



# Networking

--- Java Advanced Tutorial , Module 3

ALGOM High-End IT Training , Java Advanced

# 内容提要

Network / IP / Port

InetAddress

Uniform Resource Locator

TCP / ServerSocket / Socket

使用 NIO 实现无阻塞式网络通信

UDP / DatagramSocket/DatagramPacket

MulticastSocket

# 网络基础知识

## ■ 什么是计算机网络

- 把分布在不同地理区域的计算机通过专门的通信线路连接在一起形成的规模庞大的、功能强大的网络系统
- 一个非常著名的网络：万维网（World Wide Web，即 www）

## ■ 计算机网络的作用

- 资源共享
- 信息传输和集中处理
- 均衡负荷与分布处理
- 综合信息服务

# 网络基础知识

## ■ 计算机网络的分类

- 按照网络结构划分
  - ◆ 星型网络、总线型网络、环线型网络、树形网络、星型环线网络
- 按照介质来分
  - ◆ 双绞线网络、同轴电缆网络、光纤网、无线网、电力线网
- 按照规模来划分
  - ◆ 局域网( LAN )、城域网( MAN )、广域网( WAN )

# 网络基础知识

## ■ 网络通信协议

### ■ OSI

- ◆ OSI是Open System Interconnect的缩写，意为开放式系统互联
- ◆ 国际标准组织（国际标准化组织）制定了OSI模型
- ◆ 这个模型把网络通信的工作分为7层，分别是
  - 物理层、数据链路层、网络层、传输层、会话层、表示层和应用层

### ■ TCP/IP

- ◆ IP是英文 Internet Protocol（网络之间互连的协议）的缩写
  - 中文简称为“网协”，是为计算机网络相互连接进行通信而设计的协议
- ◆ TCP: Transmission Control Protocol 传输控制协议
  - TCP是一种面向连接（连接导向）的、可靠的、基于字节流的运输层（Transport layer）通信协议，由IETF的RFC 793说明（specified）
  - 在简化的计算机网络OSI模型中，它完成第四层传输层所指定的功能
  - UDP是同一层内另一个重要的传输协议

# 网络基础知识

## OSI分层模型与TCP/IP分层模型



# 网络基础知识

## IP地址

- IP 地址是根据 IP 协议为某个通信实体分配的地址
  - ◆ 这个通信实体，有可能是计算机、打印机、路由器的某个端口
- IP地址是数字型的，是个 32 位 (32bit) 的整数
  - ◆ 通常为了方便记忆，把这 32 位分成 4 个部分，每个部分存放 8 位
  - ◆ 因此我们实际上看到的 ip 地址形式为：192.168.95.27
- NIC 负责全球的 IP 地址的规划、管理
  - ◆ NIC : Internet Network Information Center
  - ◆ NIC 下属 Inter NIC 、 APNIC 、 PIPE 机构负责美国及其他地区的 IP 地址分配
  - ◆ APNIC 总部在日本东京大学，负责亚太地区的IP地址分配
- IP地址被分成 A 、 B 、 C 、 D 、 E 五类
  - ◆ A类 : 10.0.0.0 ~ 10.255.255.255
  - ◆ B类 : 172.16.0.0 ~ 172.31.255.255
  - ◆ C类 : 192.168.0.0 ~ 192.168.255.255

# 网络基础知识

## IP地址和端口 (port)

- 一个特殊的 IP 地址
  - ◆ 127.0.0.1 : 代表当前计算机本身
- 为什么要用到端口
  - ◆ 一个 IP 地址可以唯一地确定一个通信实体
  - ◆ 一个通信实体上可以有多个程序提供服务
    - ◆ 比如一台服务器上可以有多个DBMS, 如 MySQL、DB2、Oracle
  - ◆ 为了区别同一个通信实体上的不同服务(程序), 还要使用端口

## 端口

- 端口是一个 16 bit 的整数, 用于表示数据交给哪个通信程序处理
- 端口是应用程序与外界交流的出入口, 是种抽象的软件结构

# 网络基础知识

## 端口 (port) 分类

- 不同的应用程序处理不同端口上的数据
  - ◆ 同一个计算机上不允许有两个以上程序使用同一个端口
- 端口的范围从 0 到 65535 , 被分成三类
  - ◆ 公认端口 : 从 0 到 1023
    - 他们紧密绑定一些特定服务, 如 80 端口、23 端口、21 端口等等
  - ◆ 注册端口 : 从 1024 到 49151
    - 松散地绑定一些服务, 比如
      - » Oracle 数据库默认的端口是 1521
      - » MySQL 数据库默认的端口是 3306
      - » Tomcat 默认的端口是 8080
  - ◆ 动态或私有端口 : 从 49152 到 65535
    - 应用程序使用的动态端口, 应用程序一般不会主动去使用这些端口

我们自己在使用端口时, 尽量使用 1024 以上的端口,  
同时还要注意避开已经被已有的服务占用的端口

# InetAddress

## InetAddress 用来代表 IP 地址

- 它有两个子类
  - ◆ InetAddress : 对应 IPv4 地址
  - ◆ Inet6Address : 对应 IPv6 地址
- 没有构造，通过以下静态方法获取该类的实例
  - ◆ `getByName( String hostName ) :`  
根据指定主机名称得到对应的 InetAddress 实例
  - ◆ `getByAddress( byte[] address ) :`  
根据指定的 IP 地址获取对应的 InetAddress 实例

