

Assignment 4



TABLE OF CONTENTS

ASSIGNMENT DISCUSSION 1

Java RMI Concurrency..... 1

FIGURES, SAMPLE CODES..... 4

UML Diagram 4

Sample Codes..... 5

SAMPLE RUNS 8

Screenshots..... 7

Discussion12

CONCLUSION, REFERENCES..... 13

Conclution13

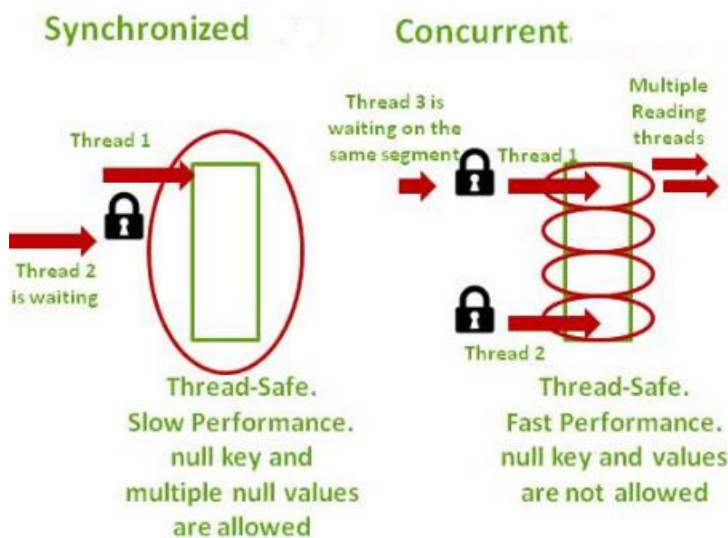
References13

ASSIGNMENT

DISCUSSION

The purpose of this practice is designing Domain Model of online Marketplace using Java RMI and MVC design pattern and by leveraging useful and proper design patterns.

JAVA RMI CONCURRENCY



www.codepumpkin.com

Above image give us a broad view of concurrent system vs synchronized system. Concurrent programming has two main benefits:

First, it allows natural solutions to software design problems that are inherently parallel or distributed. Second, concurrent programs offer better efficiency than sequential programs. However, concurrent programming poses many challenges that do not exist in the sequential setting. For instance, the processes in the system may livelock, diverge, or even deadlock. The Java Remote Method Invocation (RMI) package facilitates the implementation of concurrent applications in which, for instance, the processes reside on different hosts and communicate over the internet. More precisely, it hides the details of network communication. Unfortunately, it does not relieve the programmer from the potential pitfalls of controlling the concurrent access to remote objects. Consequently, RMI applications are prone to the same problems as concurrent programs in general.

In our application, since the system is distributed, logically, blocking in a remote object is simple. Suppose that client A calls a synchronized method of a remote object. To make access to remote objects look always exactly the same as to local objects, it would be necessary to block A in the client-side stub that implements the object's interface and to which A has direct access. Likewise, another client on a different machine would need to be blocked locally as well before its request can be sent to the server. The consequence is that we need to synchronize different clients at different machines.

An alternative approach would be to allow blocking only at the server. In principle, this works fine, but problems arise when a client crashes while its invocation is being handled by the server. We may require relatively sophisticated protocols to handle this situation, and which that may significantly affect the overall performance of remote method invocations.

Therefore, the designers of Java RMI have chosen to restrict blocking on remote objects only to the proxies (Wollrath et al., 1996). This means that threads in the same process will be prevented from concurrently accessing the same remote object, but threads in

