**Commandes en Java-Réseau**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Symbole | Signification | Code | | Utilisation | | | | | |
| Adresses IP |  | java.net | |  | | | | | |
| InetAddress | permet de manipuler les "adresses internet", cad les adresses IP | address = InetAddress.getLocalHost();  address = InetAddress.getByName("localhost");  address = InetAddress.getByName("monsite.com");  address = InetAddress.getByName("127.0.0.1");  address = InetAddress.getByAddress(new byte[]{(byte)192, (byte)168, 2, 44});  UnknownHostException | | *pour obtenir toutes les adresses dispo :*  InetAddress[] addresses = InetAddress.getAllByName("monsite.com"); | | | Rq: les adresses IP sont constituées de 4 ou 6 nombres codés sous formes de byte de valeur comprise entre 0 et 255. Or un byte signé à une valeur comprise entre -128 et +127. Il faut donc caster en byte les valeurs supérieures à 127. | | |
| Inet4Address | spécialisé dans les adresses IPv4 |  | |  | | |  | | |
| Inet6Address | spécialisé dans les adresses IPv6 |  | |  | | |  | | |
| .getHostName() | renvoi le nom de l'hôte | address.getHostName(); | |  | | |  | | |
| .getCanonicalHostName() | renvoi le nom canonique de l'hôte |  | |  | | |  | | |
| .getHostAddress() | renvoi l'adresse IP sous forme de String | addresse.getHostAddress(); | | String ip = "";  for(byte b : bAddress)  ip += (b & 0xFF) + "."; | | | ATTENTION : l'instruction &0xFF permet d'obtenir la valeur non signée du byte (cad entre 0 et 255) | | |
| .getAddress() | renvoi l'adresse IP sous forme de tableau de byte | byte[] bAddress = address.getAddress(); | |  | | |  | | |
|  |  |  | |  | | |  | | |
| NetworkInterface | permet de manipuler les interfaces de d'une machine pouvant communiquer avec un réseau |  | | pour obtenir toutes les interfaces de la machines :  try {  Enumeration<NetworkInterface> netEnum = NetworkInterface.getNetworkInterfaces();  while(netEnum.hasMoreElements()) {  NetworkInterface ni = netEnum.nextElement();  Enumeration<InetAddress> addressEnum = ni.getInetAddresses();  while(addressEnum.hasMoreElements()) {  InetAddress address = addressEnum.nextElement();  instruction(s);  }  }  } catch (SocketException e) {  e.printStackTrace();  } | | | | |  |
|  |  |  | |  | | |  | | |
| Adresses URL |  |  | |  | | |  | | |
| URL | permet de manipuler les adresses URL | URL url = new URL(http://monadresseurl.com);  MalformedURLException | |  | | |  | | |
| .getAuthority() | renvoi le nom du site |  | |  | | |  | | |
| .getDefaultPort() | renvoi en Integer le port utilisé par défaut pour accéder au contenu du serveur |  | |  | | | Rq: le port peut être précisé dans une adresse URL, mais il est défini par défaut en fonction du protocole de communication utilisé (80 pour http, 20 ou 21 pour FTP, 25 pour SMTP, …) | | |
| .getHost() | renvoi le nom de l'hôte |  | |  | | |  | | |
| .getProtocol() | renvoi le type de protocole de communication utilisé |  | |  | | |  | | |
| URLEncoder | objet permettant l'encodage des caractères spéciaux dans un URL  INDISPENSABLE pour les paramètres | String monUrl = "http://monsite.fr?" + URLEncoder.encode("monparametre", "UTF-8") + "=" + URLEncoder.encode("mavaleur", "UTF-8");  UnsupportedEncodingException | |  | | |  | | |
| URLConnection | objet permettant une communication avec le contenu du site sans passer par un explorateur | URLConnection urlConn = url.openConnection();  IOException | |  | | |  | | |
| HttpURLConnection | idem spécialisé dans le protocole Http |  | |  | | |  | | |
| HttpsURLConnection | idem spécialisé dans le protocole Https | HttpsURLConnection conn = (HttpsURLConnection)url.openConnection();  IOException  ProtocolException | |  | | |  | | |
| .getContentType() | renvoie le type de contenu et l'encodage (text/html; charset=…) | String contentType = urlConn.getContentType(); | |  | | |  | | |
| .getInputStream() | permet de récupérer le contenu de la réponse sous forme de flux | String content = "", line = null;  BufferedReader buf = new BufferedReader(new InputStreamReader(urlConn.getInputStream()));  //Nous stockons les infos retournées dans une chaine  while((line = buf.readLine()) != null) {  content += line + "\n";  }  IOException | |  | | |  | | |
| .setRequestMethod( ) | spécifie le type de méthode utilisé  (par défaut : GET) | urlConn.setRequestMethod("POST"); | |  | | |  | | |
