Module java.base
Package java.net

Class InetAddress

java.lang.Object
    java.net.InetAddress

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

Inet4Address, Inet6Address

```
public class InetAddress
extends Object
implements Serializable
```

This class represents an Internet Protocol (IP) address.

An IP address is either a 32-bit or 128-bit unsigned number used by IP, a lower-level protocol on which protocols like UDP and TCP are built. The IP address architecture is defined by *RFC 790: Assigned Numbers*, *RFC 1918: Address Allocation for Private Internets*, *RFC 2365: Administratively Scoped IP Multicast*, and *RFC 2373: IP Version 6 Addressing Architecture*. An instance of an InetAddress consists of an IP address and possibly its corresponding host name (depending on whether it is constructed with a host name or whether it has already done reverse host name resolution).

Address types

Address Type	Description
unicast	An identifier for a single interface. A packet sent to a unicast address is delivered to the interface identified by that address. The Unspecified Address -- Also called anylocal or wildcard address. It must never be assigned to any node. It indicates the absence of an address. One example of its use is as the target of bind, which allows a server to accept a client connection on any interface, in case the server host has multiple interfaces. The *unspecified* address must not be used as the destination address of an IP packet. The *Loopback* Addresses -- This is the address assigned to the loopback interface. Anything sent to this IP address loops around and becomes IP input on the local host. This address is often used when testing a client.
multicast	An identifier for a set of interfaces (typically belonging to different nodes). A packet sent to a multicast address is delivered to all interfaces identified by that address.

IP address scope

Link-local addresses are designed to be used for addressing on a single link for purposes such as auto-address configuration, neighbor discovery, or when no routers are present.

Site-local addresses are designed to be used for addressing inside of a site without the need for a global prefix.

Global addresses are unique across the internet.

Textual representation of IP addresses

The textual representation of an IP address is address family specific.

For IPv4 address format, please refer to Inet4Address#format; For IPv6 address format, please refer to Inet6Address#format.

There is a couple of System Properties affecting how IPv4 and IPv6 addresses are used.

Host Name Resolution

The InetAddress class provides methods to resolve host names to their IP addresses and vice versa. The actual resolution is delegated to an InetAddress resolver.

Host name-to-IP address resolution maps a host name to an IP address. For any host name, its corresponding IP address is returned.

Reverse name resolution means that for any IP address, the host associated with the IP address is returned.

The built-in InetAddress resolver implementation does host name-to-IP address resolution and vice versa through the use of a combination of local machine configuration information and network naming services such as the Domain Name System (DNS) and the Lightweight Directory Access Protocol (LDAP). The particular naming services that the built-in resolver uses by default depends on the configuration of the local machine.

`InetAddress` has a service provider mechanism for InetAddress resolvers that allows a custom InetAddress resolver to be used instead of the built-in implementation. `InetAddressResolverProvider` is the service provider class. Its API docs provide all the details on this mechanism.

InetAddress Caching

The InetAddress class has a cache to store successful as well as unsuccessful host name resolutions.

By default, when a security manager is installed, in order to protect against DNS spoofing attacks, the result of positive host name resolutions are cached forever. When a security manager is not installed, the default behavior is to cache entries for a finite (implementation dependent) period of time. The result of unsuccessful host name resolution is cached for a very short period of time (10 seconds) to improve performance.

If the default behavior is not desired, then a Java security property can be set to a different Time-to-live (TTL) value for positive caching. Likewise, a system admin can configure a different negative caching TTL value when needed.

Two Java security properties control the TTL values used for positive and negative host name resolution caching:

networkaddress.cache.ttl
Indicates the caching policy for successful name lookups from the name service. The value is specified as an integer to indicate the number of seconds to cache the successful lookup. The default setting is to cache for an implementation specific period of time.

A value of -1 indicates "cache forever".

networkaddress.cache.negative.ttl (default: 10)
Indicates the caching policy for un-successful name lookups from the name service. The value is specified as an integer to indicate the number of seconds to cache the failure for un-successful lookups.

A value of 0 indicates "never cache". A value of -1 indicates "cache forever".

Since:
1.0

See Also:

`getByAddress(byte[])`,
`getByAddress(java.lang.String, byte[])`,
`getAllByName(java.lang.String)`,
`getByName(java.lang.String)`,
`getLocalHost()`,
Serialized Form

