RFC 230 Promise Update

Draft

30 Pages

Abstract

Updates to the Promise specification.

0 Document Information

0.1 License

DISTRIBUTION AND FEEDBACK LICENSE, Version 2.0

The OSGi Alliance hereby grants you a limited copyright license to copy and display this document (the “Distribution”) in any medium without fee or royalty. This Distribution license is exclusively for the purpose of reviewing and providing feedback to the OSGi Alliance. You agree not to modify the Distribution in any way and further agree to not participate in any way in the making of derivative works thereof, other than as a necessary result of reviewing and providing feedback to the Distribution. You also agree to cause this notice, along with the accompanying consent, to be included on all copies (or portions thereof) of the Distribution. The OSGi Alliance also grants you a perpetual, non-exclusive, worldwide, fully paid-up, royalty free, limited license (without the right to sublicense) under any applicable copyrights, to create and/or distribute an implementation of the Distribution that: (i) fully implements the Distribution including all its required interfaces and functionality; (ii) does not modify, subset, superset or otherwise extend the OSGi Name Space, or include any public or protected packages, classes, Java interfaces, fields or methods within the OSGi Name Space other than those required and authorized by the Distribution. An implementation that does not satisfy limitations (i)-(ii) is not considered an implementation of the Distribution, does not receive the benefits of this license, and must not be described as an implementation of the Distribution. "OSGi Name Space" shall mean the public class or interface declarations whose names begin with "org.osgi" or any recognized successors or replacements thereof. The OSGi Alliance expressly reserves all rights not granted pursuant to these limited copyright licenses including termination of the license at will at any time.

EXCEPT FOR THE LIMITED COPYRIGHT LICENSES GRANTED ABOVE, THE OSGi ALLIANCE DOES NOT GRANT, EITHER EXPRESSLY OR IMPLIEDLY, A LICENSE TO ANY INTELLECTUAL PROPERTY IT, OR ANY THIRD PARTIES, OWN OR CONTROL. Title to the copyright in the Distribution will at all times remain with the OSGi Alliance. The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted therein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

THE DISTRIBUTION IS PROVIDED "AS IS," AND THE OSGi ALLIANCE (INCLUDING ANY THIRD PARTIES THAT HAVE CONTRIBUTED TO THE DISTRIBUTION) MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DISTRIBUTION ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.
NEITHER THE OSGi ALLIANCE NOR ANY THIRD PARTY WILL BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY USE OR DISTRIBUTION OF THE DISTRIBUTION.

Implementation of certain elements of this Distribution may be subject to third party intellectual property rights, including without limitation, patent rights (such a third party may or may not be a member of the OSGi Alliance). The OSGi Alliance is not responsible and shall not be held responsible in any manner for identifying or failing to identify any or all such third party intellectual property rights.

The Distribution is a draft. As a result, the final product may change substantially by the time of final publication, and you are cautioned against relying on the content of this Distribution. You are encouraged to update any implementation of the Distribution if and when such Distribution becomes a final specification.

The OSGi Alliance is willing to receive input, suggestions and other feedback (“Feedback”) on the Distribution. By providing such Feedback to the OSGi Alliance, you grant to the OSGi Alliance and all its Members a non-exclusive, non-transferable,

worldwide, perpetual, irrevocable, royalty-free copyright license to copy, publish, license, modify, sublicense or otherwise distribute and exploit your Feedback for any purpose. Likewise, if incorporation of your Feedback would cause an implementation of the Distribution, including as it may be modified, amended, or published at any point in the future ("Future Specification"), to necessarily infringe a patent or patent application that you own or control, you hereby commit to grant to all implementers of such Distribution or Future Specification an irrevocable, worldwide, sublicenseable, royalty free license under such patent or patent application to make, have made, use, sell, offer for sale, import and export products or services that implement such Distribution or Future Specification. You warrant that (a) to the best of your knowledge you have the right to provide this Feedback, and if you are providing Feedback on behalf of a company, you have the rights to provide Feedback on behalf of your company; (b) the Feedback is not confidential to you and does not violate the copyright or trade secret interests of another; and (c) to the best of your knowledge, use of the Feedback would not cause an implementation of the Distribution or a Future Specification to necessarily infringe any third-party patent or patent application known to you. You also acknowledge that the OSGi Alliance is not required to incorporate your Feedback into any version of the Distribution or a Future Specification.

I HEREBY ACKNOWLEDGE AND AGREE TO THE TERMS AND CONDITIONS DELINEATED ABOVE.

