java.util

Class ServiceLoader<S>

java.lang.Object
　　java.util.ServiceLoader<S>

Type Parameters:

`S` - The type of the service to be loaded by this loader

All Implemented Interfaces:

Iterable<S>

```
public final class ServiceLoader<S>
extends Object
implements Iterable<S>
```

A simple service-provider loading facility. "服务-提供者"加载设施

服务是接口和抽象类的集合，服务提供者是服务的特定实现。

A *service* is a well-known set of interfaces and (usually abstract) classes. A *service provider* is a specific implementation of a service. The classes in a provider typically implement the interfaces and subclass the classes defined in the service itself. Service providers can be installed in an implementation of the Java platform in the form of extensions, that is, jar files placed into any of the usual extension directories. Providers can also be made available by adding them to the application's class path or by some other platform-specific means.

For the purpose of loading, a service is represented by a single type, that is, a single interface or abstract class. (A concrete class can be used, but this is not recommended.) A provider of a given service contains one or more concrete classes that extend this *service type* with data and code specific to the provider. The *provider class* is typically not the entire provider itself but rather a proxy which contains enough information to decide whether the provider is able to satisfy a particular request together with code that can create the actual provider on demand. The details of provider classes tend to be highly service-specific; no single class or interface could possibly unify them, so no such type is defined here. The only requirement enforced by this facility is that provider classes must have a zero-argument constructor so that they can be instantiated during loading. 该设施唯一强制要求的是，提供者类必须具有一个无参数的构造函数，以便在加载期间可以被实例化。

服务提供者是由在资源目录META-INF/services中放置一个提供者配置文件的标识。

A service provider is identified by placing a *provider-configuration file* in the resource directory `META-INF/services`. The file's name is the fully-qualified binary name of the service's type. The file contains a list of fully-qualified binary names of concrete provider classes, one per line. Space and tab characters surrounding each name, as well as blank lines, are ignored. The comment character is `'#'` (`'\u0023'`, NUMBER SIGN); on each line all characters following the first comment character are ignored. The file must be encoded in UTF-8. 该文件必须被编码成UTF-8。

该文件包含一个具体的提供者类的完全限定二进制名称的列表，每行一个。

If a particular concrete provider class is named in more than one configuration file, or is named in the same configuration file more than once, then the duplicates are ignored. The configuration file naming a particular provider need not be in the same jar file or other distribution unit as the provider itself. The provider must be accessible from the same class loader that was initially queried to locate the configuration file; note that this is not necessarily the class loader from which the file was actually loaded.

提供者在需要时被延迟地定位和实例化。服务加载器维护一个到目前为止已加载的提供者的缓存。

Providers are located and instantiated lazily, that is, on demand. A service loader maintains a cache of the providers that have been loaded so far. Each invocation of the `iterator` method returns an iterator that first yields all of the elements of the cache, in instantiation order, and then lazily locates and instantiates any remaining providers, adding each one to the cache in turn. The cache can be cleared via the `reload` method. 迭代器方法的每次调用返回一个迭代器，它首先会以实例化顺序生成缓存中的所有元素，然后延迟地定位并实例化剩余的提供者，依次添加每个到缓存中。

服务加载器始终在调用者的安全上下文中执行。

Service loaders always execute in the security context of the caller. Trusted system code should typically invoke the methods in this class, and the methods of the iterators which they return, from within a privileged security context.

Instances of this class are not safe for use by multiple concurrent threads. 本类的实例在多个并发线程中使用是不安全的。

Unless otherwise specified, passing a `null` argument to any method in this class will cause a `NullPointerException` to be thrown.

示例 **Example** Suppose we have a service type `com.example.CodecSet` which is intended to represent sets of encoder/decoder pairs for some protocol. In this case it is an abstract class with two abstract methods:

一些协议的编码器/解码器对

```
public abstract Encoder getEncoder(String encodingName);
public abstract Decoder getDecoder(String encodingName);
```

Each method returns an appropriate object or `null` if the provider does not support the given encoding. Typical providers support more than one encoding.

If `com.example.impl.StandardCodecs` is an implementation of the `CodecSet` service then its jar file also contains a file named 1. 包含该命名的文件

```
        META-INF/services/com.example.CodecSet
```

This file contains the single line: 2. 此文件包含一行

```
        com.example.impl.StandardCodecs    # Standard codecs
```

The `CodecSet` class creates and saves a single service instance at initialization: 3. 在初始化时创建并保存服务实例

```
        private static ServiceLoader<CodecSet> codecSetLoader
            = ServiceLoader.load(CodecSet.class);
```

4. 静态工厂方法：循环访问已知可用的提供者列表

To locate an encoder for a given encoding name it defines a static factory method which iterates through the known and available providers, returning only when it has located a suitable encoder or has run out of providers.

```
        public static Encoder getEncoder(String encodingName) {
            for (CodecSet cp : codecSetLoader) {
                Encoder enc = cp.getEncoder(encodingName);
                if (enc != null)
                    return enc;
            }
            return null;
        }
```

A `getDecoder` method is defined similarly.

