Dart与消息循环机制[翻译]

_番茄沙司 关注
3 2016.08.17 09:29:52 字数 2,729 阅读 3,926

dart

翻译自https://www.dartlang.org/articles/event-loop/

异步任务在Dart中随处可见，例如许多库的方法调用都会返回Future对象来实现异步处理，我们也可以注册Handler来响应一些事件，如：鼠标点击事件，I/O流结束和定时器到期。

这篇文章主要介绍了Dart中与异步任务相关的消息循环机制，阅读完这篇文章后相信你可写出更赞的异步执行代码。你也能学习到如何调度Future任务并且预测他们的执行顺序。

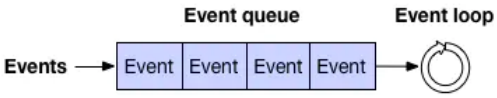
在阅读这篇文章之前，你最好先要了解一下基本的Future用法。

基本概念

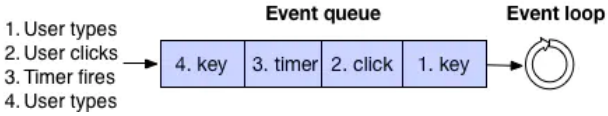
如果你写过一些关于UI的代码，你就应该熟悉消息循环和消息队列。有了他们才能保重UI的绘制操作和一些UI事件，如鼠标点击事件可以被一个一个的执行从而保证UI和UI事件的统一性。

消息循环和消息队列

一个消息循环的职责就是不断从消息队列中取出消息并处理他们直到消息队列为空。



消息队列中的消息可能来自用户输入，文件I/O消息，定时器等。例如下图的消息队列就包含了定时器消息和用户输入消息。



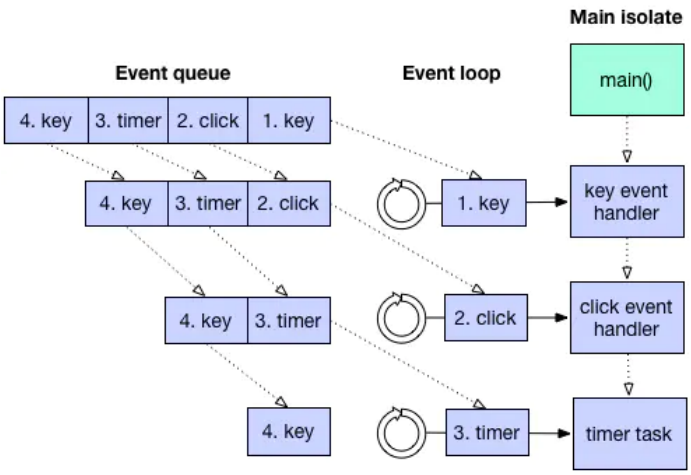
上述的这些概念你可能已经驾轻就熟了，那接下来我们就讨论一下这些概念在Dart中是怎么表现的?

Dart的单线程执行

当一个Dart的方法开始执行时，他会一直执行直至达到这个方法的退出点。换句话说Dart的方法是不会被其他Dart代码打断的。

> Note:一个Dart的命令行应用可以通过创建isolates来达到并行运行的目的。isolates之间不会共享内存，它们就像几个运行在不同进程中的app，中能通过传递message来进行交流。出了明确指出运行在额外的isolates或者workers中的代码外，所有的应用代码都是运行在应用的main isolate中。要了解更多相关内容，可以查看 https://www.dartlang.org/articles/event-loop/#use-isolates-or-workers-if-necessary

正如下图所示，当一个Dart应用开始的标志是它的main isolate执行了main方法。当main方法退出后，main isolate的线程就会去逐一处理消息队列中的消息。



事实上，上图是经过简化的流程。

Dart的消息循环和消息队列

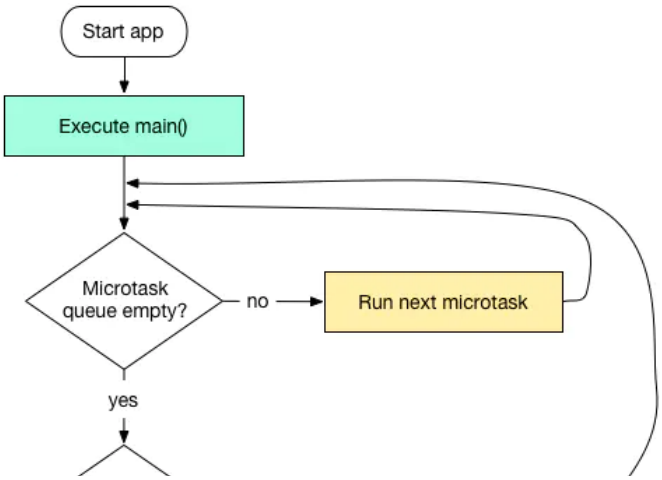
一个Dart应用有一个消息循环和两个消息队列-- *event队列*和*microtask队列*。

event队列包含所有外来的事件：I/O，mouse events，drawing events，timers，isolate之间的message等。

microtask 队列在Dart中是必要的，因为有时候事件处理想要在稍后完成一些任务但又希望是在执行下一个事件消息之前。

event队列包含Dart和来自系统其它位置的事件。但microtask队列只包含来自当前isolate的内部代码。

正如下面的流程图，当main方法退出后，event循环就开始它的工作。首先它会以FIFO的顺序执行micro task，当所有micro task执行完后它会从event 队列中取事件并执行。如此反复，直到两个队列都为空。



