Asynchronous Dynamic Invocation and Dispatch in Java

Previous Next

This page describes the asynchronous Java mapping for the `ice_invoke` proxy function and the `Blobject` class.

On this page:

Calling `ice_invoke` Asynchronously in Java

The asynchronous mapping for `ice_invoke` resembles that of the static AMI mapping. Multiple overloadings are provided to support callback styles and request contexts. The return value and the parameters `operation`, `mode`, and `inParams` have the same semantics as for the synchronous version of `ice_invoke`.

Basic Asynchronous Mapping for `ice_invoke` in Java

The basic mapping is shown below:

Java

```
package com.zeroc.Ice;

public interface Object
{
    public class Ice_invokeResult
    {
        public Ice_invokeResult()
        {
        }

        public Ice_invokeResult(boolean returnValue, byte[] outParams)
        {
            this.returnValue = returnValue;
            this.outParams = outParams;
        }

        public boolean returnValue;
        public byte[] outParams;
    }

    ...
}

public interface ObjectPrx
{
    java.util.concurrent.CompletableFuture<com.zeroc.Ice.Object.Ice_invokeResult> ice_invokeAsync(
        String operation,
        OperationMode mode,
        byte[] inParams);

    java.util.concurrent.CompletableFuture<com.zeroc.Ice.Object.Ice_invokeResult> ice_invokeAsync(
        String operation,
        OperationMode mode,
        byte[] inParams,
        java.util.Map<String, String> context);

    ...
}
```

As for statically-typed asynchronous invocations, the return value is a `CompletableFuture`. Its result is an instance of `Object.Ice_invokeResult`. Run-time exceptions cause the future to fail exceptionally, however user exceptions cause the future to succeed: the `returnValue` member of the `Ice_invokeResult` object will be set to `false`, and the `outParams` member contains the encapsulated user exception data.

Java Compat

```
Ice.AsyncResult begin_ice_invoke(
    String operation,
    Ice.OperationMode mode,
    byte[] inParams);

Ice.AsyncResult begin_ice_invoke(
    String operation,
    Ice.OperationMode mode,
    byte[] inParams,
    java.util.Map<String, String> __context);

boolean end_ice_invoke(Ice.ByteSeqHolder outParams, Ice.AsyncResult __result);
```

User exceptions are handled differently than for static asynchronous invocations. Calling `end_ice_invoke` can raise run-time exceptions but never raises user exceptions. Instead, the boolean return value of `end_ice_invoke` indicates whether the operation completed successfully (true) or raised a user exception (false). If the return value is true, the byte sequence contains an encapsulation of the results; otherwise, the byte sequence contains an encapsulation of the user exception.

Back to Top ^

Generic Asynchronous Callback Mapping for `ice_invoke` in Java Compat

The generic callback API is also available:

Java Compat

```
Ice.AsyncResult begin_ice_invoke(
    String operation,
    Ice.OperationMode mode,
    byte[] inParams,
    Ice.Callback cb);

Ice.AsyncResult begin_ice_invoke(
    String operation,
    Ice.OperationMode mode,
    byte[] inParams,
    java.util.Map<String, String> context,
    Ice.Callback cb);
```

Refer to the static AMI mapping for an example of subclassing `Ice.Callback`.

Back to Top ^

Type-Safe Asynchronous Callback Mapping for `ice_invoke` in Java Compat

The type-safe callback API looks as follows:

Java Compat

```
Ice.AsyncResult begin_ice_invoke(
    String operation,
    Ice.OperationMode mode,
    byte[] inParams,
    Ice.Callback_Object_ice_invoke cb);

Ice.AsyncResult begin_ice_invoke(
    String operation,
    Ice.OperationMode mode,
    byte[] inParams,
    java.util.Map<String, String> context,
    Ice.Callback_Object_ice_invoke cb);
```

Callers must supply a subclass of `Ice.Callback_Object_ice_invoke`:

Java Compat

```
package Ice;

public abstract class Callback_Object_ice_invoke extends ...
{
    public abstract void response(boolean ret, byte[] outParams);

    public abstract void exception(LocalException ex);
}
```

The boolean argument to `response` indicates whether the operation completed successfully (true) or raised a user exception (false). If the return value is true, the byte sequence contains an encapsulation of the results; otherwise, the byte sequence contains an encapsulation of the user exception.

Back to Top ^

Subclassing `BlobjectAsync` in Java

`BlobjectAsync` is the name of the asynchronous counterpart to `Blobject`:

Java

```
package com.zeroc.Ice;

public interface BlobjectAsync extends com.zeroc.Ice.Object
{
    java.util.concurrent.CompletionStage<Object.Ice_invokeResult> ice_invokeAsync(byte[] inEncaps, Current
current)
        throws UserException;
}
```

To implement asynchronous dynamic dispatch, a server must implement `BlobjectAsync` and define `ice_invokeAsync`.

The return value for successful completion, or for a user exception, is a `CompletionStage` whose result is an instance of `Object.Ice_invokeResult`. The servant may optionally raise a user exception directly and the Ice run time will marshal it for you.

Java Compat

```
package Ice;

public abstract class BlobjectAsync extends Ice.ObjectImpl
{
    public abstract void ice_invoke_async(
        Ice.AMD_Object_ice_invoke cb,
        byte[] inParams,
        Ice.Current current);

    // ...
}
```

To implement asynchronous dynamic dispatch, a server must subclass `BlobjectAsync` and override `ice_invoke_async`.

As with any other asynchronous operation, the first argument to the servant's member function is always a callback object. In this case, the callback object is of type `Ice.AMD_Object_ice_invoke`, shown here:

```
package Ice;

public interface AMD_Object_ice_invoke
{
    void ice_response(boolean result, byte[] outParams);
    void ice_exception(java.lang.Exception ex);
}
```

Upon a successful invocation, the servant must invoke `ice_response` on the callback object, passing `true` as the first argument and encoding the encapsulated operation results into `outParams`. To report a user exception, the servant invokes `ice_response` with `false` as the first argument and the encapsulated form of the exception in `outParams`. Alternatively, the servant can pass a user exception instance to `ice_exception`.

Back to Top ^

See Also

- Asynchronous Method Invocation (AMI) in Java
- Request Contexts
- Dynamic Invocation and Dispatch Overview

Previous

Next