0.2 Trademarks

OSGi™ is a trademark, registered trademark, or service mark of the OSGi Alliance in the US and other countries. Java is a trademark, registered trademark, or service mark of Oracle Corporation in the US and other countries. All other trademarks, registered trademarks, or service marks used in this document are the property of their respective owners and are hereby recognized.

0.3 Feedback

This document can be downloaded from the OSGi Alliance design repository at https://github.com/osgi/design The public can provide feedback about this document by opening a bug at https://www.osgi.org/bugzilla/.

0.4 Table of Contents

0.5 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in 9.1.

`Source code is shown in this typeface.`

0.6 Revision History

The last named individual in this history is currently responsible for this document.

Revision	Date	Comments
Initial	09 Sep 2016	Initial draft including new timout and delay methods on Promise.

1 Introduction

The Promise API is quite a complete and powerful model for asynchronous tasks. However some minor additions will improve its usefulness in other use cases.

2 Application Domain

A Promise object holds the result of a potentially asynchronous task. The receiver of a Promise object can register callbacks on the Promise to be notified when the result is available or can block on the result becoming available. Promises can be chained together in powerful ways to handle asynchronous work flows and recovery. Promises capture the effects of latency and errors by making these explicit in the API signatures.

Latency is represented by callbacks which will eventually be called. Errors are represented by the failure member. In essence, this is what sets Promises apart from things such as RPC calls where such effects are not explicitly captured but rather attempted to be transparently handled.

3 Problem Description

Sometimes you need to end waiting for a value after some period of time. The current API provides no means to fail a promise after some period of time.

Sometimes you need to delay the resolution of a promise to introduce some “slack” in a system. This can be done but the current API provides no easy means of doing so.

4 Requirements

P0010 – Provide a method to fail a promise after some timeout has elapsed.

P0020 – Provide a method to delay the resolution of a promise after some delay has elapsed.

5 Technical Solution

5.1 Timeout

A new timeout method is added to Promise. If a Promise is successfully resolved before the timeout, the returned Promise is resolved with the value of the Promise. If the Promise is resolved with a failure before the timeout, the returned Promise is resolved with the failure of the Promise. If the timeout is reached before the Promise is resolved, the returned Promise is failed with a TimeoutException. The time to wait in milliseconds. Zero and negative time is treated as an immediate timeout.

5.2 Delay

A new delay method is added to Promise. Once a Promise is resolved, resolve the returned Promise with the Promise after the specified delay. The time to delay in milliseconds. Zero and negative time is treated as no delay.

5.3 Checked Exceptions

The org.osgi.util.function package is also updated to improve dealing with checked exception when writing promise callbacks. The Function and Predicate methods are updated to throw Exception. A new Callback functional interface is added which is a like a runnable whose run method throws Exception. This new Callback can be used with a new overload of then.

6 Javadoc

OSGi Javadoc

9/9/16 3:44 PM

Package Summary		Page
org.osgi.util.function	Function Package Version 1.1.	8
org.osgi.util.promise	Promise Package Version 1.1.	12

Package org.osgi.util.function

```
@org.osgi.annotation.versioning.Version(value="1.1")
```

Function Package Version 1.1.

See:
Description

Interface Summary		Page
Callback	A callback that performs an operation and may throw an exception.	9
Function	A function that accepts a single argument and produces a result.	10
Predicate	A predicate that accepts a single argument and produces a boolean result.	11

Package org.osgi.util.function Description

Function Package Version 1.1.

Bundles wishing to use this package must list the package in the Import-Package header of the bundle's manifest.

Example import for consumers using the API in this package:

```
Import-Package: org.osgi.util.function; version="[1.1,2.0)"
```

Example import for providers implementing the API in this package:

```
Import-Package: org.osgi.util.function; version="[1.1,1.2)"
```

Interface Callback

org.osgi.util.function

```
@org.osgi.annotation.versioning.ConsumerType
@FunctionalInterface
public interface Callback
```

A callback that performs an operation and may throw an exception.

This is a functional interface and can be used as the assignment target for a lambda expression or method reference.

Since:
1.1

ThreadSafe

Method Summary		Page
void	run() Execute the callback.	9

Method Detail

run

```
void run()
          throws Exception
```

Execute the callback.

Throws:
`Exception` - An exception thrown by the method.

Interface Function

org.osgi.util.function

Type Parameters:

`T` - The type of the function input.
`R` - The type of the function output.

```
@org.osgi.annotation.versioning.ConsumerType
@FunctionalInterface
public interface Function
```

A function that accepts a single argument and produces a result.

This is a functional interface and can be used as the assignment target for a lambda expression or method reference.

ThreadSafe

Method Summary	Page
R **apply**(T t) Applies this function to the specified argument.	10

Method Detail

apply

```
R apply(T t)
        throws Exception
```

Applies this function to the specified argument.

