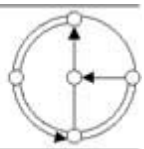
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich



SS 2005 Prof. Gustavo Alonso / Jianbo Xue

Vernetzte Systeme
Exercise 10

Ausgabe: 3. Juni 2005
Abgabe: 10. Juni 2005

1. Task: Implementation of Persistent Message Queue

This exercise is to implement a persistent message queue server and two clients. The server can receive/send messages from/to the clients. All the messages and queues are persistently stored at the server. In case the server crashes, it can recover by using the persistent storage and restore the queues and messages in the queues. Additionally, the server can create new queues and delete old queues according to the clients' requests.

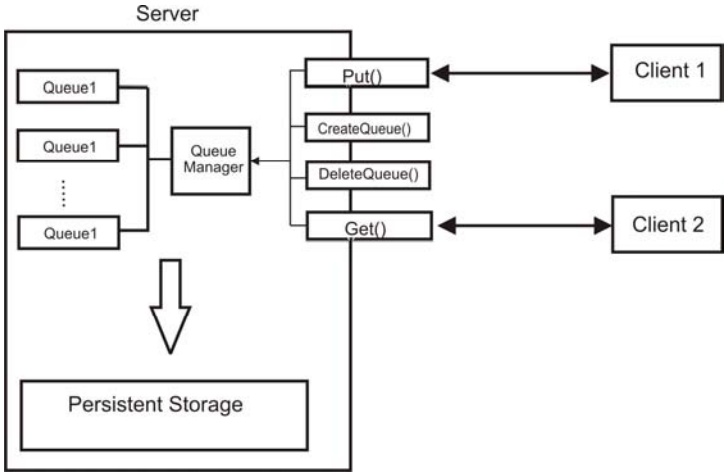


Figure 1: Message Server

- The server **`PersistMessagePoolServer`** implements four remote methods defined in the interface class **`PersistMessagePool`**:

```
package server;

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

```
public interface PersistMessagePool extends Remote{

	public void createQueue(String name) throws RemoteException;
	public void deleteQueue(String name) throws RemoteException;

	public void put(String msg, String queueName) throws
RemoteException;
	public String get(String queueName) throws RemoteException;

}
```

List 1. Interface **`PersistMessagePool`**

- **`put()`** method accepts a String message and a queue name from the client, finds the queue and stores the message into the queue in the server. In case the queue is not found, a **`QueueNotFoundException`** is thrown. In case the queue is full, **`put()`** operation will fail, and a **`QueueFullException`** will be thrown. In case the message from the client is null, the server will throw a **`MessageNullException`**.

- **`get()`** method retrieves the message from the queue specified by the name to the client which invokes it. In case the queue is not found, a **`QueueNotFoundException`** is thrown. The retrieved message will be deleted from the queue. In case the queue is empty, the **`get()`** operation will fail and a **`QueueEmptyException`** will be thrown.

- **`createQueue()`** method createa a queue with corresponding **`queueName`**. In case the queue already exists, a **`QueueDuplicateException`** is thrown. Otherwise **`true`** is return;

- **`deleteQueue()`** method deletes a queue with the corresponding **`queueName`**. In case the queue cannot be found, a **`QueueNotFoundException`** is thrown.

- The server maintains two-layer architecture for queue management: a queue manager **`PersistMessageQueueManager`** and a queue **`PersistMessageQueue.`**

  - **`PersistMessageQueueManager`** stores and manages a collection of queues and their states. Creating, deleting, and accessing to the queues have to be controlled by **`PersistMessageQueueManager.`**

  - **`PersistMessageQueue`** is a FIFO queue having a size of 100 messages. It implements **`enqueue()`** and **`dequeue()`**.

- Two clients **`MessageGetClient`** and **`MessagePutClient`** are to be implemented:
  - **`MessageGetClient`** calls the remote method **`get()`** to get message from a specific queue. For example, call **`get("queue1")`** to get message from queue **`queue1`**. The client retrieves messages periodically (1 message per 2 second); In case the specified queue cannot be found and the queue is empty, the client will wait and retry.

- **`MessagePutClient`** calls the remote method **`put()`** to send a message to a specific queue. The message could be the timestamp of the client or any random generated string. For example, call **`put("helloworld", "queue1")`** to send the message *helloworld* to the queue **`queue1`**. If the queue is not found, the client calls **`createQueue("queue1")`** to generate the queue first, and send the message. The client generates message periodically (1 message per second). In case the queue is full, the client will wait and retry.

- The following exceptions will be thrown as **`RemoteException`** in various situations. Make sure they are handled properly.

  - **`QueueFullException`** is thrown when the queue is full and there is a message to be put into the queue.
  - **`QueueEmptyException`** is thrown when the queue is empty and there is a retrieve operation
  - **`QueueDuplicateException`** is thrown when trying to create a queue with a name which already exists
  - **`QueueNotFoundException`** is thrown when trying to find or delete a queue which cannot be found
  - **`MessageNullException`** is thrown when trying to put an empty message into the queue.

- Persistency and fault tolerance: The server should be robust and fault tolerant. If the server crashes and restarts, it should be able to recover the previous state, including all the queues which have been generated, all the messages which have been stored in individual queue, and the states of the messages and queues. Once the server restarts, it serves the clients' requests as if the crash never happens.
  - The recovery mechanism of the server needs a persistent storage of all the data and states. Here is some hints:
  - For each queue, there is a queue file storing the data. The queue files are stored in a specific directory ***`queues`*** with an ending ***`.que`***. A queue has the same name as its queue file name. For example, ***`queue1`*** has its file as ***`queue1.que`***.
  - The queue manager checks the available queue files under the ***`queues`*** directory, and restores those queues.
  - Concerning efficiency, the queue operations (**`enqueue`** and **`dequeue`**) should be first done in main memory, and then stored on the file.

- For this exercise the program should be single threaded.