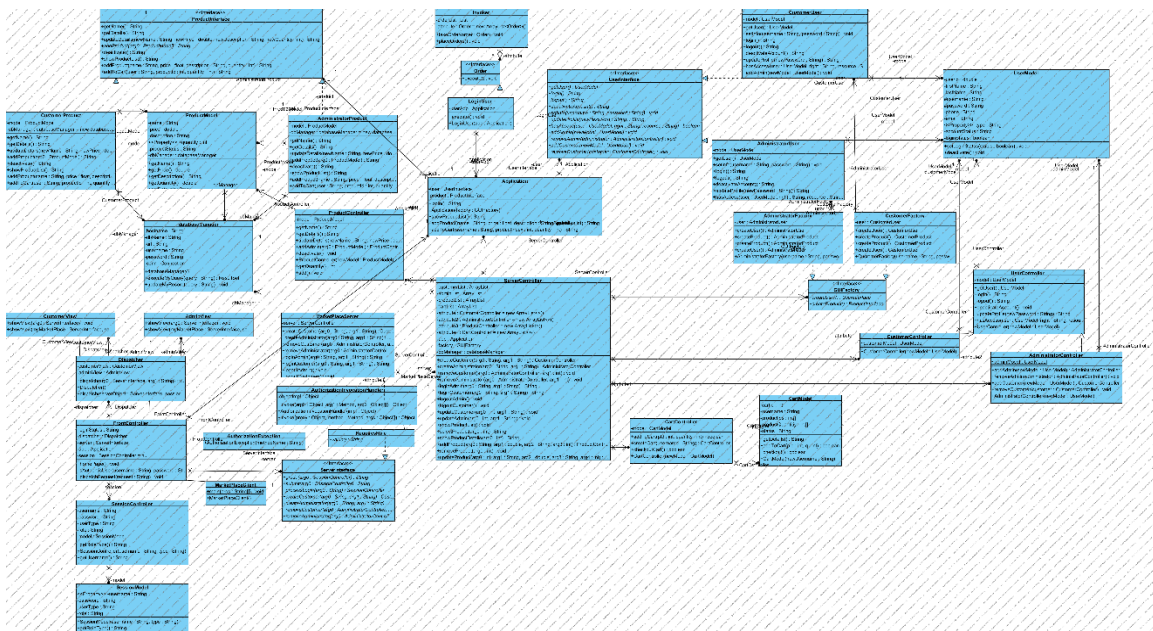
different processes will not. Obviously, these synchronization semantics are tricky: at the syntactic level (ie, when reading source code) we may see a nice, clean design. Only when the distributed application is actually executed, unanticipated behavior may be observed that should have been dealt with at design time.

If I implement my RMI remote methods as synchronized, RMI does not provide such guarantee that they are mutually exclusive on its own (unlike EJB) and two calls on the same remote object may be executed concurrently unless you implement some synchronization. Also, if I have a method that the server executes periodically. It is used to do cleanups. I have to make sure that this particular method does not execute when there is any RMI method being run/used by remote clients. If the cleanup job is in another class, I will need to define a lock that you will share between the remote object and the cleanup job. In the remote object, define an instance variable that I will use as a lock.

FIGURE, SAMPLE CODES

In following we provide you a Class Diagram of our system along with some sample codes.

UML DIAGRAM



The original UML diagram has been provided in Documentation directory. As you see, in our proposed architecture, we introduced a

new class named SessionModel and also a temporary databaseManager class.

SAMPLE CODES

In following we provide a sample code of some of the different new functionalities of the system:

Add a product to customer's cart:

```
public String addToCart(String user, int productId, int quantity) {  
    String commandStatus = null;  
    try{  
        String product_stm = String.format("UPDATE product  
SET quantity = (quantity - %s) WHERE productId  
= %s",quantity,productId);  
        dbManager.updateMyRecord(product_stm);  
  
        String cart_stm = String.format("INSERT INTO  
cart(username, productId, quantity) VALUES ('%s',%s,%s)", user,  
quantity,productId);  
        dbManager.updateMyRecord(cart_stm);  
        commandStatus = "Product has been added to your  
cart successfully!";  
    }catch(Exception e){  
        commandStatus = "Please try again later!";  
        System.out.println("Database Exception" +  
e.getMessage());  
    }  
}
```


- *Browse products:*

```
public String showProductList(){
    StringBuilder str = new StringBuilder();
    str.append("");
    try{
        ResultSet rs = dbManager.executeQuery("SELECT * FROM
`product` WHERE `status` = 'active'");

        while (rs.next()) {
            int id = rs.getInt("productId");
            String productName = rs.getString("name");
            str.append(id);
            str.append(" ");
            str.append(productName);
            str.append("\n");
            System.out.println("ID: " + id + "\t" + "Name: " +
productName);
        }
    }
```