# InetAddress

## InetAddress 用来代表 IP 地址

### 常用方法

- ◆ String getCanonicalHostName()
  - 获取此 IP 地址的完全限定域名
- ◆ String getHostAddress()
  - 返回 IP 地址字符串（以文本表现形式）
- ◆ String getHostName()
  - 获取此 IP 地址的主机名
- ◆ static InetAddress getLocalHost()
  - 返回本地主机对应的 InetAddress 实例
- ◆ boolean isReachable(int timeout)
  - 测试是否可以达到该地址

# URL

## ■ URL 对象代表 Uniform Resource Locator 统一资源定位符

- 它是指向互联网"资源"的指针
  - ◆ 这里的资源可以是简单的文件或目录
  - ◆ 也可以是更为复杂的对象（比如数据库对象）
- URL 通常由 协议名、主机名、端口和资源组成，格式如下
  - ◆ protocol : // host : port / resourceName
  - ◆ 常见的 URL 如: http://www.baidu.com:80/index.html

## ■ URL 类的构造

- 该类有很多重载的构造
  - ◆ 但不外乎就是指定 协议名、主机名、端口、资源名等参数
- 可以根据已有的 URL 创建全新的 URL 对象
  - ◆ URL(URL context, String spec)
  - ◆ URL(URL context, String spec, URLStreamHandler handler)

# URL

## URL 类中常用方法

- String getFile() 获取此 URL 的文件名
- String getHost() 获取此 URL 的主机名（如果适用）
- String getPath() 获取此 URL 的路径部分
- int getPort() 获取此 URL 的端口号
- String getProtocol() 获取此 URL 的协议名称
- String getQuery() 获取此 URL 的查询部分
- URLConnection openConnection()
  - ◆ 返回一个 URLConnection 对象，它表示到 URL 所引用的远程对象的连接
- InputStream openStream()
  - ◆ 打开到此 URL 的连接并返回一个用于从该连接读入的 InputStream

完成一个基于 URL 的多线程下载工具 (http协议版)

# TCP 协议

## TCP 协议

- 使用 IP 协议可以将一个消息从一个通信主体发送到另一个通信主体，消息在传输过程中被分割成一个一个的小数据包
- 虽然解决了数据发送和接受问题，但是不能解决数据分组在传输过程中可能出现的问题
- 为了解决以上问题，需要提供一整套保障无差错通信的措施或协议，这就是目前使用广泛的 TCP 协议
- TCP 协议被称作端对端协议
  - ◆ 通过该协议建立了两个通信实体之间用于发送和收取数据的虚拟链路

## TCP 协议

### TCP 协议的可靠性

- TCP 协议负责收集被分割的数据分组，并将其按照适当的次序发送；对方在接受到数据分组后再将其正确还原
- 为了保证数据包传送中准确无误，TCP 协议使用重发机制
  - ◆ 通信实体A发送一个消息给通信实体B后，会等待通信实体B返回的确认信息，如果A没有收到B的确认信息，则A会再次发送该消息
  - ◆ 这种重发机制，保证了数据传输的可靠性和完整性

# ServerSocket

## ■ 使用 ServerSocket 创建 TCP 服务器端

- ServerSocket 类的对象用于监听来自其它通信实体的连接
  - ◆ 该类的 accept() 方法用于监听来自外部的连接
  - ◆ 如果没有任何通信实体连接该 ServerSocket 对象，它将永远等待下去
- 获得 ServerSocket 对象
  - ◆ 该类中提供了许多重载的构造方法，可以用于实例化ServerSocket
  - ◆ 一般而言需要指定 IP 和 port

# Socket

## ■ 获得 Socket 进行通信

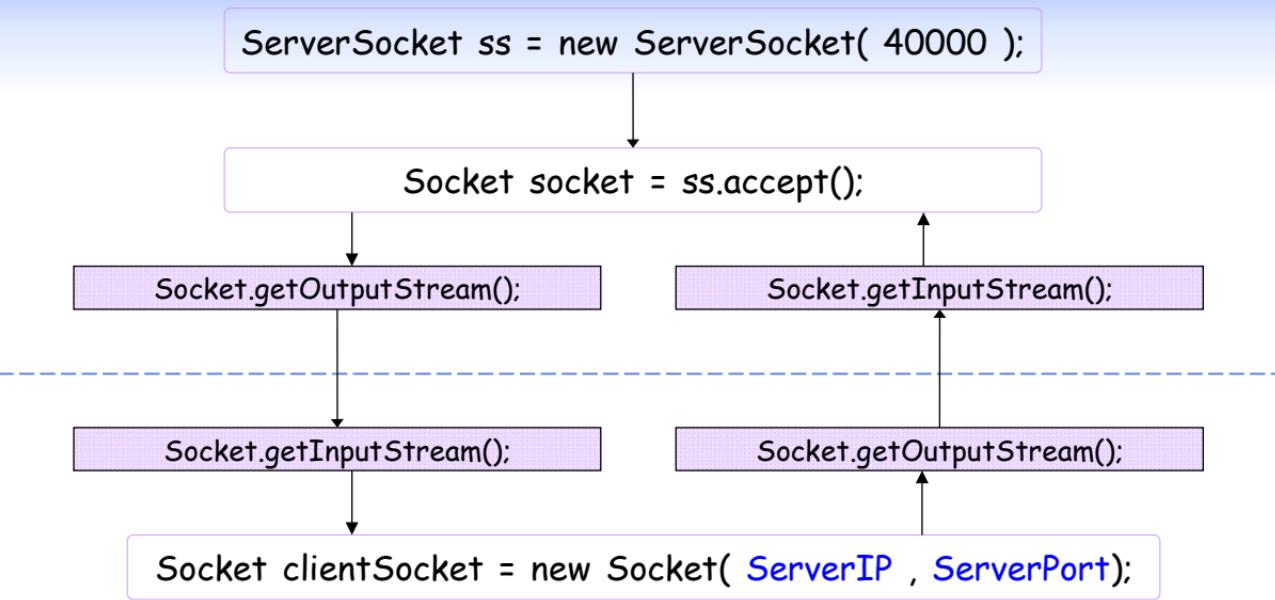
- ServerSocket 实例只负责监听来自其它通信实体的连接
  - ◆ 如果 accept() 方法监听到来自外部的通信实体的连接，它将返回一个 Socket
  - ◆ Socket accept() : 返回监听到的连接对应的 Socket 的实例
- 使用 Socket 类的实例才能实现通信
  - ◆ 该类中有两个非常重要的方法
    - InputStream getInputStream()
      - » 返回当前的 Socket 对应的输入流，让程序通过该流从 Socket 中读取数据
    - OutputStream getOutputStream()
      - » 返回当前的 Socket 对应的输出流，让程序通过该流向 Socket 中写入数据

