlocking(false);
ssc.register(sel, SelectionKey.OP_ACCEPT);

while (true) {
    sel.select();
    Set<SelectionKey> listas = sel.selectedKeys();
    for (SelectionKey key : listas) {
        if (key.isAcceptable()) {
            SocketChannel sc = ssc.accept();
            sc.configureBlocking(false);
            sc.register(sel, SelectionKey.OP_READ);
        } else if (key.isReadable()) {
            SocketChannel sc = (SocketChannel) key.channel();
            ByteBuffer buf = ByteBuffer.allocate(1024);
            int n = sc.read(buf);
            if (n == -1) { key.cancel(); sc.close(); }
            else { buf.flip(); sc.write(buf); buf.clear(); }
        }
    }
    listas.clear();
}
```

Buenas prácticas Selector

- Configurar channels en no bloqueante antes de registrar.
- Vaciar selectedKeys() con clear() tras iterar.
- Mantener buffers por conexión.

Errores comunes Selector

- Olvidar finishConnect() para clientes no bloqueantes (OP_CONNECT).
- Usar streams bloqueantes en lugar de *Channel.

Parámetros de transporte / OS

- SO_RCVBUF: get/setReceiveBufferSize() — tamaño buffer de recepción.
- SO_SNDBUF: get/setSendBufferSize() — tamaño buffer de envío.
- SO_REUSEADDR: get/setReuseAddress() — reutilizar puerto (bind rápido, multicast).
- SO_LINGER: get/setSoLinger() — tiempo de espera al cerrar.
- SO_TIMEOUT: get/setSoTimeout() — límite de espera en read().
- TCP_NODELAY: get/setTcpNoDelay() — desactiva Nagle (reduce latencia).