java.net包简介

# Package java.net Description

Provides the classes for **implementing networking applications**.

java.net包提供了**网络应用编程的类**。Since:JDK1.0

The java.net package can be roughly divided **in two sections**:

* **A Low Level API**, which deals with the following abstractions:

### • **Addresses**, which are networking identifiers, like IP addresses.

### • **Sockets**, which are basic bidirectional data communication mechanisms.

### • **Interfaces**, which describe network interfaces.

* **A High Level API**, which deals with the following abstractions:

### • **URIs**, which represent Universal Resource Identifiers.

### • **URLs**, which represent Universal Resource Locators.

### • **Connections**, which represents connections to the resource pointed to by URLs.

# Addresses

Addresses are used throughout the **java.net** APIs as *either host identifiers, or socket endpoint identifiers.*

The **InetAddress** class is the **abstraction** representing an IP (Internet Protocol) address. It has two subclasses:

* **Inet4Address** for IPv4 addresses.
* **Inet6Address** for IPv6 addresses.

But, in most cases, there is no need to deal directly with the subclasses, as the **InetAddress** abstraction should cover most of the needed functionality.

**About IPv6**

Not all systems have support for **the IPv6 protocol**, and while the Java networking stack will attempt to detect it and use it **transparently** when available, it is also possible to disable its use with a system property. In the case where IPv6 is not available, or explicitly disabled, **Inet6Address** are not valid arguments for most networking operations any more. While methods like **InetAddress.getByName(java.lang.String)** are guaranteed not to return an Inet6Address when looking up host names, it is possible, by passing literals, to create such an object. In which case, most methods, when called with an Inet6Address will throw an Exception.

transparently英 [træns'pærəntlɪ] 美 [træns'pærəntlɪ]

adv.明亮地，显然地，易觉察地

# Sockets套接字

Sockets are means to **establish a communication link** between machines over the network. The **java.net** package provides **4 kinds of Sockets**:

### **Socket** is a **TCP client API**, and will typically be used to connect to a remote host.

### **ServerSocket** is **a TCP server API**, and will typically accept connections from client sockets.

### **DatagramSocket** is **a UDP endpoint API** and is used to send and receive datagram packets.

### **MulticastSocket** is a subclass of **DatagramSocket** used when dealing with multicast groups.

Sending and receiving with TCP sockets is done **through InputStreams and OutputStreams** which can be obtained via the **Socket.getInputStream() and Socket.getOutputStream()** methods.

客户端的Socket可以通过InputStreams和OutputStreams发送和接收数据，且该输入流和输出流通过**Socket.getInputStream() and Socket.getOutputStream()**方法获取。

# Interfaces

The **NetworkInterface** class provides APIs to browse and query all the networking interfaces (e.g. ethernet connection or PPP endpoint) of the local machine. It is through that class that you can check if any of the local interfaces is configured to support IPv6.

Note, all conforming implementations must support at least one **NetworkInterface** object, which must **either be connected to a network**, or be **a "loopback" interface** that can only communicate with entities on the same machine.

# High level API

A number of classes in the **java.net** package do provide for a much higher level of abstraction and allow for **easy access to resources** on the network. The classes are:

### **URI** is the class representing a **Universal Resource Identifier**, as specified in RFC 2396. As the name indicates, this is just an Identifier and doesn't provide directly the means to access the resource.

### **URL** is the class representing a **Universal Resource Locator**, which is both an older concept for URIs and a means to access the resources.

### **URLConnection** is created from a URL and **is the communication link** used to access the resource pointed by the URL. This abstract class will delegate most of the work to the underlying protocol handlers **like http or https**.

### **HttpURLConnection** is a subclass of URLConnection and provides some additional functionalities specific to the **HTTP protocol**.

**The recommended usage** is **to use URI to identify resources, then convert it into a URL when it is time to access the resource**. From that URL, you can either get the **URLConnection** for fine control, or get directly the **InputStream**.

Here is an example:

**URI uri = new URI("http://java.sun.com/");**

**URL url = uri.toURL();**

**InputStream in = url.openStream();**

# Protocol Handlers

As mentioned, **URL and URLConnection rely on protocol handlers** which must be present, otherwise an Exception is thrown. **This is the major difference with URIs which only identify resources, and therefore don't need to have access to the protocol handler**. So, while it is possible to create an URI with any kind of protocol scheme (e.g. myproto://myhost.mydomain/resource/), a similar URL will try to instantiate the handler for the specified protocol; if it doesn't exist an exception will be thrown.

By default the protocol handlers are loaded dynamically from the default location. It is, however, possible to add to the search path by setting the java.protocol.handler.pkgs system property. For instance if it is set to myapp.protocols, then the URL code will try, in the case of http, first to load myapp.protocols.http.Handler, then, if this fails, **http.Handler** from the default location.

Note that **the Handler class** has to be a subclass of the abstract class **URLStreamHandler**.

Additional Specification:*Networking System Properties*