# Socket

## ■ 使用 Socket 创建另外一个通信实体

- 构造一个 Socket 实例，让它去连接前面建好的 ServerSocket
- Socket 类构造有多种重载形式
  - ◆ 但一般需要指定连接哪个主机或哪个ip，另外还要指定端口号，比如：
  - ◆ `Socket s = new Socket("127.0.0.1", 8888);`

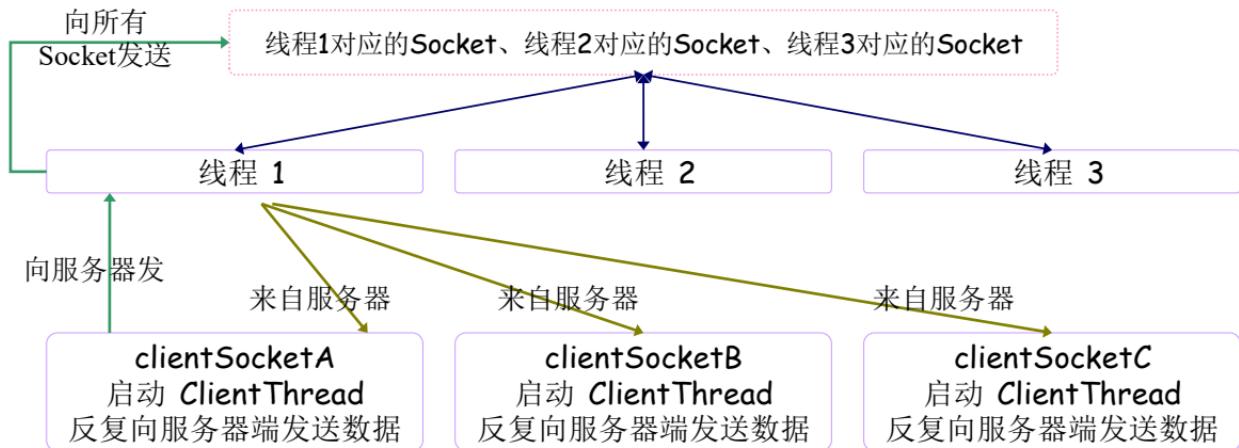
# ServerSocket / Socket 通信图



# 基于 ServerSocket / Socket 的聊天室

```
ServerSocket ss = new ServerSocket( 40000 );
```

```
Socket socket = ss.accept();  
启动 ServerThread ;
```



# 无阻塞式通信

## 阻塞式的通信

- 前述 Socket 使用阻塞式通信的特点
  - ◆ 当程序执行输入、输出操作后，在这些操作返回前会一直阻塞当前线程
  - ◆ 服务器必须为每个客户端提供独立的线程做支持
  - ◆ 服务器往往需要处理大量的客户端请求
    - 每个客户端一个线程，导致服务器负荷太重
- 基于阻塞式的通信效率较差
  - ◆ 耗费资源：启动线程和销毁线程
  - ◆ 等待时间长：只有有返回，线程才继续走下去

## 无阻塞式通信

- JDK 1.4 之后提供了 NIO 来开发高性能无阻塞的服务器
- 使用 NIO 使服务器用**极有限**个线程即可同时处理所有客户端

# NIO API中关于 **Socket** 通信的核心

## ■ Selector 类

- 它是 SelectableChannel 的多路复用器
  - ◆ 采用非阻塞方式进行通信的 Channel 都应该注册到 Selector 对象
  - ◆ 通过该类的静态方法 open( ) 可以获得该类的一个实例
- Selector 对象是非阻塞IO的核心
  - ◆ Selector 可以同时监控多个 SelectableChannel 的 IO 状况，是整个非阻塞 IO 的核心

