Package java.net

Provides the classes for implementing networking applications.

See: Description

Interface Summary

Interface	Description
ContentHandlerFactory	This interface defines a factory for content handlers.
CookiePolicy	CookiePolicy implementations decide which cookies should be accepted and which should be rejected.
CookieStore	A CookieStore object represents a storage for cookie.
DatagramSocketImplFactory	This interface defines a factory for datagram socket implementations.
FileNameMap	A simple interface which provides a mechanism to map between a file name and a MIME type string.
ProtocolFamily	Represents a family of communication protocols.
SocketImplFactory	This interface defines a factory for socket implementations.
SocketOption<T>	A socket option associated with a socket.
SocketOptions	Interface of methods to get/set socket options.
URLStreamHandlerFactory	This interface defines a factory for URL stream protocol handlers.

Class Summary

Class	Description
Authenticator	The class Authenticator represents an object that knows how to obtain authentication for a network connection.
CacheRequest	Represents channels for storing resources in the ResponseCache.
CacheResponse	Represent channels for retrieving resources from the ResponseCache.
ContentHandler	The abstract class `ContentHandler` is the superclass of all classes that read an `Object` from a `URLConnection`.
CookieHandler	A CookieHandler object provides a callback mechanism to hook up a HTTP state management policy implementation into the HTTP protocol handler.
CookieManager	CookieManager provides a concrete implementation of `CookieHandler`, which separates the storage of cookies from the policy surrounding accepting and rejecting cookies.
DatagramPacket	This class represents a datagram packet.
DatagramSocket	This class represents a socket for sending and receiving datagram packets.
DatagramSocketImpl	Abstract datagram and multicast socket implementation base class.
HttpCookie	An HttpCookie object represents an http cookie, which carries state information between server and user agent.
HttpURLConnection	A URLConnection with support for HTTP-specific features.
IDN	Provides methods to convert internationalized domain names (IDNs) between a normal Unicode representation and an ASCII Compatible Encoding (ACE) representation.
Inet4Address	This class represents an Internet Protocol version 4 (IPv4) address.
Inet6Address	This class represents an Internet Protocol version 6 (IPv6) address.
InetAddress	This class represents an Internet Protocol (IP) address.
InetSocketAddress	This class implements an IP Socket Address (IP address + port number) It can also be a pair (hostname + port number), in which case an attempt will be made to resolve the hostname.
InterfaceAddress	This class represents a Network Interface address.
JarURLConnection	A URL Connection to a Java ARchive (JAR) file or an entry in a JAR file.
MulticastSocket	The multicast datagram socket class is useful for sending and receiving IP multicast packets.
NetPermission	This class is for various network permissions.
NetworkInterface	This class represents a Network Interface made up of a name, and a list of IP addresses assigned to this interface.
PasswordAuthentication	The class PasswordAuthentication is a data holder that is used by Authenticator.
Proxy	This class represents a proxy setting, typically a type (http, socks) and a socket address.
ProxySelector	Selects the proxy server to use, if any, when connecting to the network resource referenced by a URL.
ResponseCache	Represents implementations of URLConnection caches.
SecureCacheResponse	Represents a cache response originally retrieved through secure means, such as TLS.
ServerSocket	This class implements server sockets.
Socket	This class implements client sockets (also called just "sockets").
SocketAddress	This class represents a Socket Address with no protocol attachment.
SocketImpl	The abstract class `SocketImpl` is a common superclass of all classes that actually implement sockets.
SocketPermission	This class represents access to a network via sockets.
StandardSocketOptions	Defines the *standard* socket options.
URI	Represents a Uniform Resource Identifier (URI) reference.
URL	Class URL represents a Uniform Resource Locator, a pointer to a "resource" on the World Wide Web.
URLClassLoader	This class loader is used to load classes and resources from a search path of URLs referring to both JAR files and directories.
URLConnection	The abstract class `URLConnection` is the superclass of all classes that represent a communications link between the application and a URL.
URLDecoder	Utility class for HTML form decoding.
URLEncoder	Utility class for HTML form encoding.
URLStreamHandler	The abstract class `URLStreamHandler` is the common superclass for all stream protocol handlers.

Enum Summary

Enum	Description
Authenticator.RequestorType	The type of the entity requesting authentication.
Proxy.Type	Represents the proxy type.
StandardProtocolFamily	Defines the standard families of communication protocols.

Exception Summary

Exception	Description
BindException	Signals that an error occurred while attempting to bind a socket to a local address and port.
ConnectException	Signals that an error occurred while attempting to connect a socket to a remote address and port.
HttpRetryException	Thrown to indicate that a HTTP request needs to be retried but cannot be retried automatically, due to streaming mode being enabled.
MalformedURLException	Thrown to indicate that a malformed URL has occurred.
NoRouteToHostException	Signals that an error occurred while attempting to connect a socket to a remote address and port.
PortUnreachableException	Signals that an ICMP Port Unreachable message has been received on a connected datagram.
ProtocolException	Thrown to indicate that there is an error in the underlying protocol, such as a TCP error.
SocketException	Thrown to indicate that there is an error creating or accessing a Socket.
SocketTimeoutException	Signals that a timeout has occurred on a socket read or accept.
UnknownHostException	Thrown to indicate that the IP address of a host could not be determined.
UnknownServiceException	Thrown to indicate that an unknown service exception has occurred.
URISyntaxException	Checked exception thrown to indicate that a string could not be parsed as a URI reference.

Package java.net Description

Provides the classes for implementing networking applications.

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.

Sockets

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.

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.

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 ftp.
- `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

Since:
JDK1.0