> **注意：**当事件循环正在处理micro task的时候。event队列会被堵塞。这时候app就无法进行UI绘制，响应鼠标事件和I/O等事件

虽然你可以预测任务执行的顺序，但你无法准确的预测到事件循环何时会处理你期望的任务。例如当你创建一个延时1s的任务，但在排在你之前的任务结束前事件循环是不会处理这个延时任务的，也就是或任务执行可能是大于1s的。

通过链接的方式指定任务顺序

如果你的代码之间存在依赖，那么尽量让他们之间的依赖关系明确一点。明确的依赖关系可以很好的帮助其他开发者理解你的代码，并且可以让你的代码更稳定也更容易重构。

先来看看下面这段错误代码：

```
// 这样写错误的原因就是没有明确体现出设置变量和使用变量之间的依赖关系
future.then(...set an important variable...);
Timer.run(() {...use the important variable...});
```

正确的写法应该是：

```
// 明确表现出了后者依赖前者设置的变量值
future.then(...set an important variable...)
  .then((_) {...use the important variable...});
```

为了表示明确的前后依赖关系，我们可以使用**then()()**来表明要使用变量就必须要等设置完这个变量。这里可以使用**whenComplete()**来代替then，它与then的不同点在于哪怕设置变量出现了异常也会被调用到。这个有点像java中的finally。

如果上面这个使用变量也要花费一段时间，那么可以考虑将其放入一个新的Future中：

```
future.then(...set an important variable...)
  .then((_) {new Future(() {...use the important variable...})});
```

使用一个新的Future可以给事件循环一个机会先去处理列队中的其他事件。

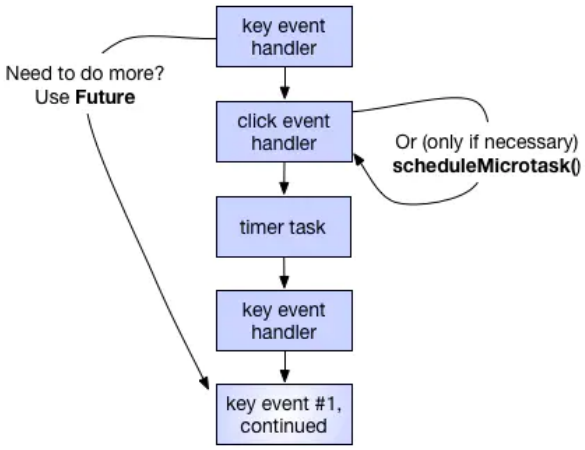
怎么安排一个任务

当你需要指定一些代码稍后运行的时候，你可以使用**dart:async**提供的两种方式：

```
1.Future类，它可以向event队列的尾部添加一个事件。
2.使用顶级方法**scheduleMicrotask()**，它可以向microtask队列的尾部添加一个微任务。
```

使用合理的队列

有可能的还是尽量使用Future来向event队列添加事件。使用event队列可以保持microtask队列的简短，以此减少microtask的过度使用导致event队列的堵塞。
如果一个任务确实要在event队列的任何一个事件前完成，那么你应该尽量直接写在main方法中而不是使用这两个队列。如果你不能那么就用scheduleMicrotask来向microtask添加一个微任务。



Event队列

使用 `new Future` 或者 `new Future.delayed()` 来向event队列中添加事件。

> **注意:**你也可以使用 `Timer` 来安排任务，但是使用Timer的过程中如果出现异常，则会退出程序。这里推荐使用Future，它是构建在Timer之上并加入了更多的功能，比如检测任务是否完成和异常反馈。

立刻需要将任务加入event队列可以使用new Future

```
//向event队列中添加一个任务
new Future(() {
  //任务具体代码
});
```

你也可以使用then或者whenComplete在Future结束后立刻执行某段代码。如下面这段代码在这个Future被执行后会立刻输出42：

```
new Future(() => 21)
    .then((v) => v*2)
    .then((v) => print(v));
```

如果要在一段时间后添加一个任务，可以使用new Future.delayed():

```
// 一秒以后将任务添加至event队列
new Future.delayed(const Duration(seconds:1), () {
  //任务具体代码
});
```