# NIO API中关于 Socket 通信的核心

## ■ Selector 类

- 一个 Selector 实例有 3 个 SelectionKey 集合
  - ◆ 所有的 SelectionKey 的集合
    - 代表所有注册在当前 Selector 对象上的 Channel 对象
    - 通过 keys() 方法可以得到该集合
  - ◆ 被选择的 SelectionKey 的集合
    - 代表所有可以通过 select() 检测到，需要进行 IO 处理的 Channel
    - 通过 selectedKeys() 方法可以获取该集合
  - ◆ 被取消的 SelectionKey 的集合
    - 所有被取消注册的 Channel 的集合
    - 下次执行 select() 方法时，这些 Channel 对应的 SelectionKey 将彻底删除
    - 程序通常无法访问该集合

# NIO API中关于 **Socket** 通信的核心

## ■ Selector 类

- 另外，该类中还提供了一些跟 select() 相关的方法：
  - ◆ int select() : 监控所有注册的 Channel
    - 当已注册的 Channel 中有需要处理的IO操作时，该方法返回
    - 并将对应的 SelectionKey 加入被选择的 SelectionKey 集合中
    - 最后该方法返回这些 Channel 的数量
  - ◆ int select( long timeout ) : 可以设置超时时长的 select() 操作
  - ◆ int selectNow() : 执行一个立即返回的 select() 操作
    - 相对于无参的 select() 来说，该方法不会阻塞线程
  - ◆ Selector wakeup() : 使一个还未返回的 select() 方法立刻返回

# NIO API中关于 **Socket** 通信的核心

## ■ SelectableChannel

- 代表可以支持非阻塞 IO 操作的 Channel 对象
- 通过 register() 方法可以将其注册到某个 Selector 实例上
- 这种注册关系由 SelectionKey 实例来表示
- Selector 的 select() 方法，允许程序同时监控多个 IO Channel
- SelectableChannel 对象支持两种操作模式
  - ◆ 阻塞模式：使默认情况
    - 相当于 configureBlocking( true )
  - ◆ 非阻塞模式：必须指定非阻塞模式，才能使用非阻塞 IO 操作
    - 相当于 configureBlocking( false )
  - ◆ 使用 isBlocking() 可以判断该 Channel 是否是阻塞模式

# NIO API中关于 Socket 通信的核心

## ■ SelectableChannel

- 不同的 SelectableChannel 所支持的操作不一样
  - ◆ ServerSocketChannel 代表一个 ServerSocket , 只支持 OP\_ACCEPT
- SelectableChannel 的了一个方法, 返回它所支持的所有操作
  - ◆ int validOps() : 返回一个 bit mask , 表示这个 Channel 上支持的I0 操作
- SelectionKey 中定义了 4 种 IO 操作(4个静态常量)
  - ◆ OP\_READ = 1
  - ◆ OP\_WRITE = 4
  - ◆ OP\_CONNECT = 8
  - ◆ OP\_ACCEPT = 16
- validOps() 返回的值
  - ◆ 如果返回是 5 则表示当前的 Channel 支持 OP\_READ 和 OP\_WRITE

# NIO API中关于 **Socket** 通信的核心

## ■ SelectableChannel

- SelectableChannel 提供了如下方法来获取其注册状态
  - ◆ boolean isRegistered()
    - 判定该 Channel 是否已经注册在一个或多个 Selector 实例上
  - ◆ SelectionKey keyFor( Selector sel )
    - 该方法返回该 Channel 和 sel 之间的注册关系
    - 如果不存在注册关系，返回 null
- 在无阻塞通信种，主要使用 SelectableChannel 类的两个子类
  - ◆ ServerSocketChannel
  - ◆ SocketChannel