Parameters:

`t` - The input to this function.

Returns:

The output of this function.

Throws:

`Exception` - An exception thrown by the method.

Interface Predicate

org.osgi.util.function

Type Parameters:

`T` - The type of the predicate input.

```
@org.osgi.annotation.versioning.ConsumerType
@FunctionalInterface
public interface Predicate
```

A predicate that accepts a single argument and produces a boolean result.

This is a functional interface and can be used as the assignment target for a lambda expression or method reference.

ThreadSafe

Method Summary		Page
boolean	**test**(T t) Evaluates this predicate on the specified argument.	11

Method Detail

test

```
boolean test(T t)
      throws Exception
```

Evaluates this predicate on the specified argument.

Parameters:

`t` - The input to this predicate.

Returns:

`true` if the specified argument is accepted by this predicate; `false` otherwise.

Throws:

`Exception` - An exception thrown by the method.

Package org.osgi.util.promise

```
@org.osgi.annotation.versioning.Version(value="1.1")
```

Promise Package Version 1.1.

See:

Description

Interface Summary		Page
Failure	Failure callback for a Promise.	17
Promise	A Promise of a value.	18
Success	Success callback for a Promise.	28

Class Summary		Page
Deferred	A Deferred Promise resolution.	13
Promises	Static helper methods for `Promise`s.	26

Exception Summary		Page
FailedPromisesException	Promise failure exception for a collection of failed Promises.	16
TimeoutException	Timeout exception for a Promise.	29

Package org.osgi.util.promise Description

Promise Package Version 1.1.

Bundles wishing to use this package must list the package in the Import-Package header of the bundle's manifest.

Example import for consumers using the API in this package:

```
Import-Package: org.osgi.util.promise; version="[1.1,2.0)"
```

Example import for providers implementing the API in this package:

```
Import-Package: org.osgi.util.promise; version="[1.1,1.2)"
```

Class Deferred

org.osgi.util.promise

```
java.lang.Object
  └ org.osgi.util.promise.Deferred
```

Type Parameters:

T - The value type associated with the created Promise.

```
public class Deferred
extends Object
```

A Deferred Promise resolution.

Instances of this class can be used to create a Promise that can be resolved in the future. The associated Promise can be successfully resolved with resolve(Object) or resolved with a failure with fail(Throwable). It can also be resolved with the resolution of another promise using resolveWith(Promise).

The associated Promise can be provided to any one, but the Deferred object should be made available only to the party that will responsible for resolving the Promise.

Immutable

Constructor Summary	Page
Deferred() Create a new Deferred with an associated Promise.	13

Method Summary		Page
void	**fail**(Throwable failure) Fail the Promise associated with this Deferred.	14
Promise<T>	**getPromise**() Returns the Promise associated with this Deferred.	14
void	**resolve**(T value) Successfully resolve the Promise associated with this Deferred.	14
Promise<Void>	**resolveWith**(Promise<? extends T> with) Resolve the Promise associated with this Deferred with the specified Promise.	14
String	**toString**() Returns a string representation of the associated Promise.	15

Constructor Detail

Deferred

```
public Deferred()
```

Create a new Deferred with an associated Promise.

Method Detail

getPromise

`public Promise<T> getPromise()`

Returns the Promise associated with this Deferred.

Returns:
The Promise associated with this Deferred.

---

resolve

`public void resolve(T value)`

Successfully resolve the Promise associated with this Deferred.

After the associated Promise is resolved with the specified value, all registered `callbacks` are called and any `chained` Promises are resolved. This may occur asynchronously to this method.

Resolving the associated Promise *happens-before* any registered callback is called. That is, in a registered callback, `Promise.isDone()` must return `true` and `Promise.getValue()` and `Promise.getFailure()` must not block.

Parameters:
`value` - The value of the resolved Promise.
Throws:
`IllegalStateException` - If the associated Promise was already resolved.

---

fail

`public void fail(Throwable failure)`

Fail the Promise associated with this Deferred.

After the associated Promise is resolved with the specified failure, all registered `callbacks` are called and any `chained` Promises are resolved. This may occur asynchronously to this method.

Resolving the associated Promise *happens-before* any registered callback is called. That is, in a registered callback, `Promise.isDone()` must return `true` and `Promise.getValue()` and `Promise.getFailure()` must not block.

Parameters:
`failure` - The failure of the resolved Promise. Must not be `null`.
Throws:
`IllegalStateException` - If the associated Promise was already resolved.

---

resolveWith

`public Promise<Void> resolveWith(Promise<? extends T> with)`

Resolve the Promise associated with this Deferred with the specified Promise.

