JavaScript Mapping for Exceptions

Previous Next

On this page:

JavaScript Mapping for User Exceptions

Here is a fragment of the Slice definition for our world time server once more:

Slice

```
exception GenericError
{
    string reason;
}
exception BadTimeVal extends GenericError {}
exception BadZoneName extends GenericError {}
```

These exception definitions map as follows:

JavaScript

```
M.GenericError = class extends Ice.UserException
{
    constructor(reason = "", _cause = ""){ ... }
    ice_name()
    {
        return "M::GenericError";
    }
}

M.BadTimeVal = class extends M.GenericError
{
    constructor(reason, _cause = ""){ ... }
    ice_name()
    {
        return "M::BadTimeVal";
    }
}

M.BadZoneName= class extends M.GenericError
{
    constructor(reason, _cause = ""){ ... }
    ice_name()
    {
        return "M::BadZoneName";
    }
}
```

Each Slice exception maps to a JavaScript class with the same name. Each data member of an exception corresponds to a property in the JavaScript type.

The inheritance structure of the Slice exceptions is preserved for the generated types, so the prototypes for `BadTimeVal` and `BadZoneName` inherit from `GenericError`.

Each exception also defines the `ice_name` member function, which returns the name of the exception.

All user exceptions ultimately derive from the base type `Ice.UserException`. This allows you to handle any user exception generically as an `Ice.UserException`. Similarly, you can handle any Ice run-time exceptions as an `Ice.LocalException`, and you can handle any Ice exception as an `Ice.Exception`.

Note that the generated exception types may contain other properties and functions that are not shown. However, these elements are internal to the JavaScript mapping and are not meant to be used by application code.

Back to Top ^

JavaScript Constructors for User Exceptions

The generated constructor has one parameter for each data member. This allows you to construct and initialize an instance in a single statement (instead of first having to construct the instance and then assign to its members). For derived exceptions, the constructor accepts one argument for each base exception member, plus one argument for each derived exception member, in base-to-derived order.

The constructor assigns a default value to each property appropriate for the type of its corresponding member:

Data Member Type	Default Value
`string`	Empty string
`enum`	First enumerator in enumeration
`struct`	Default-constructed value
Numeric	Zero
`bool`	False
`sequence`	Null
`dictionary`	Null
`class/interface`	Null

If you wish to ensure that data members of primitive and enumerated types are initialized to specific values, you can declare default values in your Slice definition. The constructor initializes each of these data members to its declared value instead.

The generated constructor accepts an optional trailing argument representing the entity that caused the exception. This value is typically an instance of `Error`, but that is not a requirement. The value is used to initialize the `ice_cause` property inherited from `Ice.Exception`.

Back to Top ^

JavaScript Mapping for Run-Time Exceptions

The Ice run time throws run-time exceptions for a number of pre-defined error conditions. All run-time exceptions directly or indirectly derive from `Ice.LocalException` (which, in turn, derives from `Ice.Exception`).

Recall the inheritance diagram for user and run-time exceptions. By catching exceptions at the appropriate point in the hierarchy, you can handle exceptions according to the category of error they indicate:

- `Ice.Exception`
  This is the root of the inheritance tree for both run-time and user exceptions.
- `Ice.LocalException`
  This is the root of the inheritance tree for run-time exceptions.
- `Ice.UserException`
  This is the root of the inheritance tree for user exceptions.
- `Ice.TimeoutException`
  This is the base exception for both operation-invocation and connection-establishment timeouts.
- `Ice.ConnectTimeoutException`
  This exception is raised when the initial attempt to establish a connection to a server times out.

For example, a `ConnectTimeoutException` can be handled as `ConnectTimeoutException`, `TimeoutException`, `LocalException`, or `Exception`.

You will probably have little need to catch run-time exceptions as their most-derived type and instead catch them as `LocalException`; the fine-grained error handling offered by the remainder of the hierarchy is of interest mainly in the implementation of the Ice run time. Exceptions to this rule are the exceptions related to facet and object life cycles, which you may want to catch explicitly. These exceptions are `FacetNotExistException` and `ObjectNotExistException`, respectively.

Back to Top ^

See Also

- User Exceptions
- Run-Time Exceptions
- JavaScript Mapping for Modules
- JavaScript Mapping for Built-In Types
- JavaScript Mapping for Enumerations
- JavaScript Mapping for Structures
- JavaScript Mapping for Sequences
- JavaScript Mapping for Dictionaries
- JavaScript Mapping for Constants

Previous

Next