## ■ SelectionKey

- 该对象代表 SelectableChannel 和 Selector 之间的注册关系

# NIO API中关于 **Socket** 通信的核心

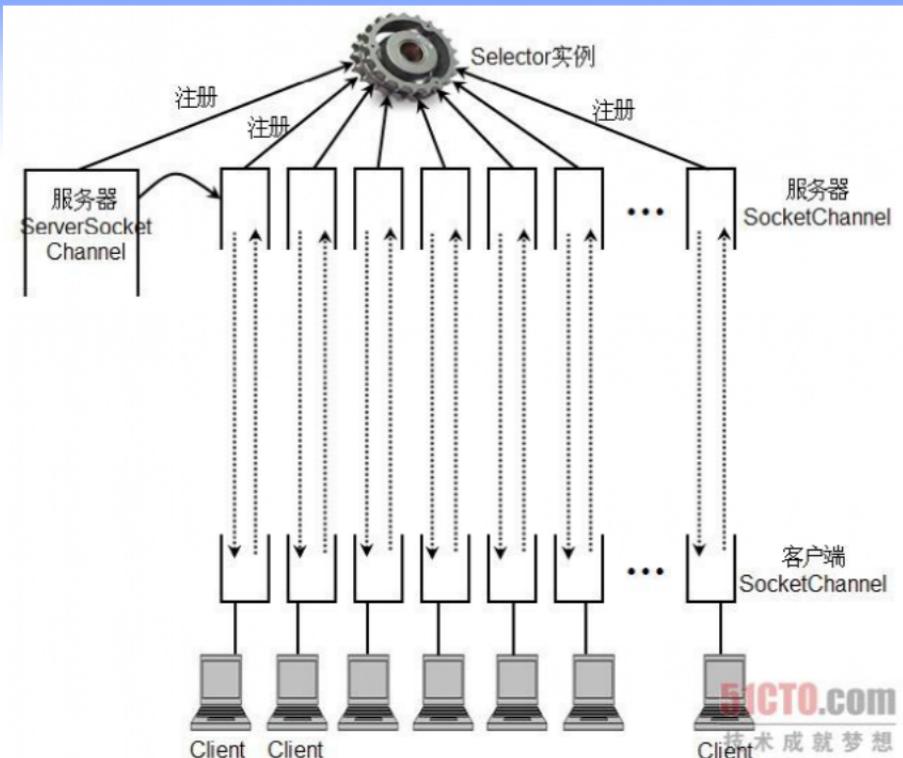
## ■ ServerSocketChannel

- 支持非阻塞操作
- 对应于 `java.net.ServerSocket` 类
- 提供了 TCP 协议的 IO 接口，支持 `OP_ACCEPT` 操作
- 提供了 `accept()` 方法，相当于 `java.net.ServerSocket` 的 `accept()`

## ■ SocketChannel

- 支持非阻塞操作
- 对应于 `java.net.Socket` 类
- 提供了 TCP 协议的 IO 接口，支持以下操作
  - ◆ `OP_CONNECT` 、 `OP_READ` 、 `OP_WRITE`
- 同时实现了以下接口
  - ◆ `ByteChanel` 、 `ScatteringByteChanel` 、 `GatheringByteChanel`
  - ◆ 因此可以直接通过 `SocketChannel` 来读写 `ByteBuffer` 对象

# 无阻塞通信示意图



# User Datagram Protocol

## ■ 数据报 ( Datagram )

### ■ 通过网络传输的数据的基本单元

- ◆ 包含一个报头 (header) 和数据本身
- ◆ 其中报头描述了数据的目的地以及和其它数据之间的关系

报头 ( header )

其它数据 ( 如果有 )

### ● 数据报工作方式的特点

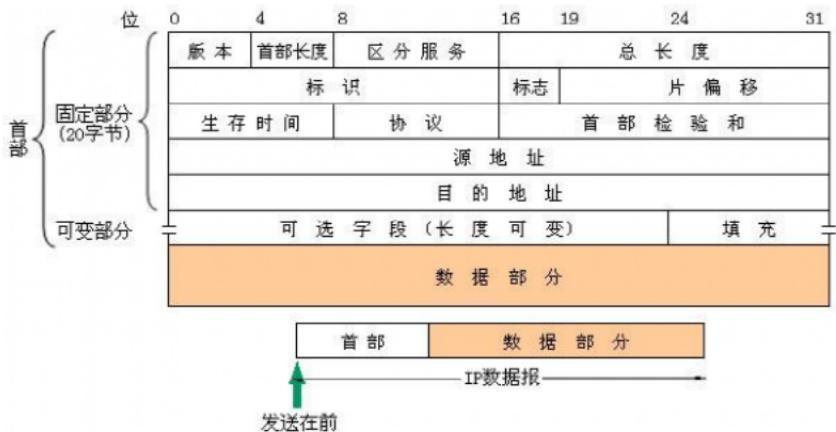
- ◆ 同一报文的不同分组可以由不同的传输路径通过通信子网
- ◆ 同一报文的不同分组到达目的结点时可能出现乱序、重复与丢失现象
- ◆ 每一个分组在传输过程中都必须带有目的地址与源地址
- ◆ 数据报方式报文传输延迟较大，适用于突发性通信，不适用于长报文、会话式通信

# User Datagram Protocol

## IP数据报 ( IP Datagram )

### TCP/IP 协议定义的在因特网上传输的包

- 这是一个与硬件无关的虚拟包，由首部和数据两部分组成
- 首部的前一部分是固定长度，共20字节，是所有IP数据报必须具有的
- 在首部的固定部分的后面是一些可选字段，其长度是可变的
- 首部中的源地址和目的地址都是IP协议地址

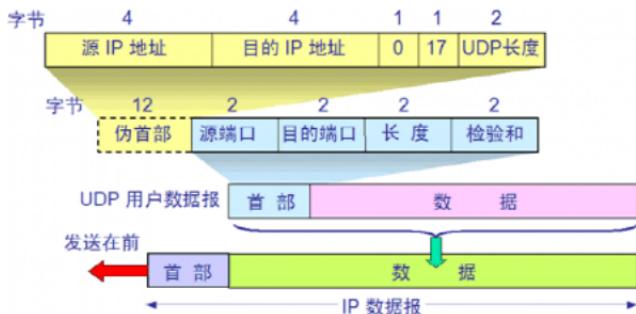


# User Datagram Protocol

## 用户数据报 (User Datagram)

### 基于 UDP 在因特网上传输的数据包

- ◆ UDP在IP数据报的头部仅仅加入了复用和数据校验
- ◆ UDP首部字段由4个部分组成，其中两个是可选的
  - 来源端口和目的端口用来标记发送和接受的应用进程
    - » 因UDP不需要应答，故来源端口是可选的，若来源端口不用，那么置为零
  - 在目的端口后面是长度固定的以字节为单位的长度域
    - » 用来指定UDP数据报包括数据部分的长度，长度最小值为8byte。
  - 首部剩下的部分是用来对首部和数据部分一起做校验和（Checksum）的
    - » 这部分是可选的，但在实际应用中一般都使用这一功能