If the specified Promise is successfully resolved, the associated Promise is resolved with the value of the specified Promise. If the specified Promise is resolved with a failure, the associated Promise is resolved with the failure of the specified Promise.

After the associated Promise is resolved with the specified Promise, all registered `callbacks` are called and any `chained` Promises are resolved. This may occur asynchronously to this method.

Resolving the associated Promise *happens-before* any registered callback is called. That is, in a registered callback, `Promise.isDone()` must return `true` and `Promise.getValue()` and `Promise.getFailure()` must not block.

Parameters:

`with` - A Promise whose value or failure must be used to resolve the associated Promise. Must not be `null`.

Returns:

A Promise that is resolved only when the associated Promise is resolved by the specified Promise. The returned Promise must be successfully resolved with the value `null`, if the associated Promise was resolved by the specified Promise. The returned Promise must be resolved with a failure of `IllegalStateException`, if the associated Promise was already resolved when the specified Promise was resolved.

---

toString

```
public String toString()
```

Returns a string representation of the associated Promise.

Overrides:

`toString` in class `Object`

Returns:

A string representation of the associated Promise.

Since:

1.1

Class FailedPromisesException

org.osgi.util.promise

```
java.lang.Object
  └java.lang.Throwable
      └java.lang.Exception
          └java.lang.RuntimeException
              └org.osgi.util.promise.FailedPromisesException
```

All Implemented Interfaces:

Serializable

```
public class FailedPromisesException
extends RuntimeException
```

Promise failure exception for a collection of failed Promises.

Constructor Summary	Page
FailedPromisesException(Collection<Promise<?>> failed, Throwable cause) Create a new FailedPromisesException with the specified Promises.	16

Method Summary		Page
Collection<Promise<?>>	**getFailedPromises**() Returns the collection of Promises that have been resolved with a failure.	16

Constructor Detail

FailedPromisesException

```
public FailedPromisesException(Collection<Promise<?>> failed,
                               Throwable cause)
```

Create a new FailedPromisesException with the specified Promises.

Parameters:

`failed` - A collection of Promises that have been resolved with a failure. Must not be `null`, must not be empty and all of the elements in the collection must not be `null`.
`cause` - The cause of this exception. This is typically the failure of the first Promise in the specified collection.

Method Detail

getFailedPromises

```
public Collection<Promise<?>> getFailedPromises()
```

Returns the collection of Promises that have been resolved with a failure.

Returns:

The collection of Promises that have been resolved with a failure. The returned collection is unmodifiable.

Interface Failure

org.osgi.util.promise

```
@org.osgi.annotation.versioning.ConsumerType
@FunctionalInterface
public interface Failure
```

Failure callback for a Promise.

A Failure callback is registered with a `Promise` using the `Promise.then(Success, Failure)` method and is called if the Promise is resolved with a failure.

This is a functional interface and can be used as the assignment target for a lambda expression or method reference.

ThreadSafe

Method Summary		Page
void	**fail**(Promise<?> resolved) Failure callback for a Promise.	17

Method Detail

fail

```
void fail(Promise<?> resolved)
          throws Exception
```

Failure callback for a Promise.

This method is called if the Promise with which it is registered resolves with a failure.

In the remainder of this description we will refer to the Promise returned by `Promise.then(Success, Failure)` when this Failure callback was registered as the chained Promise.

If this methods completes normally, the chained Promise must be failed with the same exception which failed the resolved Promise. If this method throws an exception, the chained Promise must be failed with the thrown exception.

Parameters:

`resolved` - The failed resolved `Promise`.

Throws:

`Exception` - The chained Promise must be failed with the thrown exception.

Interface Promise

org.osgi.util.promise

Type Parameters:

`T` - The value type associated with this Promise.

```
@org.osgi.annotation.versioning.ProviderType
public interface Promise
```

A Promise of a value.

A Promise represents a future value. It handles the interactions for asynchronous processing. A `Deferred` object can be used to create a Promise and later resolve the Promise. A Promise is used by the caller of an asynchronous function to get the result or handle the error. The caller can either get a callback when the Promise is resolved with a value or an error, or the Promise can be used in chaining. In chaining, callbacks are provided that receive the resolved Promise, and a new Promise is generated that resolves based upon the result of a callback.

Both `callbacks` and `chaining` can be repeated any number of times, even after the Promise has been resolved.

