# JEP 418: Internet-Address Resolution SPI

| | |
|---|---|
| *Owner* | Aleksej Efimov |
| *Type* | Feature |
| *Scope* | SE |
| *Status* | Closed / Delivered |
| *Release* | 18 |
| *Component* | core-libs / java.net |
| *Discussion* | net dash dev at openjdk dot java dot net |
| *Duration* | S |
| *Reviewed by* | Alan Bateman, Brian Goetz, Chris Hegarty, Daniel Fuchs |
| *Endorsed by* | Brian Goetz |
| *Created* | 2021/03/16 16:43 |
| *Updated* | 2022/09/09 10:29 |
| *Issue* | 8263693 |

## Summary

Define a service-provider interface (SPI) for host name and address resolution, so that `java.net.InetAddress` can make use of resolvers other than the platform's built-in resolver.

## Non-Goals

- It is not a goal to develop an alternative resolver to include in the JDK. The JDK's built-in resolver will continue to be used by default.
- It is not a goal for the SPI to support resolution operations beyond those required by the `InetAddress` API.
- It is not a goal to define non-blocking or asynchronous resolution APIs.

## Motivation

The `java.net.InetAddress` API resolves host names to Internet Protocol (IP) addresses, and vice versa. The API currently uses the operating system's native resolver, which is typically configured to use a combination of a local `hosts` file and the Domain Name System (DNS).

Motivations for defining a service-provider interface for name and address resolution include:

- *Project Loom* — A resolution operation with the `InetAddress` API currently blocks in an operating-system call. This is a problem for Loom's user-mode virtual threads, since it prevents underlying platform threads from servicing other virtual threads while waiting for a resolution operation to complete. An alternative resolver could implement the DNS client protocol directly, without blocking.
- *Emerging network protocols* — A resolver SPI would enable the seamless integration of new resolution protocols such as DNS over QUIC, TLS, or HTTPS.
- *Customization* — A resolver SPI would enable frameworks and applications to have finer control over resolution results, and would allow existing libraries to be retrofitted with a custom resolver.
- *Testing* — Prototyping and testing activities often require control of host name and address resolution results, for example when mocking components that use the `InetAddress` API.

## Description

The `InetAddress` API defines multiple methods for lookup operations:

- `InetAddress::getAllByName` performs a *forward* lookup, mapping a host name to a set of IP addresses.
- `InetAddress::getByName` also performs a forward lookup, mapping a host name to the first address in its set of addresses.
- `InetAddress::getCanonicalHostName` performs a *reverse* lookup, mapping an IP address to a fully qualified domain name. For example:

```
var addressBytes = new byte[] { (byte) 192, 0, 43, 7};
var resolvedHostName = InetAddress.getByAddress(addressBytes)
                                  .getCanonicalHostName();
```

- `InetAddress::getHostName` also performs a reverse lookup, if needed.

By default, `InetAddress` uses the operating system's native resolver to perform lookups. The result of that lookup, whether positive or negative, may be cached in order to avoid further lookups of the same host.

### *Service-provider interface*

The `InetAddress` API will use a service loader to locate a resolver provider. If no provider is found, the built-in implementation will be used as before.

The new classes in the `java.net.spi` package are:

- `InetAddressResolverProvider` — an abstract class defining the service to be located by `java.util.ServiceLoader`. An `InetAddressResolverProvider` is, essentially, a factory for resolvers. The

instantiated resolver will be set as the system-wide resolver, and `InetAddress` will delegate all lookup requests to that resolver.

- `InetAddressResolver` — an interface that defines methods for the fundamental forward and reverse lookup operations. An instance of this interface is obtained from an instance of `InetAddressResolverProvider`.
- `InetAddressResolver.LookupPolicy` — a class whose instances describe the characteristics of a resolution request, including the requested address type and the order in which addresses should be returned.
- `InetAddressResolverProvider.Configuration` — an interface describing the platform's built-in configuration for resolution operations. It provides access to the local host name and the built-in resolver. It is used by custom resolver providers to bootstrap resolver construction or to implement partial delegation of resolution requests to the operating system's native resolver.

## Alternatives

Without an SPI such as the one proposed here, applications will have to continue to use the workarounds available today.

- An application can use the Java Naming and Directory Interface (JNDI) and its DNS provider to look up network names and addresses. This approach can be useful for applications that require fine control of DNS lookups, but it is decoupled from `InetAddress` and thus using it with the platform's networking API requires additional effort.
- An application can use the operating system's resolver libraries directly via the Java Native Interface (JNI) or the foreign function API from Project Panama. As with JNDI, however, this approach is decoupled from `InetAddress` and thus more awkward to use.
- An application can use the non-standard, JDK-specific system property `jdk.net.hosts.file` to configure `InetAddress` to use a specific file, rather than the operating system's native resolver, to map host names to IP addresses. This feature is useful for testing but it is not a general purpose solution since the complete list of host names is not always known in advance.

## Testing

We will develop new tests for the resolver SPI.

We will develop proof-of-concept resolver providers to demonstrate and verify that the SPI can be used to develop and deploy alternative implementations that are used in preference to the JDK's built-in implementation. We will make at least one of these providers available to seed development of more complete implementations.