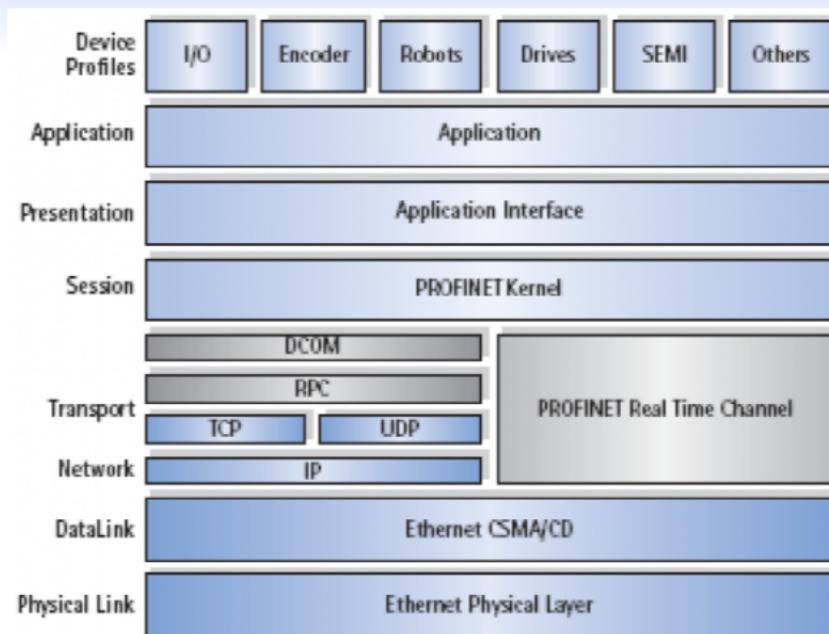
# User Datagram Protocol

## ■ 用户数据报协议 (User Datagram Protocol, UDP)

- 一个简单的面向数据报的传输层协议
  - ◆ UDP为网络层以上和应用层以下提供了一个简单的接口
  - ◆ UDP只提供数据的不可靠传递
    - 它一旦把应用程序发给网络层的数据发送出去，就不保留数据备份
    - 基于UDP的数据发送方式，没有类似于TCP的重发机制
  - ◆ UDP在IP数据报的头部仅仅加入了复用和数据校验（字段）
- IETF RFC 768 是UDP的正式规范

# User Datagram Protocol

## 用户数据报协议在网络中的地位



# User Datagram Protocol

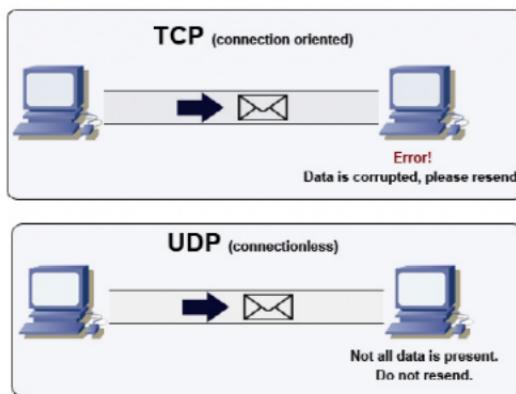
## TCP 与 UDP 的对比

### 面向连接的 TCP :

- ◆ 可靠，传输大小无限制，但是需要连接建立时间，差错控制开销大。

### 面向非连接的 UDP :

- ◆ 不可靠，差错控制开销较小，传输大小限制在64K以下，不需要建立连接



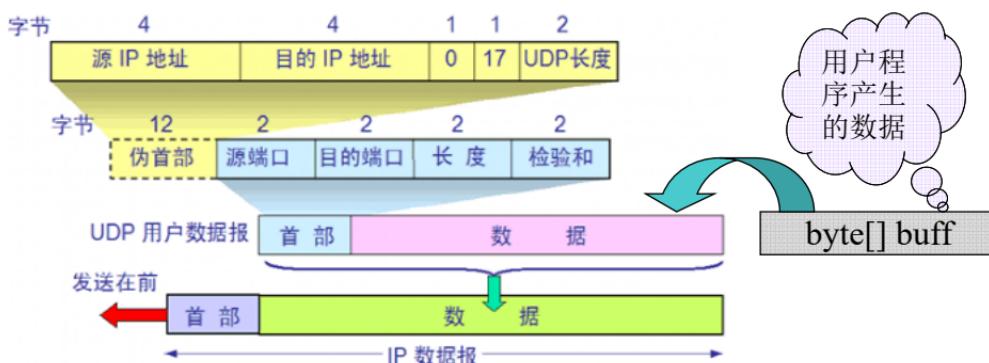
这里所说的连接，就是前面  
述及的虚拟数  
据链路

# Java 中的 UDP 通信技术

## 核心 API

### DatagramPacket (数据报包 )

- 该类的构造都需要接受一个字节数组做参数: byte[] buff
- 该字节数组即为需要传送的数据
- 该类中有很多重载的构造, 可以根据实际情况来调用



# Java 中的 UDP 通信技术

## 核心 API

### DatagramSocket

- ◆ 重载的构造比较多，根据实际情况来调用
- ◆ 常用方法：
  - bind(SocketAddress addr) 将此DatagramSocket绑定到特定的地址和端口
  - close() 关闭此 DatagramSocket
  - InetAddress getLocalAddress() 获取 DatagramSocket 绑定的本地地址
  - int getLocalPort() 返回此 DatagramSocket 绑定的本地主机上的端口号
  - void receive(DatagramPacket p) 从此 DatagramSocket 接收数据报包
  - void send(DatagramPacket p) 从此 DatagramSocket 发送数据报包