Example callback usage:

```
Promise<String> foo = foo();
foo.onResolve(() -> System.out.println("resolved"));
```

Example chaining usage;

```
Success<String,String> doubler = p -> Promises
  .resolved(p.getValue() + p.getValue());
Promise<String> foo = foo().then(doubler).then(doubler);
```

ThreadSafe

Method Summary		Page
Promise<T>	**delay**(long milliseconds) Delay after the resolution of this Promise.	24
Promise<T>	**fallbackTo**(Promise<? extends T> fallback) Fall back to the value of the specified Promise if this Promise fails.	24
Promise<T>	**filter**(Predicate<? super T> predicate) Filter the value of this Promise.	22
Promise<R>	**flatMap**(Function<? super T,Promise<? extends R>> mapper) FlatMap the value of this Promise.	22
Throwable	**getFailure**() Returns the failure of this Promise.	19
T	**getValue**() Returns the value of this Promise.	19
boolean	**isDone**() Returns whether this Promise has been resolved.	19
Promise<R>	**map**(Function<? super T,? extends R> mapper) Map the value of this Promise.	22
Promise<T>	**onResolve**(Runnable callback) Register a callback to be called when this Promise is resolved.	20

Promise<T>	**recover**(Function<Promise<?>,? extends T> recovery) Recover from a failure of this Promise with a recovery value.	23
Promise<T>	**recoverWith**(Function<Promise<?>,Promise<? extends T>> recovery) Recover from a failure of this Promise with a recovery Promise.	23
Promise<T>	**then**(Callback callback) Chain a new Promise to this Promise with a callback.	21
Promise<R>	**then**(Success<? super T,? extends R> success) Chain a new Promise to this Promise with a Success callback.	21
Promise<R>	**then**(Success<? super T,? extends R> success, Failure failure) Chain a new Promise to this Promise with Success and Failure callbacks.	20
Promise<T>	**timeout**(long milliseconds) Time out the resolution of this Promise.	24

Method Detail

isDone

```
boolean isDone()
```

Returns whether this Promise has been resolved.

This Promise may be successfully resolved or resolved with a failure.

Returns:

`true` if this Promise was resolved either successfully or with a failure; `false` if this Promise is unresolved.

---

getValue

```
T getValue()
    throws InvocationTargetException,
           InterruptedException
```

Returns the value of this Promise.

If this Promise is not `resolved`, this method must block and wait for this Promise to be resolved before completing.

If this Promise was successfully resolved, this method returns with the value of this Promise. If this Promise was resolved with a failure, this method must throw an `InvocationTargetException` with the `failure exception` as the cause.

Returns:

The value of this resolved Promise.

Throws:

`InvocationTargetException` - If this Promise was resolved with a failure. The cause of the `InvocationTargetException` is the failure exception.
`InterruptedException` - If the current thread was interrupted while waiting.

---

getFailure

```
Throwable getFailure()
             throws InterruptedException
```

Returns the failure of this Promise.

If this Promise is not `resolved`, this method must block and wait for this Promise to be resolved before completing.

If this Promise was resolved with a failure, this method returns with the failure of this Promise. If this Promise was successfully resolved, this method must return `null`.

Returns:
The failure of this resolved Promise or `null` if this Promise was successfully resolved.
Throws:
`InterruptedException` - If the current thread was interrupted while waiting.

---

onResolve

```
Promise<T> onResolve(Runnable callback)
```

Register a callback to be called when this Promise is resolved.

The specified callback is called when this Promise is resolved either successfully or with a failure.

This method may be called at any time including before and after this Promise has been resolved.

Resolving this Promise *happens-before* any registered callback is called. That is, in a registered callback, `isDone()` must return `true` and `getValue()` and `getFailure()` must not block.

A callback may be called on a different thread than the thread which registered the callback. So the callback must be thread safe but can rely upon that the registration of the callback *happens-before* the registered callback is called.

Parameters:
`callback` - A callback to be called when this Promise is resolved. Must not be `null`.
Returns:
This Promise.

---

then

```
Promise<R> then(Success<? super T,? extends R> success,
                Failure failure)
```

Chain a new Promise to this Promise with Success and Failure callbacks.

The specified `Success` callback is called when this Promise is successfully resolved and the specified `Failure` callback is called when this Promise is resolved with a failure.

This method returns a new Promise which is chained to this Promise. The returned Promise must be resolved when this Promise is resolved after the specified Success or Failure callback is executed. The result of the executed callback must be used to resolve the returned Promise. Multiple calls to this method can be used to create a chain of promises which are resolved in sequence.

If this Promise is successfully resolved, the Success callback is executed and the result Promise, if any, or thrown exception is used to resolve the returned Promise from this method. If this Promise is resolved with a failure, the Failure callback is executed and the returned Promise from this method is failed.

This method may be called at any time including before and after this Promise has been resolved.

Resolving this Promise *happens-before* any registered callback is called. That is, in a registered callback, `isDone()` must return `true` and `getValue()` and `getFailure()` must not block.

