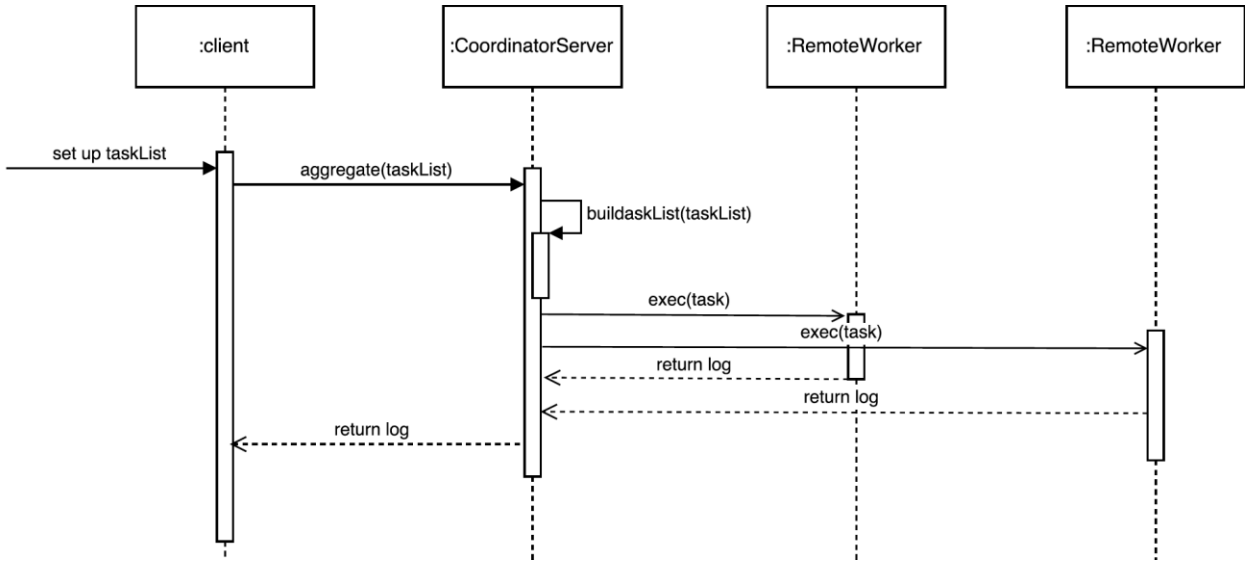
Design of My Solution

The project takes advantage of RMI to realize the remote task service. All tasks which need to send to a remote worker to execute implement the Task interface. Every task should be separated into several different subtask functions by itself. The data structure to store the sequence of these functions' names is a List<Set<String>>. The functions in Set can be executed in multi-thread by the remote worker and different Set in List should be executed in sequential way.

Making use of java reflection, the functions in a specific Task class can be restored in RemoteWorker by the functions' names in the List<Set<String>>.

The interactive diagram of the remote task service is as follows:



Key Design Decision

Task granularity:

1. Maximize parallelism:

— Dispatch different tasks into different RemoteWorkers (if the number of tasks do not exceed the number of worker), so that multiple tasks can execute in parallel.

— Invoke multiple threads in RemoteWorker to execute multiple independent subtasks at the same time.

2. Minimize contention:

— Because different Tasks execute in different RemoteWorkers, so every task can run in its own process. There is no shared resources among tasks, which decreases the contention among tasks.

Design pattern:

The thread pool design pattern: The Coordinator and every RemoteWorker have its own thread pool which maintains multiple threads waiting for tasks to be allocated for concurrent execution.

The Coordinator and every RemoteWorker also take advantage of Callable and Future interface to execute subtasks which have dependency with each other step by step. Then get the log result of all subtasks by future.get(), and return them to client.

Synchronize strategy

The task itself should recognize the race condition of subtasks in the worker. And it needs to synchronize the access to shared fields by themselves.