# Java 中的 UDP 通信技术

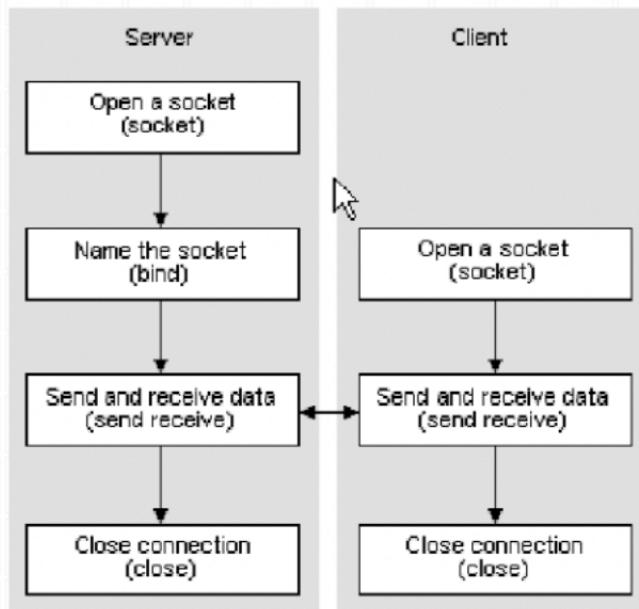
## 核心 API

### ■ DatagramChannel : UDP 的 NIO 支持

- ◆ 该类是抽象类，因此构造不能被用来创建对象
- ◆ 常用方法
  - open() : 静态方法，用于打开一个数据报通道
  - receive(ByteBuffer dst) 通过此通道接收数据报
  - send(ByteBuffer src, SocketAddress target) 通过此通道发送数据报
  - 三个重载的 write() 用于将数据报写入此通道
  - socket() 获取与此通道关联的数据报套接字
- ◆ 继承的方法
  - 该类继承了 java.nio.channels.spi.AbstractSelectableChannel 和 java.nio.channels.SelectableChannel，这两个类中的方法均可使用
- ◆ DatagramChannel 也可以注册于 Selector 实例