| .setDoOutput( )  .setRequestProperty( , ) | autorise la modification des paramètres de l'entête de la requête  modifie la valeur d'un paramètre | urlConn.setDoOutput(true);  urlConn.setRequestProperty("xx-forwarded-for", "ma valeur"); | |  | | |  | | |
|  |  |  | |  | | |  | | |
|  |  |  | |  | | |  | | |
| Socket | connexion en mode TCP | java.net | |  | | |  | | |
| Socket | objet permettant une connexion réseau côté client | Socket socket = null;  try {  socket = new Socket("www.monsite.com", 80);  socket = new Socket("127.0.0.1", 80);  socket = new Socket(InetAddress.getByName("www.monsite.com"), 80);  } catch (UnknownHostException e) {  } catch (IOException e) {  } finally {  if(socker != null) {  try {  socket.close();  } catch (IOException e) {  sock = null;  }  }  } | | *code permettant de tester les ports d'une machine autorisant la connexion :*  for(int i = 1; i <= 1024; i++) {  Socket soc = new Socket("127.0.0.1", i);  System.out.println("Port n° " + i + " autorisé");  } catch (UncknownHostException e) {  e.printStackTrace();  } catch (IOException e) { } | | | ATTENTION : les Socket sont des flux qui se manipulent comme ceux des fichiers, cad qu'il faut obligatoirement prendre la précaution de les refermer après les avoir utilisés ! | | |
|  | si une réponse est attendue, on peut préciser l'adresse de la machine et le port qui devra être utilisé pour cela dans le constructeur | InetAddress responseAddress = InetAddress.getByName("127.0.0.1");  new Socket("monsite.com", 80, responseAddress, 0); | |  | | | Rq: le 0 en quatrième paramètre indique que pour la réponse, n'importe quel numéro de port sera accepté. Sinon, on précise le numéro souhaité. | | |
| Proxy | utilisation d'un proxy (= intermédiaire) dans la conexion | //adresse correspondant au proxy  SocketAddress proxyAddress = new InetSocketAddress(10.10.10.10, 8080);  //Instanciation du proxy  Proxy proxy = new Proxy(Proxy.Type.SOCKS, proxyAddress);  //Création de la socket  Socket socket = new Socket(proxy);  //adresse que l'on souhaite atteindre via le proxy  SocketAddress targetAddress = new InetSocketAddress("monsite.com", 80);  //On connecte le tout  try {  socket.connect(targetAddress);  } catch (IOException e) { } | | | | Proxy.Type.DIRECT  Proxy.Type.HTTP  Proxy.Type.SOCKS | connexion directe = sans proxy  proxy Http  proxy standard dans une structure (entreprise, …) et protégé par firewall permettant l'accès à l'extérieur de la structure | | |
| .getPort() | retourne le port utilisé côté serveur | maSocket.getPort() | |  | | |  | | |
| .getLocalPort() | retourne le port utilisé côté client |  | |  | | |  | | |
| .getRemoteSocketAdress() | retourne l'adresse "Socket" (cad adresse et port) de l'hôte cible |  | |  | | |  | | |
| .getInetAdress() | retourne l'objet InetAdress et permet d'avoir accès à ses paramètres (*voir plus haut*) |  | |  | | |  | | |
| .getInputStream() | permet de récupérer la réponse sous forme de flux | BufferedInputStream bis = new BufferedInputStream(socket.getInputStream());  String content = "";  int stream;  byte[] b = new byte[1024];  while((stream = bis.read(b)) != -1) {  content += new String(b, 0, stream);  }  IOException  finally socket.close() | |  | | |  | | |
| .getOutputStream() | permet de récupérer un flux de sorti afin d'envoyer une requête | Socket socket = new Socket("monsite.com", 80);  *Mise en forme de la requête selon le protocole ----->*  BufferedOutputStream bos = new BufferedOutputStream(socket.getOutputStream());  bos.write(request.getBytes());  bos.flush(); //assure que les données sont bien écrites  *=> récup de la réponse avec .getOutputStream()*  UnknownHostException  IOException  finally socket.close() | | *Mise en forme de la requête selon le protocole*   * *HTTP :*   String request = "GET /arborescence/demapage HTTP/2.0\r\n";  request += "Host: monsite.com\r\n";  request += "\r\n";   * *FTP :*   String request1 = "USER nomUtilisateur\r\n";  String request2 = "PASS monMotDePasse\r\n"; | | | permet d'atteindre :  http://monsite.com/arborescence/demapage | | |
|  |  |  | |  | | |  | | |
