project2_executive_summary_weiqing_gao

Assignment Overview

From my perspective, the objective of the project 2 is to learn how to implement RMI. The details include the remote interface and how to implement it, how to deal with remote objects compared with local objects, especially the use of remote object references. Another goal is to touch the differences between TCP/UDP and RMI. As the upper layer in the architecture, the RMI hides the underlying network details, for example, we don't need to use raw sockets (TCP/UCP) and manual message parsing as we did in project 1. The scope includes 1. RMI, which means a client can invoke any of the three operations remotely on the server just as if it were a local method call (transparency); 2. multi-threading, which means the server must be able to handle concurrent requests from multiple clients at the same time; 3. mutual exclusion, which means that the server must avoid race conditions when implementing the concurrency.

Technical Impression

During the implementation, I found it is pretty important to clarify how exceptions are thrown and interpreted on the client side. Although both serializaiton and deserialization of the arguments and results of remote invocations are generally carried out automatically by the middleware, it requires me to properly define the exceptions and method signatures to ensure clients receive clear error messages, especially in this project, there is a `MalformedRequestException` in addition to the `RemoteException`. What's more, there are also potential improvement here since I did not take internal server errors, concurrency conflict errors (although I used lock to prevent it) and other potential errors.

I think it is also crucial to design the proper concurrency control. Because Java RMI automatically spawns multiple threads to handle incoming remote calls from clients, I must implemennt concurrency control, especially when my server uses a shared data structure (the HashMap used to store key-value pair). Based on the fact that this project involves only three operations, I used `ReentrantLock` to avoid the race conditions. But if the throughput is high, then it will not perform well at efficiency. A more detailed lock, for example the read-write locks, can be used to allow certain operations to be parallelized safely. Or using the thread pools to limit the maximum concurrent throughout and implement more complex scheduling.

Screenshots - RMI Client

```
weiqinggao@Weiqings-MBP src % java KeyValueStoreRMIClient 127.0.0.1 1099
Connected to RMI Key-Value Store on 127.0.0.1:1099
Pre-populated: key_0: value_0
Pre-populated: key_1: value_1
Pre-populated: key_2: value_2
Pre-populated: key_3: value_3
Pre-populated: key_4: value_4
Enter command (PUT/GET/DELETE) or 'exit': PUT
Enter key: Lexus
Enter value: LS430
PUT succeeded.
Enter command (PUT/GET/DELETE) or 'exit': put
Enter key: Lexus
Enter value: LS460
Operation failed: The key "Lexus" already exists.
Enter command (PUT/GET/DELETE) or 'exit': PUT
Enter key: Lincoln
Enter value: Continental
PUT succeeded.
Enter command (PUT/GET/DELETE) or 'exit': PUT
Enter key: Genesis
Enter value: G90
PUT succeeded.
Enter command (PUT/GET/DELETE) or 'exit': PUT
Enter key: Ford
Enter value: Tarus
PUT succeeded.
Enter command (PUT/GET/DELETE) or 'exit': PUT
Enter key: Acura
Enter value: TLX
PUT succeeded.
Enter command (PUT/GET/DELETE) or 'exit': GET
Enter key: Ford
GET result: Tarus
Enter command (PUT/GET/DELETE) or 'exit': get
Enter key: key1
```

```
GET result: NOT_FOUND
Enter command (PUT/GET/DELETE) or 'exit': get
Enter key: key_1
GET result: value_1
Enter command (PUT/GET/DELETE) or 'exit': Get
Enter key: Genesis
GET result: G90
Enter command (PUT/GET/DELETE) or 'exit': GET
Enter key: Lexus
GET result: LS430
Enter command (PUT/GET/DELETE) or 'exit': GET
Enter key: Lincoln
GET result: Continental
```

```
Enter command (PUT/GET/DELETE) or 'exit': GET
Enter key: Lincoln
GET result: Continental
Enter command (PUT/GET/DELETE) or 'exit': DELETE
Enter key: key_2
DELETE succeeded.
Enter command (PUT/GET/DELETE) or 'exit': Get
Enter key: key_2
GET result: NOT_FOUND
Enter command (PUT/GET/DELETE) or 'exit': DELETE
Enter key: Lexus
DELETE succeeded.
Enter command (PUT/GET/DELETE) or 'exit': DELETE
Enter key: Genesis
DELETE succeeded.
Enter command (PUT/GET/DELETE) or 'exit': Delete
Enter key: Lincoln
DELETE succeeded.
Enter command (PUT/GET/DELETE) or 'exit': Delete
Enter key: Ford
DELETE succeeded.
Enter command (PUT/GET/DELETE) or 'exit': Delete
Enter key: Volvo
Operation failed: Key not found: Volvo
Enter command (PUT/GET/DELETE) or 'exit': exit
Exiting client...
```

Screenshots - RMI Server

```
Weiqings-MBP src % java KeyValueStoreRMIServer 1099
RMI Key-Value Store Server is running on port 1099...
Press Ctrl+C to stop.
[Server] PUT: key_0 => value_0
[Server] PUT: key_1 => value_1
[Server] PUT: key_2 => value_2
[Server] PUT: key_3 => value_3
[Server] PUT: key_4 => value_4
[Server] PUT: Lexus => LS430
[Server] PUT: Lincoln => Continental
[Server] PUT: Genesis => G90
[Server] PUT: Ford => Tarus
[Server] PUT: Acura => TLX
[Server] GET: Ford => Tarus
[Server] GET: key1 => NOT_FOUND
[Server] GET: key_1 => value_1
[Server] GET: Genesis => G90
[Server] GET: Lexus => LS430
[Server] GET: Lincoln => Continental
[Server] DELETE: key_2 => OK
[Server] GET: key_2 => NOT_FOUND
[Server] DELETE: Lexus => OK
[Server] DELETE: Genesis => OK
[Server] DELETE: Lincoln => OK
[Server] DELETE: Ford => OK
^C%
```