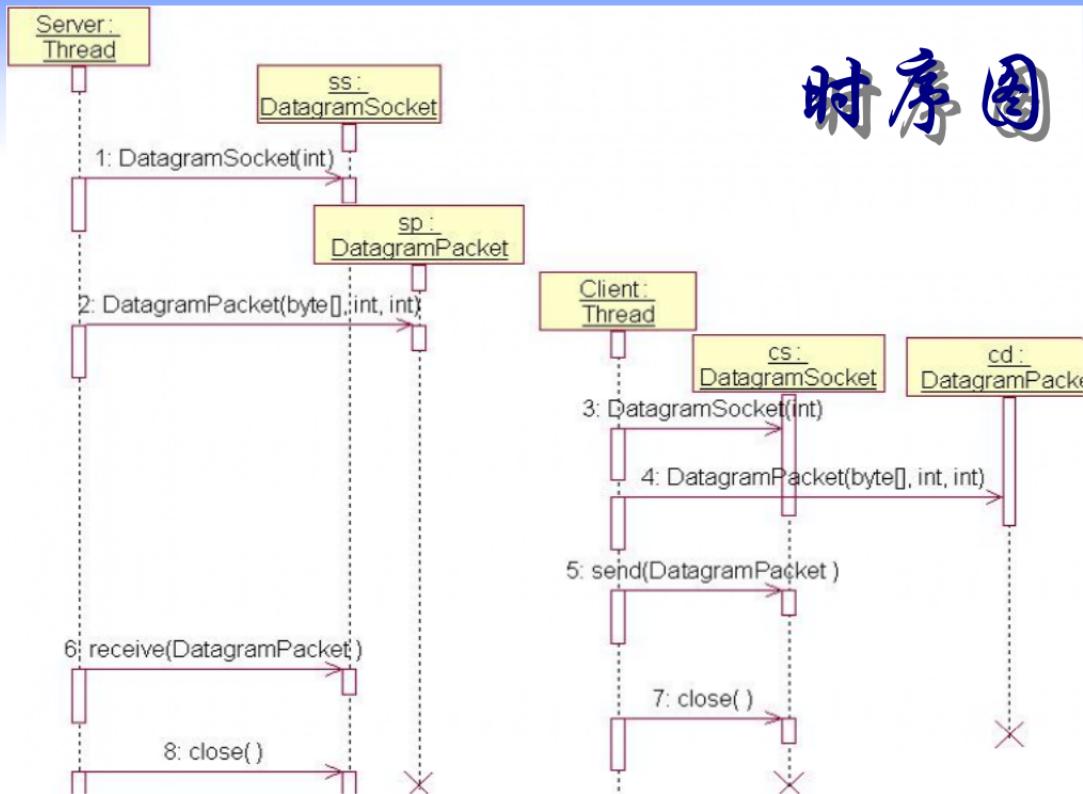
# 基于DatagramSocket的UDP通信

## UDP Server-Client关系图

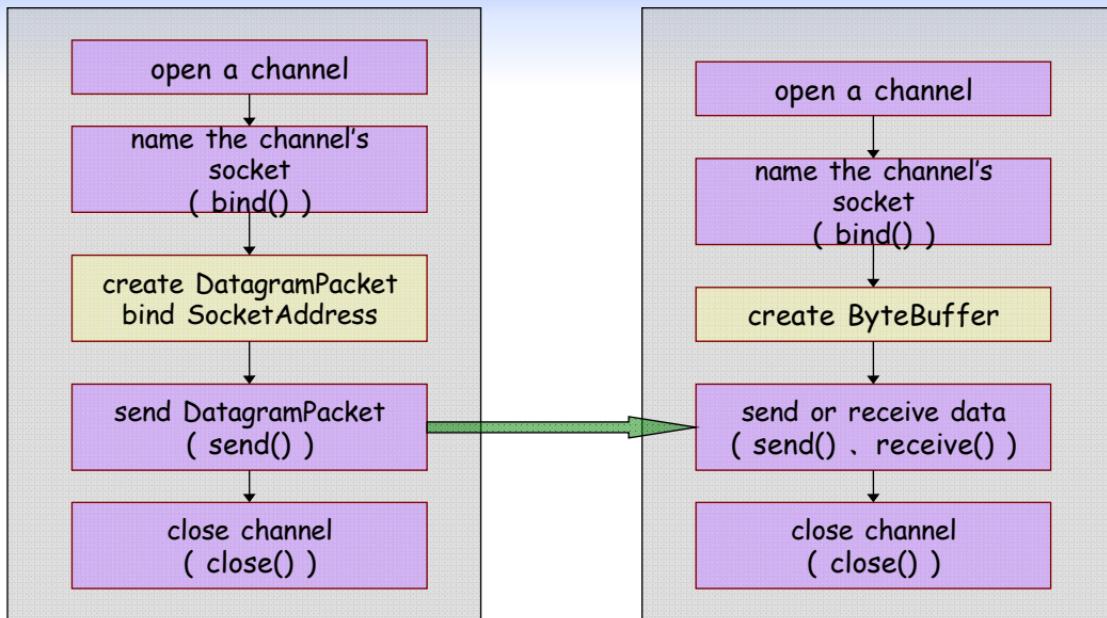


# 基于DatagramSocket的UDP通信

时序图



# 基于DatagramChannel的UDP通信



## Class MulticastSocket

- Useful for sending and receiving IP multicast packets
  - MulticastSocket 是一种 (UDP) DatagramSocket
  - 它具有加入 Internet 上其他多播主机的“组”的附加功能
  - 多播组通过 D 类 IP 地址和标准 UDP 端口号指定
    - ◆ D 类 IP 地址在 224.0.0.0 和 239.255.255.255 的范围内（包括两者）
    - ◆ 地址 224.0.0.0 被保留，不应使用

# Class MulticastSocket

## Constructors

- `MulticastSocket()`

- ◆ 使用本机默认地址、随机端口来创建 `MulticastSocket` 对象

- `MulticastSocket(int port)`

- ◆ 使用本机默认地址和指定端口号来创建 `MulticastSocket` 对象

- `MulticastSocket(SocketAddress bindaddr)`

- ◆ 创建绑定到指 `SocketAddress` 的 `MulticastSocket` 对象
  - ◆ 该 `SocketAddress` 中包含了 IP 地址 和 端口号

## Class MulticastSocket

### Methods

- void joinGroup(InetAddress mcastaddr)
  - ◆ 加入多播组
- void joinGroup( SocketAddress sa, NetworkInterface ni )
  - ◆ 加入指定接口上的指定多播组
- void leaveGroup( InetAddress ia )
  - ◆ 离开多播组
- void leaveGroup( SocketAddress sa, NetworkInterface ni )
  - ◆ 离开指定本地接口上的多播组

## Class MulticastSocket

### Methods

- int getTimeToLive()

- ◆ 获取在套接字上发出的多播数据包的默认生存时间

- void setTimeToLive(int ttl)

- ◆ 设置所发出的多播数据包的默认生存时间(用以控制多播的范围)

- ttl = 0 , 表示指定数据报应停留在本地主机
    - ttl = 1 , 表示指定数据报发送到本地局域网 (这是默认值 )
    - ttl = 32 , 表示指定数据报只能发送到本站的的网络上
    - ttl = 64 , 表示指定数据报应保留在本地区
    - ttl = 128 , 表示指定数据报应保留在本大洲
    - ttl = 255 , 表示指定数据报可以发送到 Internet 的所有地方

## Class MulticastSocket

### Methods

- `boolean getLoopbackMode()`
  - ◆ 获取多播数据报的本地回送的设置
- `void setLoopbackMode(boolean disable)`
  - ◆ 启用/禁用多播数据报的本地回送
- `NetworkInterface getNetworkInterface()`
  - ◆ 获取多播网络接口集合
- `void setNetworkInterface(NetworkInterface netIf)`
  - ◆ 指定在此套接字上发送的输出多播数据报的网络接口
- `InetAddress getInterface()`
  - ◆ 获取用于多播数据包的网络接口的地址
- `void setInterface(InetAddress inf)`
  - ◆ 设置多播网络接口，供其行为将受网络接口值影响的方法使用



海纳百川，有容乃大。壁立千仞，无欲则刚

[www.algom.cn](http://www.algom.cn)