类加载器的类路径包含的远程网络 URLs 会在提供者配置文件搜索过程处理中被取消

使用说明 **Usage Note** If the class path of a class loader that is used for provider loading includes remote network URLs then those URLs will be dereferenced in the process of searching for provider-configuration files.

This activity is normal, although it may cause puzzling entries to be created in web-server logs. If a web server is not configured correctly, however, then this activity may cause the provider-loading algorithm to fail spuriously.

A web server should return an HTTP 404 (Not Found) response when a requested resource does not exist. Sometimes, however, web servers are erroneously configured to return an HTTP 200 (OK) response along with a helpful HTML error page in such cases. This will cause a `ServiceConfigurationError` to be thrown when this class attempts to parse the HTML page as a provider-configuration file. The best solution to this problem is to fix the misconfigured web server to return the correct response code (HTTP 404) along with the HTML error page.

当本类尝试将 HTML 页面解析为提供者配置文件时，会抛出一个服务配置错误异常。

Since:

1.6

Method Summary

Methods

Modifier and Type	Method and Description
`Iterator<S>`	`iterator()` 延迟地加载该服务的可用提供者 Lazily loads the available providers of this loader's service.
`static <S> ServiceLoader<S>`	`load(Class<S> service)` 创建一个给定的服务类型的服务加载器(使用当前线程上下文类加载) Creates a new service loader for the given service type, using the current thread's context class loader.
`static <S> ServiceLoader<S>`	`load(Class<S> service, ClassLoader loader)` Creates a new service loader for the given service type and class loader. 创建一个给定的服务类型和类加载器的服务加载器
`static <S> ServiceLoader<S>`	`loadInstalled(Class<S> service)` Creates a new service loader for the given service type, using the extension class loader.

void	**reload**() 清除该加载器的提供者缓存，以便所有提供者将被重新加载。 Clear this loader's provider cache so that all providers will be reloaded.
String	**toString**() Returns a string describing this service.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Method Detail

reload

```
public void reload()
```

Clear this loader's provider cache so that all providers will be reloaded.

After invoking this method, subsequent invocations of the iterator method will lazily look up and instantiate providers from scratch, just as is done by a newly-created loader.

This method is intended for use in situations in which new providers can be installed into a running Java virtual machine.

iterator

```
public Iterator<S> iterator()
```

Lazily loads the available providers of this loader's service.

The iterator returned by this method first yields all of the elements of the provider cache, in instantiation order. It then lazily loads and instantiates any remaining providers, adding each one to the cache in turn.

To achieve laziness the actual work of parsing the available provider-configuration files and instantiating providers must be done by the iterator itself. Its hasNext and next methods can therefore throw a ServiceConfigurationError if a provider-configuration file violates the specified format, or if it names a provider class that cannot be found and instantiated, or if the result of instantiating the class is not assignable to the service type, or if any other kind of exception or error is thrown as the next provider is located and instantiated. To write robust code it is only necessary to catch ServiceConfigurationError when using a service iterator.

If such an error is thrown then subsequent invocations of the iterator will make a best effort to locate and instantiate the next available provider, but in general such recovery cannot be guaranteed.

> **Design Note** Throwing an error in these cases may seem extreme. The rationale for this behavior is that a malformed provider-configuration file, like a malformed class file, indicates a serious problem with the way the Java virtual machine is configured or is being used. As such it is preferable to throw an error rather than try to recover or, even worse, fail silently.

The iterator returned by this method does not support removal. Invoking its remove method will cause an UnsupportedOperationException to be thrown.

Specified by:

iterator in interface Iterable<S>

Returns:

An iterator that lazily loads providers for this loader's service

load

```
public static <S> ServiceLoader<S> load(Class<S> service,
                                        ClassLoader loader)
```

Creates a new service loader for the given service type and class loader.

Parameters:

`service` - The interface or abstract class representing the service

`loader` - The class loader to be used to load provider-configuration files and provider classes, or `null` if the system class loader (or, failing that, the bootstrap class loader) is to be used

Returns:

A new service loader

load

```
public static <S> ServiceLoader<S> load(Class<S> service)
```

Creates a new service loader for the given service type, using the current thread's context class loader.

An invocation of this convenience method of the form

```
    ServiceLoader.load(service)
```

is equivalent to

```
    ServiceLoader.load(service,
                       Thread.currentThread().getContextClassLoader())
```

Parameters:

`service` - The interface or abstract class representing the service

Returns:

A new service loader

loadInstalled

```
public static <S> ServiceLoader<S> loadInstalled(Class<S> service)
```

Creates a new service loader for the given service type, using the extension class loader.

This convenience method simply locates the extension class loader, call it *extClassLoader*, and then returns

```
    ServiceLoader.load(service, extClassLoader)
```

If the extension class loader cannot be found then the system class loader is used; if there is no system class loader then the bootstrap class loader is used.

This method is intended for use when only installed providers are desired. The resulting service will only find and load providers that have been installed into the current Java virtual machine; providers on the application's class path will be ignored.

Parameters:

`service` - The interface or abstract class representing the service

Returns:

A new service loader

toString

```
public String toString()
```

Returns a string describing this service.

Overrides:

`toString` in class `Object`

Returns:

A descriptive string