- *Concurrency simulating:*

```
public void concurrencyTestSync(String host){

    try{

        System.out.println(host + "synchronized concurrencyTest
call...."); Thread.sleep(6000);

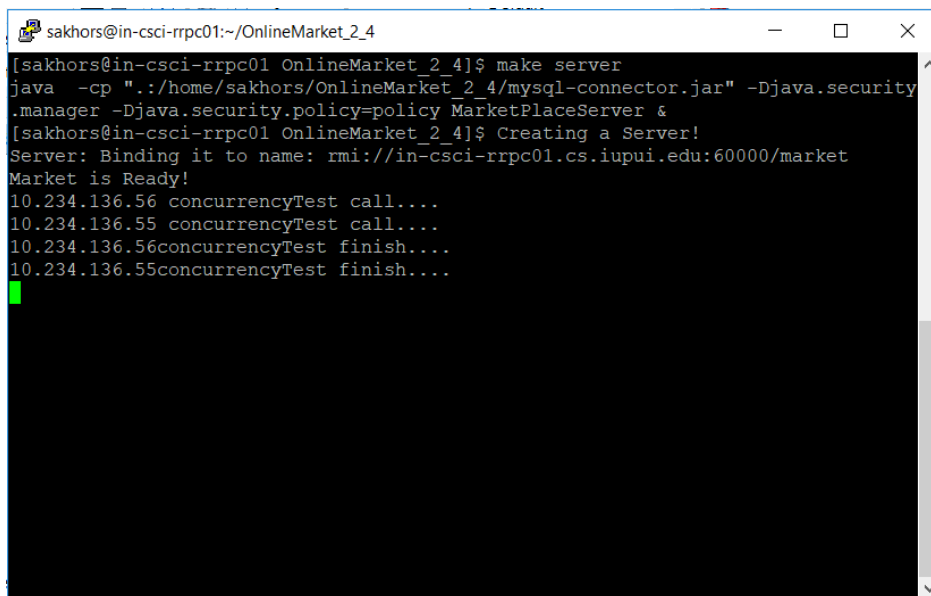
        System.out.println( host + "synchronized concurrencyTest
finish" + "\n");
    }
```

SAMPLE RUNS

In this section you'll see screenshots of sample runs. The system is minimal, so there is no GUI, instead every interaction happen using command prompt.

SCREENSHOTS

Here is a sample run of simulating concurrency using two client from two different machine:



```
sakhors@in-csci-rrpc01:~/OnlineMarket_2_4
[sakhors@in-csci-rrpc01 OnlineMarket_2_4]$ make server
java -cp ".:~/home/sakhors/OnlineMarket_2_4/mysql-connector.jar" -Djava.security
.manager -Djava.security.policy=policy MarketPlaceServer &
[sakhors@in-csci-rrpc01 OnlineMarket_2_4]$ Creating a Server!
Server: Binding it to name: rmi://in-csci-rrpc01.cs.iupui.edu:60000/market
Market is Ready!
10.234.136.56 concurrencyTest call....
10.234.136.55 concurrencyTest call....
10.234.136.56concurrencyTest finish....
10.234.136.55concurrencyTest finish....
```

This is a sample run of browsing system products from database:

```
Using username "sakhors".
sakhors@in-csci-rrpc01.cs.iupui.edu's password:
Last login: Tue Apr  3 22:58:14 2018 from 140-182-64-126.ssl-vpn.iupui.edu
[sakhors@in-csci-rrpc01 ~]$ cd OnlineMarket_2_4/
[sakhors@in-csci-rrpc01 OnlineMarket_2_4]$ make client
java -cp ".:~/home/sakhors/OnlineMarket_2_4/mysql-connector.jar" -Djava.security
.manager -Djava.security.policy=policy MarketPlaceClient
Welcom to marketplace
*****
1- Login as admin
2- Login as customer
3- Exit Marketplace!
*****
Please choose an option:      2
Please enter your username:   customer
Please enter your password:   customer
User is authenticated successfully.
Page Requested: CUSTOMER
Welcome to the Customer Page!
Customer is Authenticated!!!
*****
* 1- Show product list      *
* 2- Select a product       *
* 3- Show Product Details   *
* 4- Add product to cart (Purchase a product) *
* 5- Show cart details      *
* 6- Logout from system     *
*****
Please choose an option:      1
Showing product list...
1 Shoe
2 TV
3 Cellphone
4 Bag
5 new product
6 test product
7 test
*****
* 1- Show product list      *
* 2- Select a product       *
* 3- Show Product Details   *
* 4- Add product to cart (Purchase a product) *
* 5- Show cart details      *
* 6- Logout from system     *
*****
Please choose an option:      █
```

