Assignment Overview

Nuozhou Wang

This project requires us to extend project1 from two aspects. Java RMI should be set up to enable the client-server communication. The server should also be able to handle multiple running instances of clients at once with multi-threading. For my understanding, the purpose and scopes of this project is to practice the following skills:

- RMI (Remote Method Invocation) and RPC (Remote Procedure Call): RPC and RMI are the mechanism which enable clients to invoke the procedures or methods from the server. The main difference between the two mechanisms is that RPC only supports procedural programming thus is C based whereas RMI supports object-oriented programming and is Java based. Moreover, the parameters passed to RPC consist of ordinary data structures and the ones passed to RMI consist of objects. With RPC, we can call remote functions exported into the server, and in RMI, we will have references to remote objects to invoke their methods.

- Implementation of Java RMI. RMI is widely used in distributed system and allows an object residing in one JVM to access or invoke an object running on another JVM. In an RMI application, a remote object is created with a remote interface implemented. The server side will have the reference to the remote object. The client program will request the remote object and invoke its methods. To be noted, RMI registry is a namespace where all server objects are placed. The server will register a bind name for its object with the RMI registry with bind() or rebind() methods. The client fetches the object from the registry with lookup() method.

- Handling multiple clients' requests: In the case of socket-based communication, multi-threading should be written in the server side to enable it to connect with multiple clients. RMI is designed to be multithreaded. Multiple clients can execute methods on the remote object simultaneously, which means no extra effort should be made to implement the multi-threading by our own. But we should pay attention to the mutual exclusion to make our code thread safe.

- Handle timeout cases: The client should be able to handle unresponsive server failure by using some timeout mechanisms. If the client does not receive a response to a particular request, it should be printed out in the client log and start to send the next request. With multi-threading implementation, we can assign a new thread and introduce a time-out mechanism for each query on the client's side.

All in all, this project requires a wide knowledge of RMI. It's a good practice to learn how to implement RMI and client's side multi-threading.

Technical impression

Nuozhou Wang

Detailed implementation will be described as follows:

- RMI: To write an RMI Java application, we should define the remote interface, write the remote object that implement the remote interface, write the server program, write the client program, compile and execute the application. To create the remote interface, it should extend the predefined interface Remote which belongs to the rmi package. All the methods which will be accessed by clients should be stated in the interface. An exception named RemoteException should be thrown in case there is network issues during remote calls. A remote object should be developed to implement the remote interface. In the server class, a remote object will be created and bind to the RMI registry. Client class will get the object by looking up the class Registry with a String value representing the bind name as a parameter. To be noted, the lookup() returns an object of Remote, we should down cast it to our remote interface type.

- Synchronized HashMap: By default, Java HashMap is not synchronized. With multi-threading implementation, if we add/remove key-value pairs from a HashMap, we may end up having inconsistent state of the map. Instead, we can use ConcurrentHashMap class in the concurrent environment. ConcurrentHashMap is thread-safe and multiple threads can operate on the single object without any complications. In details, any number of threads can perform retrieval operation at a time on the ConcurrentHashMap, but there will be a segment locking if any updating operation is under processing.

- Callable: Defining tasks using Runnable is convenient, but it is limited by the fact that the tasks cannot return a result. In Java, there is a Callable interface to define tasks that return a result and throw a checked exception. Callable interface has a single method call() which is meant to include the code that will be executed by the thread. The submit() method of executor service will submit the task for execution by a thread. And it will return a special type of value called a Future which can be used to fetch the result of the task when it is available. Once a future is given, other tasks can be executed as needed in parallel while the submitted task is executing, and then use future.get() method to retrieve the result of the future.

- Adding Timeouts: The future.get() method blocks and waits for the task to complete. If a remote service is executed in the callable task and the remote service is down, then future.get() will block forever, which will make the application unresponsive. To solve this problem, we can add a timeout in the get() method. The future.get() method will throw a TimeoutException if the task is not completed within the specified time. In this project, it client receive no response from server in certain time span, TimeoutException will be thrown, client will move to the next query. But we should be noted that server side will still keep processing the last request in this case.