| ServerSocket | objet permettant une connexion réseau côté serveur en "écoutant" le port spécifié en paramètre  pr ttes les adresses IP de la machine  pr ttes IP avec limite max de connexion  uniquement pour l'IP spécifiée, avec limite max de connexion | int port = 24, fileAttente = 100;  String IP = "192.168.10.10";  ServerSocket serverSock = new ServerSocket(port);  ServerSocket serverSock = new ServerSocket(port, fileAttente);  new ServerSocket(port, fileAttente, InetAddress.getByName(IP));  UnknownHostException  IOException | |  | | | Rq: par défaut, les serveurs limitent la connexion à 5 clients par socket si ce paramètre n'est pas spécifié. | | |
| .accept() | la socket attend une connexion cliente  *(lorsque celle-ci est détectée, elle transmet la Socket client à un nouveau thread pour son traitement pendant que la* ServerSocket *continue l'écoute, si le nombre limite de connexion n'est pas dépassé)* |  | *//Ecoute dans une boucle infinie, donc dans un thread à part !*  Thread ecouteThread = new Thread(new Runnable() {  public void run() {  while(isRunning) {  try {  *//Attente de la connexion d'un client*  Socket clientSocket = serverSock.accept();  *//si client détecté, on confi le traitement de sa requète à un nouveau Thread*  Thread traitement = new Thread(new MonProcessusTraitementClient(clientSocket));  traitement.start();  } catch (IOException e) {  e.printStackTrace();  }  }  *//sortie de la boucle lorsque écoute interrompue*  try {  serveurSocket.close();  } catch (IOException e)  e.printStackTrace();  serverSocket = null;  }  }  });  ecouteThread.start(); | | | | |  | |
|  |  |  | |  | | |  | | |
| Datagram | Connexion en mode UDP |  | |  | | |  | | |
| DatagramPacket | gère les données à envoyer ou recevoir via le protocole UDP (datagrammes = paquets UDP)  réception  envoi | byte[] buffer = new byte[1024];  int lenght = 8192, offset = 20, port = 2345;  new DatagramPacket(buffer, length);  new DatagramPacket(buffer, offset, length);  new DatagramPacket(buffer, length, inetAddress, port);  new DatagramPacket(buffer, offset, length, inetAddress, port);  new DatagramPacket(buffer, length, socketAddress, port);  new DatagramPacket(buffer, offset, length, socketAddress, port);  SocketException | | |  | | ATTENTION : la plupart des plateformes limites la tailles des Datagrammes à 8 ko (= 8192 bytes) !  Rq: avec offset, le transfert de donné commence à l'emplacement buffer[offset] et non pas buffer[0] | | |
| .getAddress() | retourne l'adresse de l'émetteur sous forme d'InetAddress | InetAddress expediteur = monDatagramPacket.getAddress(); | |  | | |  | | |
| .getSocketAddress() | retourne l'adresse et le port de l'emmeteur sous forme de SocketAddress |  | |  | | |  | | |
| .getPort() | retourne le port de l'éméteur sous forme int | int port = monDatagramPacket.getProt(); | |  | | |  | | |
| .getData() | retourne le tableau de byte correspondant aux données su datagramme | byte[] data = monDatagramPacket.getData(); | |  | | |  | | |
| .setLenght( ) | réinitialise la taille du tampon après chaque envoi ou réception | monDatagramPacket.setLenght(buffer.lenght); | |  | | |  | | |
|  |  |  | |  | | |  | | |
| DatagramSocket | permet d'émettre ou de recevoir les datagrammes  recherche auto du port d'émission  spécifie port d'écoute  spécifie port et adresse d'écoute/envoi  idem  permet d'utiliser une implémentation de DatagramSocket personnalisée | int port = 2345;  new DatagramSocket();  new DatagramSocket(port);  new DatagramSocket(port, inetAddress);  new DatagramSocket(socketAddress);  new DatagramSocket(maDatagramSocketImpl);  SocketException | |  | | |  | | |
| .send( ) | envoi le paquet passé en paramètre | maDatagramSocket.send(monDatagramPacket); | |  | | |  | | |
| .receive( ) | reçois un paquet depuis l'extérieur | maDatagramSocket.receive(monDatagramPacket); | |  | | | ATTENTION : bloque le Thread jusqu'à réception d'un datagramme ! | | |
|  |  |  | |  | | |  | | |
|  |  |  | |  | | |  | | |
|  |  |  | |  | | |  | | |
|  |  |  | |  | | |  | | |
|  |  |  | |  | | |  | | |

**Exemples complets de codes en Java - Réseau**

|  |  |
| --- | --- |
|  |  |
| Communication en protocole TCP  envoie de commande et réception des réponses en TCP | Classe métier |
