Classes with Operations

Previous Next

Classes, in addition to data members, can have operations.

> **Deprecated Feature**
>
> Operations on classes are deprecated as of Ice 3.7. Skip this page unless you need to communicate with old applications that rely on this feature.

The syntax for operation definitions in classes is identical to the syntax for operations in interfaces. For example, we can modify the expression tree from Self-Referential Classes as follows:

Slice

```
module M
{
    enum UnaryOp { UnaryPlus, UnaryMinus, Not }
    enum BinaryOp { Plus, Minus, Multiply, Divide, And, Or }

    class Node
    {
        idempotent long eval();
    }

    class UnaryOperator extends Node
    {
        UnaryOp operator;
        Node operand;
    }

    class BinaryOperator extends Node
    {
        BinaryOp op;
        Node operand1;
        Node operand2;
    }

    class Operand
    {
        long val;
    }
}
```

The only change compared to the version in Self-Referential Classes is that the `Node` class now has an `eval` operation. The semantics of this are as for a virtual member function in C++: each derived class inherits the operation from its base class and can choose to override the operation's definition. For our expression tree, the `Operand` class provides an implementation that simply returns the value of its `val` member, and the `UnaryOperator` and `BinaryOperator` classes provide implementations that compute the value of their respective subtrees. If we call `eval` on the root node of an expression tree, it returns the value of that tree, regardless of whether we have a complex expression or a tree that consists of only a single `Operand` node.

Operations on classes are normally executed in the caller's address space, that is, operations on classes are *local* operations that do not result in a remote procedure call.

> It is also possible to invoke an operation on a remote class instance.

Of course, this immediately raises an interesting question: what happens if a client receives a class instance with operations from a server, but client and server are implemented in different languages? Classes with operations require the receiver to supply a factory for instances of the class. The Ice run time only marshals the data members of the class. If a class has operations, the receiver of the class must provide a class factory that can instantiate the class in the receiver's address space, and the receiver is responsible for providing an implementation of the class's operations.

Therefore, if you use classes with operations, it is understood that client and server each have access to an implementation of the class's operations. No code is shipped over the wire (which, in an environment of heterogeneous nodes using different operating systems and languages is infeasible).

Back to Top ^

See Also

- Self-Referential Classes
- Pass-by-Value Versus Pass-by-Reference

Previous

Next