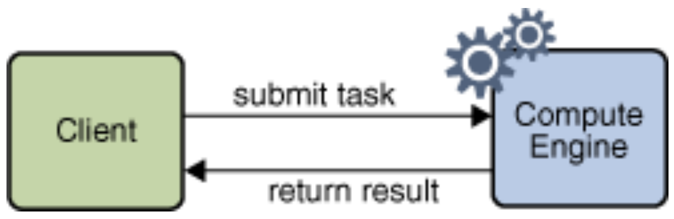
ORACLE Java Documentation

The Java™ Tutorials

Trail: RMI
Section: Writing an RMI Server

Designing a Remote Interface

At the core of the compute engine is a protocol that enables tasks to be submitted to the compute engine, the compute engine to run those tasks, and the results of those tasks to be returned to the client. This protocol is expressed in the interfaces that are supported by the compute engine. The remote communication for this protocol is illustrated in the following figure.



Each interface contains a single method. The compute engine's remote interface, `Compute`, enables tasks to be submitted to the engine. The client interface, `Task`, defines how the compute engine executes a submitted task.

The `compute.Compute` interface defines the remotely accessible part, the compute engine itself. Here is the source code for the `Compute` interface:

```
package compute;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Compute extends Remote {
    <T> T executeTask(Task<T> t) throws RemoteException;
}
```

By extending the interface `java.rmi.Remote`, the `Compute` interface identifies itself as an interface whose methods can be invoked from another Java virtual machine. Any object that implements this interface can be a remote object.

As a member of a remote interface, the `executeTask` method is a remote method. Therefore, this method must be defined as being capable of throwing a `java.rmi.RemoteException`. This exception is thrown by the RMI system from a remote method invocation to indicate that either a communication failure or a protocol error has occurred. A `RemoteException` is a checked exception, so any code invoking a remote method needs to handle this exception by either catching it or declaring it in its `throws` clause.

The second interface needed for the compute engine is the `Task` interface, which is the type of the parameter to the `executeTask` method in the `Compute` interface. The `compute.Task` interface defines the interface between the compute engine and the work that it needs to do, providing the way to start the work. Here is the source code for the `Task` interface:

```
package compute;

public interface Task<T> {
    T execute();
}
```

The `Task` interface defines a single method, `execute`, which has no parameters and throws no exceptions. Because the interface does not extend Remote, the method in this interface doesn't need to list `java.rmi.RemoteException` in its throws clause.

The `Task` interface has a type parameter, T, which represents the result type of the task's computation. This interface's `execute` method returns the result of the computation and thus its return type is T.

The `Compute` interface's `executeTask` method, in turn, returns the result of the execution of the `Task` instance passed to it. Thus, the `executeTask` method has its own type parameter, T, that associates its own return type with the result type of the passed `Task` instance.

RMI uses the Java object serialization mechanism to transport objects by value between Java virtual machines. For an object to be considered serializable, its class must implement the `java.io.Serializable` marker interface. Therefore, classes that implement the `Task` interface must also implement `Serializable`, as must the classes of objects used for task results.

Different kinds of tasks can be run by a `Compute` object as long as they are implementations of the `Task` type. The classes that implement this interface can contain any data needed for the computation of the task and any other methods needed for the computation.

Here is how RMI makes this simple compute engine possible. Because RMI can assume that the `Task` objects are written in the Java programming language, implementations of the `Task` object that were previously unknown to the compute engine are downloaded by RMI into the compute engine's Java virtual machine as needed. This capability enables clients of the compute engine to define new kinds of tasks to be run on the server machine without needing the code to be explicitly installed on that machine.

The compute engine, implemented by the `ComputeEngine` class, implements the `Compute` interface, enabling different tasks to be submitted to it by calls to its `executeTask` method. These tasks are run using the task's implementation of the `execute` method and the results, are returned to the remote client.

Previous page: Writing an RMI Server
Next page: Implementing a Remote Interface