Method Summary

All Methods | Static Methods | Instance Methods | Concrete Methods

Modifier and Type	Method	Description
boolean	**equals**(**Object** obj)	Compares this object against the specified object.
byte[]	**getAddress**()	Returns the raw IP address of this InetAddress object.
static **InetAddress**[]	**getAllByName**(**String** host)	Given the name of a host, returns an array of its IP addresses, based on the configured system resolver.
static **InetAddress**	**getByAddress**(byte[] addr)	Returns an InetAddress object given the raw IP address .
static **InetAddress**	**getByAddress**(**String** host, byte[] addr)	Creates an InetAddress based on the provided host name and IP address.
static **InetAddress**	**getByName**(**String** host)	Determines the IP address of a host, given the host's name.
String	**getCanonicalHostName**()	Gets the fully qualified domain name for this IP address.
String	**getHostAddress**()	Returns the IP address string in textual presentation.
String	**getHostName**()	Gets the host name for this IP address.
static **InetAddress**	**getLocalHost**()	Returns the address of the local host.
static **InetAddress**	**getLoopbackAddress**()	Returns the loopback address.
int	**hashCode**()	Returns a hashcode for this IP address.
boolean	**isAnyLocalAddress**()	Utility routine to check if the InetAddress is a wildcard address.
boolean	**isLinkLocalAddress**()	Utility routine to check if the InetAddress is a link local address.

boolean	**isLoopbackAddress**()	Utility routine to check if the InetAddress is a loopback address.
boolean	**isMCGlobal**()	Utility routine to check if the multicast address has global scope.
boolean	**isMCLinkLocal**()	Utility routine to check if the multicast address has link scope.
boolean	**isMCNodeLocal**()	Utility routine to check if the multicast address has node scope.
boolean	**isMCOrgLocal**()	Utility routine to check if the multicast address has organization scope.
boolean	**isMCSiteLocal**()	Utility routine to check if the multicast address has site scope.
boolean	**isMulticastAddress**()	Utility routine to check if the InetAddress is an IP multicast address.
boolean	**isReachable**(int timeout)	Test whether that address is reachable.
boolean	**isReachable**(**NetworkInterface** netif, int ttl, int timeout)	Test whether that address is reachable.
boolean	**isSiteLocalAddress**()	Utility routine to check if the InetAddress is a site local address.
String	**toString**()	Converts this IP address to a `String`.

Methods declared in class java.lang.Object

clone, finalize, getClass, notify, notifyAll, wait, wait, wait

Method Details

isMulticastAddress

```
public boolean isMulticastAddress()
```

Utility routine to check if the InetAddress is an IP multicast address.

Returns:

a `boolean` indicating if the InetAddress is an IP multicast address

Since:

1.1

isAnyLocalAddress

```
public boolean isAnyLocalAddress()
```

Utility routine to check if the InetAddress is a wildcard address.

Returns:

a `boolean` indicating if the InetAddress is a wildcard address.

Since:

1.4

isLoopbackAddress

```
public boolean isLoopbackAddress()
```

Utility routine to check if the InetAddress is a loopback address.

Returns:

a `boolean` indicating if the InetAddress is a loopback address; or false otherwise.

Since:

1.4

isLinkLocalAddress

```
public boolean isLinkLocalAddress()
```

Utility routine to check if the InetAddress is a link local address.

Returns:

a `boolean` indicating if the InetAddress is a link local address; or false if address is not a link local unicast address.

Since:

1.4

isSiteLocalAddress

```
public boolean isSiteLocalAddress()
```

Utility routine to check if the InetAddress is a site local address.

Returns:

a `boolean` indicating if the InetAddress is a site local address; or false if address is not a site local unicast address.

Since:

1.4

isMCGlobal

```
public boolean isMCGlobal()
```

Utility routine to check if the multicast address has global scope.

Returns:

a `boolean` indicating if the address has is a multicast address of global scope, false if it is not of global scope or it is not a multicast address

Since:

1.4

isMCNodeLocal

```
public boolean isMCNodeLocal()
```

Utility routine to check if the multicast address has node scope.

Returns:

a `boolean` indicating if the address has is a multicast address of node-local scope, false if it is not of node-local scope or it is not a multicast address

Since:

1.4

isMCLinkLocal

```
public boolean isMCLinkLocal()
```

Utility routine to check if the multicast address has link scope.

Returns:

a `boolean` indicating if the address has is a multicast address of link-local scope, false if it is not of link-local scope or it is not a multicast address

Since:

1.4

isMCSiteLocal

```
public boolean isMCSiteLocal()
```

Utility routine to check if the multicast address has site scope.

Returns:

a `boolean` indicating if the address has is a multicast address of site-local scope, false if it is not of site-local scope or it is not a multicast address

Since:

1.4

isMCOrgLocal

```
public boolean isMCOrgLocal()
```

Utility routine to check if the multicast address has organization scope.

Returns:

a `boolean` indicating if the address has is a multicast address of organization-local scope, false if it is not of organization-local scope or it is not a multicast address

Since:

1.4

isReachable

```
public boolean isReachable(int timeout)
                   throws IOException
```

Test whether that address is reachable. Best effort is made by the implementation to try to reach the host, but firewalls and server configuration may block requests resulting in an unreachable status while some specific ports may be accessible. A typical implementation will use ICMP ECHO REQUESTs if the privilege can be obtained, otherwise it will try to establish a TCP connection on port 7 (Echo) of the destination host.