This a sample run of adding a new product by admin user:

```

java -cp ".:~/home/sakhors/OnlineMarket_2_4/mysql-connector.jar" -Djava.security.manager -Djava.security.policy=policy MarketPlaceClient
Welcom to marketplace
*****
1- Login as admin
2- Login as customer
3- Exit Marketplace!
*****
Please choose an option: 1
Please enter your username: admin
Please enter your password: admin
Session is administrator
User is authenticated successfully.
Page Requested: ADMIN
Welcome to the Admin Page!
Administrator is Authenticated!!!
*****
* 1- Create a Customer *
* 2- Update a Customer *
* 3- Remove a Customer *
* ---- *
* 4- Create an Administrator *
* 5- Update an Administrator *
* 6- Remove an Administrator *
* ---- *
* 7- Add new product *
* 8- Update a product *
* 9- Remove a product *
* 10- Logout from system *
*****
Please choose an option: 7
Adding new product...
Please enter product name: test sample
Please enter product price: 12.43
Please enter product decription: this a test
Please enter product quantity: 5
NAME:::test sample*****
* 1- Create a Customer *
* 2- Update a Customer *
* 3- Remove a Customer *
* ---- *
* 4- Create an Administrator *
* 5- Update an Administrator *
* 6- Remove an Administrator *
* ---- *
* 7- Add new product *
* 8- Update a product *
* 9- Remove a product *
* 10- Logout from system *
*****
Please choose an option:

```

And this is a sample run of adding a product to the user shopping cart (Purchase):

```

* 3- Show Product Details *
* 4- Add product to cart (Purchase a product) *
* 5- Show cart details *
* 6- Logout from system *
*****
Please choose an option: 1
Showing product list...
1 Shoe
2 TV
3 Cellphone
4 Bag
5 new product
6 test product
7 test

*****
* 1- Show product list *
* 2- Select a product *
* 3- Show Product Details *
* 4- Add product to cart (Purchase a product) *
* 5- Show cart details *
* 6- Logout from system *
*****
Please choose an option: 2
Enter the number regarding to your product: 4
You have selected Bag and we have 2 of it
*****
* 1- Show product list *
* 2- Select a product *
* 3- Show Product Details *
* 4- Add product to cart (Purchase a product) *
* 5- Show cart details *
* 6- Logout from system *
*****
Please choose an option: 4
Enter a valid product index

4
Enter a valid product quantity

1
*****
* 1- Show product list *
* 2- Select a product *
* 3- Show Product Details *
* 4- Add product to cart (Purchase a product) *
* 5- Show cart details *
* 6- Logout from system *
*****
Please choose an option:

```

DISCUSSION

During this assignment, we investigated the concurrency in java RMI and also implemented three different functionalities including Browse, add products and purchase a product. Based on the concept behind purchase, browse and add products, and having multiple clients, we would like to have them in a synchronized way rather than concurrent way.

About the concurrency, we found that each client is separate than others and has its own threads.

CONCLUTION

REFERENCES

CONCLUTION

In conclusions, we found that java RMI has no guarantee in term of keeping our system thread safe, but we found that each client has its own thread pool and we just need to keep thread safe system for each client of RMI server.

REFERENCES

- <https://pdos.csail.mit.edu/6.824/papers/waldo-rmi.pdf>
- <https://pdfs.semanticscholar.org/9f05/b720ada20b6910fe2b1b20bd6650f93e8784.pdf>
- *Java and developers forums*
- *Lecture slides*