A callback may be called on a different thread than the thread which registered the callback. So the callback must be thread safe but can rely upon that the registration of the callback *happens-before* the registered callback is called.

Type Parameters:

`R` - The value type associated with the returned Promise.

Parameters:

`success` - A Success callback to be called when this Promise is successfully resolved. May be `null` if no Success callback is required. In this case, the returned Promise must be resolved with the value `null` when this Promise is successfully resolved.

`failure` - A Failure callback to be called when this Promise is resolved with a failure. May be `null` if no Failure callback is required.

Returns:

A new Promise which is chained to this Promise. The returned Promise must be resolved when this Promise is resolved after the specified Success or Failure callback, if any, is executed.

---

then

`Promise<R> then(Success<? super T,? extends R> success)`

Chain a new Promise to this Promise with a Success callback.

This method performs the same function as calling `then(Success, Failure)` with the specified Success callback and `null` for the Failure callback.

Type Parameters:

`R` - The value type associated with the returned Promise.

Parameters:

`success` - A Success callback to be called when this Promise is successfully resolved. May be `null` if no Success callback is required. In this case, the returned Promise must be resolved with the value `null` when this Promise is successfully resolved.

Returns:

A new Promise which is chained to this Promise. The returned Promise must be resolved when this Promise is resolved after the specified Success, if any, is executed.

See Also:

`then(Success, Failure)`

---

then

`Promise<T> then(Callback callback)`

Chain a new Promise to this Promise with a callback.

The specified `Callback` is called when this Promise is resolved either successfully or with a failure.

This method returns a new Promise which is chained to this Promise. The returned Promise must be resolved when this Promise is resolved after the specified callback is executed. If the callback throws an exception, the returned Promise is failed with that exception. Otherwise the returned Promise is resolved with this Promise.

This method may be called at any time including before and after this Promise has been resolved.

Resolving this Promise *happens-before* any registered callback is called. That is, in a registered callback, `isDone()` must return `true` and `getValue()` and `getFailure()` must not block.

A callback may be called on a different thread than the thread which registered the callback. So the callback must be thread safe but can rely upon that the registration of the callback *happens-before* the registered callback is called.

Parameters:
`callback` - A callback to be called when this Promise is resolved. Must not be `null`.
Returns:
A new Promise which is chained to this Promise. The returned Promise must be resolved when this Promise is resolved after the specified callback is executed.
Since:
1.1

---

filter

`Promise<T> filter(Predicate<? super T> predicate)`

Filter the value of this Promise.

If this Promise is successfully resolved, the returned Promise must either be resolved with the value of this Promise, if the specified Predicate accepts that value, or failed with a `NoSuchElementException`, if the specified Predicate does not accept that value. If the specified Predicate throws an exception, the returned Promise must be failed with the exception.

If this Promise is resolved with a failure, the returned Promise must be failed with that failure.

This method may be called at any time including before and after this Promise has been resolved.

Parameters:
`predicate` - The Predicate to evaluate the value of this Promise. Must not be `null`.
Returns:
A Promise that filters the value of this Promise.

---

map

`Promise<R> map(Function<? super T,? extends R> mapper)`

Map the value of this Promise.

If this Promise is successfully resolved, the returned Promise must be resolved with the value of specified Function as applied to the value of this Promise. If the specified Function throws an exception, the returned Promise must be failed with the exception.

If this Promise is resolved with a failure, the returned Promise must be failed with that failure.

This method may be called at any time including before and after this Promise has been resolved.

Type Parameters:
`R` - The value type associated with the returned Promise.
Parameters:
`mapper` - The Function that must map the value of this Promise to the value that must be used to resolve the returned Promise. Must not be `null`.
Returns:
A Promise that returns the value of this Promise as mapped by the specified Function.

---

flatMap

`Promise<R> flatMap(Function<? super T,Promise<? extends R>> mapper)`

FlatMap the value of this Promise.

If this Promise is successfully resolved, the returned Promise must be resolved with the Promise from the specified Function as applied to the value of this Promise. If the specified Function throws an exception, the returned Promise must be failed with the exception.

If this Promise is resolved with a failure, the returned Promise must be failed with that failure.

This method may be called at any time including before and after this Promise has been resolved.

Type Parameters:
`R` - The value type associated with the returned Promise.
Parameters:
`mapper` - The Function that must flatMap the value of this Promise to a Promise that must be used to resolve the returned Promise. Must not be `null`.
Returns:
A Promise that returns the value of this Promise as mapped by the specified Function.

---

recover

`Promise<T> recover(Function<Promise<?>,? extends T> recovery)`

Recover from a failure of this Promise with a recovery value.