The timeout value, in milliseconds, indicates the maximum amount of time the try should take. If the operation times out before getting an answer, the host is deemed unreachable. A negative value will result in an IllegalArgumentException being thrown.

Parameters:

`timeout` - the time, in milliseconds, before the call aborts

Returns:

a `boolean` indicating if the address is reachable.

Throws:

`IOException` - if a network error occurs

`IllegalArgumentException` - if `timeout` is negative.

Since:

1.5

isReachable

```
public boolean isReachable(NetworkInterface netif,
                           int ttl,
                           int timeout)
                    throws IOException
```

Test whether that address is reachable. Best effort is made by the implementation to try to reach the host, but firewalls and server configuration may block requests resulting in a unreachable status while some specific ports may be accessible. A typical implementation will use ICMP ECHO REQUESTs if the privilege can be obtained, otherwise it will try to establish a TCP connection on port 7 (Echo) of the destination host.

The `network interface` and `ttl` parameters let the caller specify which network interface the test will go through and the maximum number of hops the packets should go through. A negative value for the `ttl` will result in an IllegalArgumentException being thrown.

The timeout value, in milliseconds, indicates the maximum amount of time the try should take. If the operation times out before getting an answer, the host is deemed unreachable. A negative value will result in an IllegalArgumentException being thrown.

Parameters:

`netif` - the NetworkInterface through which the test will be done, or null for any interface

`ttl` - the maximum numbers of hops to try or 0 for the default

`timeout` - the time, in milliseconds, before the call aborts

Returns:

a `boolean` indicating if the address is reachable.

Throws:

`IllegalArgumentException` - if either `timeout` or `ttl` are negative.

`IOException` - if a network error occurs

Since:

1.5

getHostName

```
public String getHostName()
```

Gets the host name for this IP address.

If this InetAddress was created with a host name, this host name will be remembered and returned; otherwise, a reverse name lookup will be performed and the result will be returned based on the system configured resolver. If a lookup of the name service is required, call `getCanonicalHostName`.

If there is a security manager, its `checkConnect` method is first called with the hostname and `-1` as its arguments to see if the operation is allowed. If the operation is not allowed, it will return the textual representation of the IP address.

Returns:

the host name for this IP address, or if the operation is not allowed by the security check, the textual representation of the IP address.

See Also:

`getCanonicalHostName()`,
`SecurityManager.checkConnect(java.lang.String, int)`

getCanonicalHostName

```
public String getCanonicalHostName()
```

Gets the fully qualified domain name for this IP address. Best effort method, meaning we may not be able to return the FQDN depending on the underlying system configuration.

If there is a security manager, this method first calls its `checkConnect` method with the hostname and `-1` as its arguments to see if the calling code is allowed to know the hostname for this IP address, i.e., to connect to the host. If the operation is not allowed, it will return the textual representation of the IP address.

Returns:

the fully qualified domain name for this IP address, or if the operation is not allowed by the security check, the textual representation of the IP address.

Since:

1.4

See Also:

`SecurityManager.checkConnect(java.lang.String, int)`

getAddress

```
public byte[] getAddress()
```

Returns the raw IP address of this `InetAddress` object. The result is in network byte order: the highest order byte of the address is in `getAddress()[0]`.

Returns:

the raw IP address of this object.

getHostAddress

```
public String getHostAddress()
```

Returns the IP address string in textual presentation.

Returns:

the raw IP address in a string format.

Since:

1.0.2

hashCode

```
public int hashCode()
```

Returns a hashcode for this IP address.

Overrides:

`hashCode` in class `Object`

Returns:

a hash code value for this IP address.

See Also:

`Object.equals(java.lang.Object)`,
`System.identityHashCode(java.lang.Object)`

equals

```
public boolean equals(Object obj)
```

Compares this object against the specified object. The result is `true` if and only if the argument is not `null` and it represents the same IP address as this object.

Two instances of `InetAddress` represent the same IP address if the length of the byte arrays returned by `getAddress` is the same for both, and each of the array components is the same for the byte arrays.

Overrides:

`equals` in class `Object`

Parameters:

`obj` - the object to compare against.

Returns:

`true` if the objects are the same; `false` otherwise.

See Also:

`getAddress()`

toString

```
public String toString()
```

Converts this IP address to a `String`. The string returned is of the form: hostname / literal IP address. If the host name is unresolved, no reverse lookup is performed. The hostname part will be represented by an empty string.

Overrides:

`toString` in class `Object`

Returns:

a string representation of this IP address.

getByAddress

```
public static InetAddress getByAddress(String host,
                                       byte[] addr)
                                throws UnknownHostException
```

Creates an InetAddress based on the provided host name and IP address. The system-wide resolver is not used to check the validity of the address.