虽然上面这个例子中一秒后向event队列添加一个任务，但是这个任务想要被执行的话必须满足一下几点：

1. main方法执行完毕
2. microtask队列为空

3. 该任务前的任务全部执行完毕
   所以该任务真正被执行可能是大于1秒后。

关于Future的有趣事实：

1. 被添加到then()中的方法会在Future执行后立马执行(这方法没有被加入任何队列，只是被回调了)。
2. 如果在then()调用之前Future就已经执行完毕了，那么会有一个任务被加入到microtask队列中。这个任务执行的就是被传入then的方法。
3. Future()和Future.delayed()构造方法并不会被立刻完成，他们会向event队列中添加一个任务。
4. Future.value()构造方法会在一个microtask中完成。
5. Future,sync()构造方法会立马执行其参数方法，并在microtask中完成。

Microtask队列： scheduleMicrotask()

dart:async定义了一个顶级方法scheduleMicrotask() ，你可以这样使用：

```
scheduleMicrotask(() {
  // ...code goes here...
});
```

如果有必要可以使用isolate或worker

如果你想要完成一些重量级的任务，为了保证你应用可响应，你应该将任务添加到isolate或者worker中。isolate可能会运行在不同的进程或线程中.这取决于Dart的具体实现。

那一般情况下你应该使用多少个isolate来完成你的工作呢？通常情况下可以根据你的cpu的个数来决定。

但你也可以使用超过cpu个数的isolate，前提是你的app能有一个好的架构。让不同的isolate来分担不同的代码块运行，但这前提是你能保证这些isolate之间没有数据的共享。

测试一下你的理解程度

目前为止你已经掌握了调度任务的基本知识，下面来测试一下你的理解程度。

问题1

下面这段代码的输出是什么?

```
import 'dart:async';
main() {
  print('main #1 of 2');
  scheduleMicrotask(() => print('microtask #1 of 2'));

  new Future.delayed(new Duration(seconds:1),
                     () => print('future #1 (delayed)'));
  new Future(() => print('future #2 of 3'));
  new Future(() => print('future #3 of 3'));

  scheduleMicrotask(() => print('microtask #2 of 2'));

  print('main #2 of 2');
}
```

别急着看答案，自己在纸上写写答案呢?

答案：

```
main #1 of 2
main #2 of 2
microtask #1 of 2
microtask #2 of 2
future #2 of 3
future #3 of 3
future #1 (delayed)
```

上面的答案是否就是你所期望的呢？这段代码一共执行了三个分支：

1. main()方法
2. microtask队列
3. event队列（先new Future后new Future.delayed）

main方法中的普通代码都是同步执行的，所以肯定是main打印先全部打印出来，等main方法结束后会开始检查microtask中是否有任务，若有则执行，执行完继续检查microtask，直到microtask列队为空。所以接着打印的应该是microtask的打印。最后会去执行event队列。由于有一个使用的delay方法，所以它的打印应该是在最后的。

问题2

下面这个问题相对有些复杂：

```
import 'dart:async';
main() {
  print('main #1 of 2');
  scheduleMicrotask(() => print('microtask #1 of 3'));

  new Future.delayed(new Duration(seconds:1),
      () => print('future #1 (delayed)'));

  new Future(() => print('future #2 of 4'))
      .then((_) => print('future #2a'))
      .then((_) {
        print('future #2b');
        scheduleMicrotask(() => print('microtask #0 (from future #2b)'));
      })
      .then((_) => print('future #2c'));

  scheduleMicrotask(() => print('microtask #2 of 3'));

  new Future(() => print('future #3 of 4'))
      .then((_) => new Future(
                () => print('future #3a (a new future)')))
      .then((_) => print('future #3b'));

  new Future(() => print('future #4 of 4'));
  scheduleMicrotask(() => print('microtask #3 of 3'));
  print('main #2 of 2');
}
```

答案：

```
main #1 of 2
main #2 of 2
microtask #1 of 3
microtask #2 of 3
microtask #3 of 3
future #2 of 4
future #2a
future #2b
future #2c
microtask #0 (from future #2b)
future #3 of 4
future #4 of 4
future #3a (a new future)
future #3b
future #1 (delayed)
```

总结

以下有几点关于dart的事件循环机制需要牢记于心：

- Dart事件循环执行两个队列里的事件：event队列和microtask队列。
- event队列的事件来自dart（future，timer，isolate message等）和系统（用户输入，I/O等）。
- 目前为止，microtask队列的事件只来自dart。
- 事件循环会优先清空microtask队列，然后才会去处理event队列。
- 当两个队列都清空后，dart就会退出。
- main方法，来自event队列和microtask队列的所有事件都运行在Dart的main isolate中。

当你要安排一个任务时，请遵守以下规则：

- 如果可以，尽量将任务放入event队列中。
- 使用Future的then方法或whenComplete方法来指定任务顺序。
- 为了保持你app的可响应性，尽量不要将大计算量的任务放入这两个队列。
- 大计算量的任务放入额外的isolate中。

_番茄沙司 bilibili码农一枚
总资产206 共写了5.4W字 获得851个赞 共560个粉丝

关注