If this Promise is successfully resolved, the returned Promise must be resolved with the value of this Promise.

If this Promise is resolved with a failure, the specified Function is applied to this Promise to produce a recovery value.

- If the recovery value is not `null`, the returned Promise must be resolved with the recovery value.
- If the recovery value is `null`, the returned Promise must be failed with the failure of this Promise.
- If the specified Function throws an exception, the returned Promise must be failed with that exception.

To recover from a failure of this Promise with a recovery value of `null`, the `recoverWith(Function)` method must be used. The specified Function for `recoverWith(Function)` can return `Promises.resolved(null)` to supply the desired `null` value.

This method may be called at any time including before and after this Promise has been resolved.

Parameters:
`recovery` - If this Promise resolves with a failure, the specified Function is called to produce a recovery value to be used to resolve the returned Promise. Must not be `null`.
Returns:
A Promise that resolves with the value of this Promise or recovers from the failure of this Promise.

---

recoverWith

`Promise<T> recoverWith(Function<Promise<?>,Promise<? extends T>> recovery)`

Recover from a failure of this Promise with a recovery Promise.

If this Promise is successfully resolved, the returned Promise must be resolved with the value of this Promise.

If this Promise is resolved with a failure, the specified Function is applied to this Promise to produce a recovery Promise.

- If the recovery Promise is not `null`, the returned Promise must be resolved with the recovery Promise.
- If the recovery Promise is `null`, the returned Promise must be failed with the failure of this Promise.
- If the specified Function throws an exception, the returned Promise must be failed with that exception.

This method may be called at any time including before and after this Promise has been resolved.

Parameters:

`recovery` - If this Promise resolves with a failure, the specified Function is called to produce a recovery Promise to be used to resolve the returned Promise. Must not be `null`.

Returns:

A Promise that resolves with the value of this Promise or recovers from the failure of this Promise.

---

fallbackTo

`Promise<T> fallbackTo(Promise<? extends T> fallback)`

Fall back to the value of the specified Promise if this Promise fails.

If this Promise is successfully resolved, the returned Promise must be resolved with the value of this Promise.

If this Promise is resolved with a failure, the successful result of the specified Promise is used to resolve the returned Promise. If the specified Promise is resolved with a failure, the returned Promise must be failed with the failure of this Promise rather than the failure of the specified Promise.

This method may be called at any time including before and after this Promise has been resolved.

Parameters:

`fallback` - The Promise whose value must be used to resolve the returned Promise if this Promise resolves with a failure. Must not be `null`.

Returns:

A Promise that returns the value of this Promise or falls back to the value of the specified Promise.

---

timeout

`Promise<T> timeout(long milliseconds)`

Time out the resolution of this Promise.

If this Promise is successfully resolved before the timeout, the returned Promise is resolved with the value of this Promise. If this Promise is resolved with a failure before the timeout, the returned Promise is resolved with the failure of this Promise. If the timeout is reached before this Promise is resolved, the returned Promise is failed with a `TimeoutException`.

Parameters:

`milliseconds` - The time to wait in milliseconds. Zero and negative time is treated as an immediate timeout.

Returns:

A Promise that is resolved when either this Promise is resolved or the specified timeout is reached.

Since:

1.1

---

delay

`Promise<T> delay(long milliseconds)`

Delay after the resolution of this Promise.

Once this Promise is resolved, resolve the returned Promise with this Promise after the specified delay.

Parameters:

`milliseconds` - The time to delay in milliseconds. Zero and negative time is treated as no delay.

Returns:

A Promise that is resolved with this Promise after this Promise is resolved and the specified delay has elapsed.

Since:

1.1

Class Promises

org.osgi.util.promise

```
java.lang.Object
  └ org.osgi.util.promise.Promises
```

```
public class Promises
extends Object
```

Static helper methods for Promises.

ThreadSafe

Method Summary	Page
static Promise<List<T>> **all**(Collection<Promise<S>> promises) Create a new Promise that is a latch on the resolution of the specified Promises.	27
static Promise<List<T>> **all**(Promise<? extends T>... promises) Create a new Promise that is a latch on the resolution of the specified Promises.	27
static Promise<T> **failed**(Throwable failure) Create a new Promise that has been resolved with the specified failure.	26
static Promise<T> **resolved**(T value) Create a new Promise that has been resolved with the specified value.	26

Method Detail

resolved

```
public static Promise<T> resolved(T value)
```

Create a new Promise that has been resolved with the specified value.

Type Parameters:
`T` - The value type associated with the returned Promise.
Parameters:
`value` - The value of the resolved Promise.
Returns:
A new Promise that has been resolved with the specified value.