The host name can either be a machine name, such as "`www.example.com`", or a textual representation of its IP address.

No validity checking is done on the host name either.

If addr specifies an IPv4 address an instance of Inet4Address will be returned; otherwise, an instance of Inet6Address will be returned.

IPv4 address byte array must be 4 bytes long and IPv6 byte array must be 16 bytes long

Parameters:

`host` - the specified host

`addr` - the raw IP address in network byte order

Returns:

an InetAddress object created from the raw IP address.

Throws:

`UnknownHostException` - if IP address is of illegal length

Since:

1.4

getByName

```
public static InetAddress getByName(String host)
                             throws UnknownHostException
```

Determines the IP address of a host, given the host's name.

The host name can either be a machine name, such as "`www.example.com`", or a textual representation of its IP address. If a literal IP address is supplied, only the validity of the address format is checked.

For `host` specified in literal IPv6 address, either the form defined in RFC 2732 or the literal IPv6 address format defined in RFC 2373 is accepted. IPv6 scoped addresses are also supported. See here for a description of IPv6 scoped addresses.

If the host is `null` or `host.length()` is equal to zero, then an `InetAddress` representing an address of the loopback interface is returned. See RFC 3330 section 2 and RFC 2373 section 2.5.3.

If there is a security manager, and `host` is not `null` or `host.length()` is not equal to zero, the security manager's `checkConnect` method is called with the hostname and `-1` as its arguments to determine if the operation is allowed.

Parameters:

`host` - the specified host, or `null`.

Returns:

an IP address for the given host name.

Throws:

`UnknownHostException` - if no IP address for the `host` could be found, or if a scope_id was specified for a global IPv6 address.

`SecurityException` - if a security manager exists and its checkConnect method doesn't allow the operation

getAllByName

```
public static InetAddress[] getAllByName(String host)
                                  throws UnknownHostException
```

Given the name of a host, returns an array of its IP addresses, based on the configured system resolver.

The host name can either be a machine name, such as "`www.example.com`", or a textual representation of its IP address. If a literal IP address is supplied, only the validity of the address format is checked.

For `host` specified in *literal IPv6 address*, either the form defined in RFC 2732 or the literal IPv6 address format defined in RFC 2373 is accepted. A literal IPv6 address may also be qualified by appending a scoped zone identifier or scope_id. The syntax and usage of scope_ids is described here.

If the host is `null` or `host.length()` is equal to zero, then an `InetAddress` representing an address of the loopback interface is returned. See RFC 3330 section 2 and RFC 2373 section 2.5.3.

If there is a security manager, and `host` is not `null` or `host.length()` is not equal to zero, the security manager's `checkConnect` method is called with the hostname and `-1` as its arguments to determine if the operation is allowed.

Parameters:

`host` - the name of the host, or `null`.

Returns:

an array of all the IP addresses for a given host name.

Throws:

`UnknownHostException` - if no IP address for the `host` could be found, or if a scope_id was specified for a global IPv6 address.

`SecurityException` - if a security manager exists and its `checkConnect` method doesn't allow the operation.

See Also:

`SecurityManager.checkConnect(java.lang.String, int)`

getLoopbackAddress

```
public static InetAddress getLoopbackAddress()
```

Returns the loopback address.

The InetAddress returned will represent the IPv4 loopback address, 127.0.0.1, or the IPv6 loopback address, ::1. The IPv4 loopback address returned is only one of many in the form 127.*.*.*

Returns:

the InetAddress loopback instance.

Since:

1.7

getByAddress

```
public static InetAddress getByAddress(byte[] addr)
                                throws UnknownHostException
```

Returns an `InetAddress` object given the raw IP address . The argument is in network byte order: the highest order byte of the address is in `getAddress()[0]`.

This method doesn't block, i.e. no reverse lookup is performed.

IPv4 address byte array must be 4 bytes long and IPv6 byte array must be 16 bytes long

Parameters:

`addr` - the raw IP address in network byte order

Returns:

an InetAddress object created from the raw IP address.

Throws:

`UnknownHostException` - if IP address is of illegal length

Since:

1.4

getLocalHost

```
public static InetAddress getLocalHost()
                                throws UnknownHostException
```

Returns the address of the local host. This is achieved by retrieving the name of the host from the system, then resolving that name into an `InetAddress`.

Note: The resolved address may be cached for a short period of time.

If there is a security manager, its `checkConnect` method is called with the local host name and `-1` as its arguments to see if the operation is allowed. If the operation is not allowed, an InetAddress representing the loopback address is returned.

Returns:

the address of the local host.

Throws:

`UnknownHostException` - if the local host name could not be resolved into an address.

See Also:

`SecurityManager.checkConnect(java.lang.String, int)`,
`getByName(java.lang.String)`