For this assignment, we are going to explore dart language and use it to solve a concurrency problem.

This video consists of three parts:

Firstly, we will explain the concurrency structure of dart, followed by some code snippet to show the features.

And then we will investigate the byzantine problem, and give our solution written in dart.

Finally, we will discuss the highlights and shortboards of the asynchronous structure.

Let’s get to know the concurrency design of Dart at first.

There are two main elements in Dart: Isolate and Event loop

Isolation is the special version of thread in dart. Each isolation is considered as a separate thread that doesn’t have any shared memory space with others. The only way of synchronising between two threads or to say isolates is by message passing. Dart use ports to transfer messages amongst isolates. Let me show you an example:

There are three function for this program:

Main, create isolate and doWork

In the main function, there are several print statements and also create isolate is called inside.

The second function is a simplest skeleton code for creating new threads in dart.

The way of generating a new thread is to call the function isolate.spawn.

And there are two arguments: The first one is the function that the thread is going to run, and the other one is the parameters passed to that function.

Also, different isolates need to communicate with each other. So we defined a receive port that receives messages from the corresponding send port. Run the listen function to process any incoming messages.

And inside doWork, we just print some random text and send a message to the given send port, which is the one created in the previous function.

Let’s run this program and see the output.

Isolate is just the multithreading solution to the concurrent applications. Dart also applied an asynchronous scheme within one thread which is based on event loop.

After synchronously executing all the codes, the dart virtual machine will start to empty the task queues. There two types of tasks: Micro task and event. As illustrated in this graph, microtask always have a higher priority over events. And dart will try to empty these two queues in a first in first out order. The program will exit once both of the queues become empty.

Let’s look at an example:

Dart uses schedule micro task to enqueue a micro task, and uses Future to create a new event. So here, according to the rule we discussed before, this line will be printed at first, followed by the microtask and event.

Let’s compile and see the result:

This is exactly the same as our expectation.

Also, in the newer version of dart, keywords async and await are introduced to simplify the coding style. You can simply use async when defining an asynchronous function and use await keyword to force the program to wait until a async command to be fulfilled before moving to the next line.

All in all, thanks to the support to both single thread and multithread async. dart has the ability to handle most concurrent tasks. And it helped us to implement king’s algorithm easily. Also, the unique design of isolates let us ignore the problems related to shared memory and helps us to get rid of thread locks on variables. And it also helped us to avoid lots race conditions.

However, if threads need to share multiple information, managing messages might become more complex. The best practice is to put the time-consuming simple tasks into isolates for a more fluent runtime experience.

# 引用作品

|  |  |
| --- | --- |
| [1] | J. ̊. Pohjola, “Lecture slide of Commitment and Consensus,” [Online]. Available: http://www.cse.unsw.edu.au/~cs3151/22T2/Week%2009/Wednesday%20Slides.pdf. |
| [2] | S. khan, “Concurrency In Dart,” 25 October 2021. [Online]. Available: https://medium.flutterdevs.com/concurrency-in-dart-b9171278af31. |
| [3] | Indian Statistical Institute, “Phase-King Algorithm,” [Online]. Available: https://www.isical.ac.in/~ansuman/dist\_sys/PhaseKing.pdf. |
| [4] | dart dev team, “Concurrency in Dart,” [Online]. Available: https://dart.dev/guides/language/concurrency. |
| [5] | Dart dev team, “The Event Loop and Dart,” [Online]. Available: https://dart.cn/articles/archive/event-loop. |