---

failed

```
public static Promise<T> failed(Throwable failure)
```

Create a new Promise that has been resolved with the specified failure.

Type Parameters:
`T` - The value type associated with the returned Promise.
Parameters:
`failure` - The failure of the resolved Promise. Must not be `null`.
Returns:
A new Promise that has been resolved with the specified failure.

---

all

```
public static Promise<List<T>> all(Collection<Promise<S>> promises)
```

Create a new Promise that is a latch on the resolution of the specified Promises.

The new Promise acts as a gate and must be resolved after all of the specified Promises are resolved.

Type Parameters:

`T` - The value type of the List value associated with the returned Promise.
`S` - A subtype of the value type of the List value associated with the returned Promise.

Parameters:

`promises` - The Promises which must be resolved before the returned Promise must be resolved. Must not be `null` and all of the elements in the collection must not be `null`.

Returns:

A Promise that is resolved only when all the specified Promises are resolved. The returned Promise must be successfully resolved with a List of the values in the order of the specified Promises if all the specified Promises are successfully resolved. The List in the returned Promise is the property of the caller and is modifiable. The returned Promise must be resolved with a failure of `FailedPromisesException` if any of the specified Promises are resolved with a failure. The failure `FailedPromisesException` must contain all of the specified Promises which resolved with a failure.

---

all

```
@SafeVarargs
public static Promise<List<T>> all(Promise<? extends T>... promises)
```

Create a new Promise that is a latch on the resolution of the specified Promises.

The new Promise acts as a gate and must be resolved after all of the specified Promises are resolved.

Type Parameters:

`T` - The value type associated with the specified Promises.

Parameters:

`promises` - The Promises which must be resolved before the returned Promise must be resolved. Must not be `null` and all of the arguments must not be `null`.

Returns:

A Promise that is resolved only when all the specified Promises are resolved. The returned Promise must be successfully resolved with a List of the values in the order of the specified Promises if all the specified Promises are successfully resolved. The List in the returned Promise is the property of the caller and is modifiable. The returned Promise must be resolved with a failure of `FailedPromisesException` if any of the specified Promises are resolved with a failure. The failure `FailedPromisesException` must contain all of the specified Promises which resolved with a failure.

Interface Success

org.osgi.util.promise

Type Parameters:

`T` - The value type of the resolved Promise passed as input to this callback.
`R` - The value type of the returned Promise from this callback.

```
@org.osgi.annotation.versioning.ConsumerType
@FunctionalInterface
public interface Success
```

Success callback for a Promise.

A Success callback is registered with a `Promise` using the `Promise.then(Success)` method and is called if the Promise is resolved successfully.

This is a functional interface and can be used as the assignment target for a lambda expression or method reference.

ThreadSafe

Method Summary		Page
`Promise<R>`	**call**(`Promise<T>` resolved) Success callback for a Promise.	28

Method Detail

call

```
Promise<R> call(Promise<T> resolved)
          throws Exception
```

Success callback for a Promise.

This method is called if the Promise with which it is registered resolves successfully.

In the remainder of this description we will refer to the Promise returned by this method as the returned Promise and the Promise returned by `Promise.then(Success)` when this Success callback was registered as the chained Promise.

If the returned Promise is `null` then the chained Promise must resolve immediately with a successful value of `null`. If the returned Promise is not `null` then the chained Promise must be resolved when the returned Promise is resolved.

Parameters:

`resolved` - The successfully resolved `Promise`.

Returns:

The Promise to use to resolve the chained Promise, or `null` if the chained Promise is to be resolved immediately with the value `null`.

Throws:

`Exception` - The chained Promise must be failed with the thrown exception.

Class TimeoutException

org.osgi.util.promise

```
java.lang.Object
  └java.lang.Throwable
      └java.lang.Exception
          └org.osgi.util.promise.TimeoutException
```

All Implemented Interfaces:

Serializable

```
public class TimeoutException
extends Exception
```

Timeout exception for a Promise.

Since:

1.1

Constructor Summary	Page
`TimeoutException()` Create a new `TimeoutException`.	29

Constructor Detail

TimeoutException

```
public TimeoutException()
```

Create a new `TimeoutException`.

7 Considered Alternatives

None.

8 Security Considerations

The Promise API does not define any OSGi services nor does the API perform any privileged actions. Therefore, it has no security considerations.

9 Document Support

9.1 References

[1]. Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, March 1997.

[2]. Software Requirements & Specifications. Michael Jackson. ISBN 0-201-87712-0

9.2 Author’s Address

Name	BJ Hargrave
Company	IBM

9.3 Acronyms and Abbreviations

9.4 End of Document