| /\* Modelle d'utilisation :  ConnexionServeurFTP cs = new ConnexionServeurFTP();  cs.connectToServer();    cs.sendRequest(ConnexionServeurFTP.FTPCommands.USER, "monserveurftp");  cs.passiveModeRequest(ConnexionServeurFTP.FTPCommands.LIST);  cs.sendRequest(ConnexionServeurFTP.FTPCommands.QUIT, "");  cs.disconnectToServer();  \*/  public class ConnexionServeurFTP {  public enum FTPCommands {  USER("USER ", "331"),  PASS("PASS ", "230"),  QUIT("QUIT ", "221"),  PWD("PWD ", "257"),  CWD("CWD ", "250"),  ASCII("TYPE ASCII", "200"),  PASV("PASV", "227"),  LIST("LIST", "150"),  datacheckCODE("", "226"),  connectCODE("", "220");    private String command = "";  private String checkCode = "";    FTPCommands(String command, String checkCode) {  this.command = command;  this.checkCode = checkCode;  }  public String getCommand() {  return command;  }  public String getCheckCode() {  return checkCode;  }  }      private String IP = "127.0.0.1";  private int port = 21;    private String dataIP = "";  private int dataPort = 0;      private Socket socket = null;  private BufferedInputStream reader = null;  private BufferedOutputStream writer = null;    public ConnexionServeurFTP() {  }    public ConnexionServeurFTP(String ip, int port) {  this.IP = ip;  this.port = port;  }    public String connectToServer() {  try {  this.socket = new Socket(this.IP, this.port);  this.reader = new BufferedInputStream(socket.getInputStream());  this.writer = new BufferedOutputStream(socket.getOutputStream());  } catch (UnknownHostException e) {  e.printStackTrace();  } catch (IOException e) {  e.printStackTrace();  }  return input(FTPCommands.connectCODE);  }    public String disconnectToServer() {  try {  reader.close();  writer.close();  socket.close();  } catch (IOException e) {  e.printStackTrace();  } finally {  if(reader != null || writer != null || socket != null) {  reader = null;  writer = null;  socket = null;  }  }  return "Déconnexion réussie.";  }      public String sendRequest(FTPCommands command, String value) {  String request = command.getCommand() + value + "\r\n";  try {  writer.write(request.getBytes());  writer.flush();  } catch (IOException e) {  e.printStackTrace();  }    return input(command);  }    private String input(FTPCommands command) {  String response ="";  try {  int stream;  byte[] b = new byte[4096];  stream = reader.read(b);  response = new String(b, 0, stream);  } catch (IOException e) {  e.printStackTrace();  }  if(!response.startsWith(command.getCheckCode())) {  response = "ServerFTP received bad data link information : " + response;  } else if(command.equals(FTPCommands.PASV))  obtainDataSocketParameters(response);    return response;  }      public String passiveModeRequest(FTPCommands passivCommand) {  String content = "";  try {  sendRequest(FTPCommands.PASV, "");  Socket dataSocket = new Socket(dataIP, dataPort);  BufferedInputStream dataReader = new BufferedInputStream(dataSocket.getInputStream());  BufferedOutputStream dataWriter = new BufferedOutputStream(dataSocket.getOutputStream());    sendRequest(passivCommand, "");  content = dataInput(dataReader);    content += "\n\t" + input(FTPCommands.datacheckCODE);    } catch (UnknownHostException e) {  e.printStackTrace();  } catch (IOException e) {  e.printStackTrace();  }  return content;  }    private String dataInput(BufferedInputStream dataReader) {  String content ="";  try {  int stream;  byte[] b = new byte[1024];  while((stream = dataReader.read(b)) != -1) {  content = new String(b, 0, stream);  }  } catch (IOException e) {  e.printStackTrace();  }  return content;  }    private void obtainDataSocketParameters(String response) {  int debut = response.indexOf('(');  int fin = response.indexOf(')', debut + 1);  if(debut > 0){  String dataLink = response.substring(debut + 1, fin);  StringTokenizer tokenizer = new StringTokenizer(dataLink, ",");  String ip = "";  int port = 0;  //L'adresse IP est séparée par des virgules, on les remplace donc par des points...  ip = tokenizer.nextToken() + "." + tokenizer.nextToken() + "." + tokenizer.nextToken() + "." + tokenizer.nextToken();  //Le port est un entier de type int mais cet entier est découpé en deux  //Il faut multiplier le premier nombre par 256 et l'additionner au second  //pour avoir le numéro du port ouvert par le serveur  port = Integer.parseInt(tokenizer.nextToken()) \* 256 + Integer.parseInt(tokenizer.nextToken());  this.dataIP = ip;  this.dataPort = port;  }  } |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |