Flutter：使用 `Completer` 实现自定义任务队列

李小轰

flutter 实现自定义任务队列，先进先出，执行完一个任务再迭代下一个任务。

使用场景：

- 队列耗时任务执行

直接上代码:

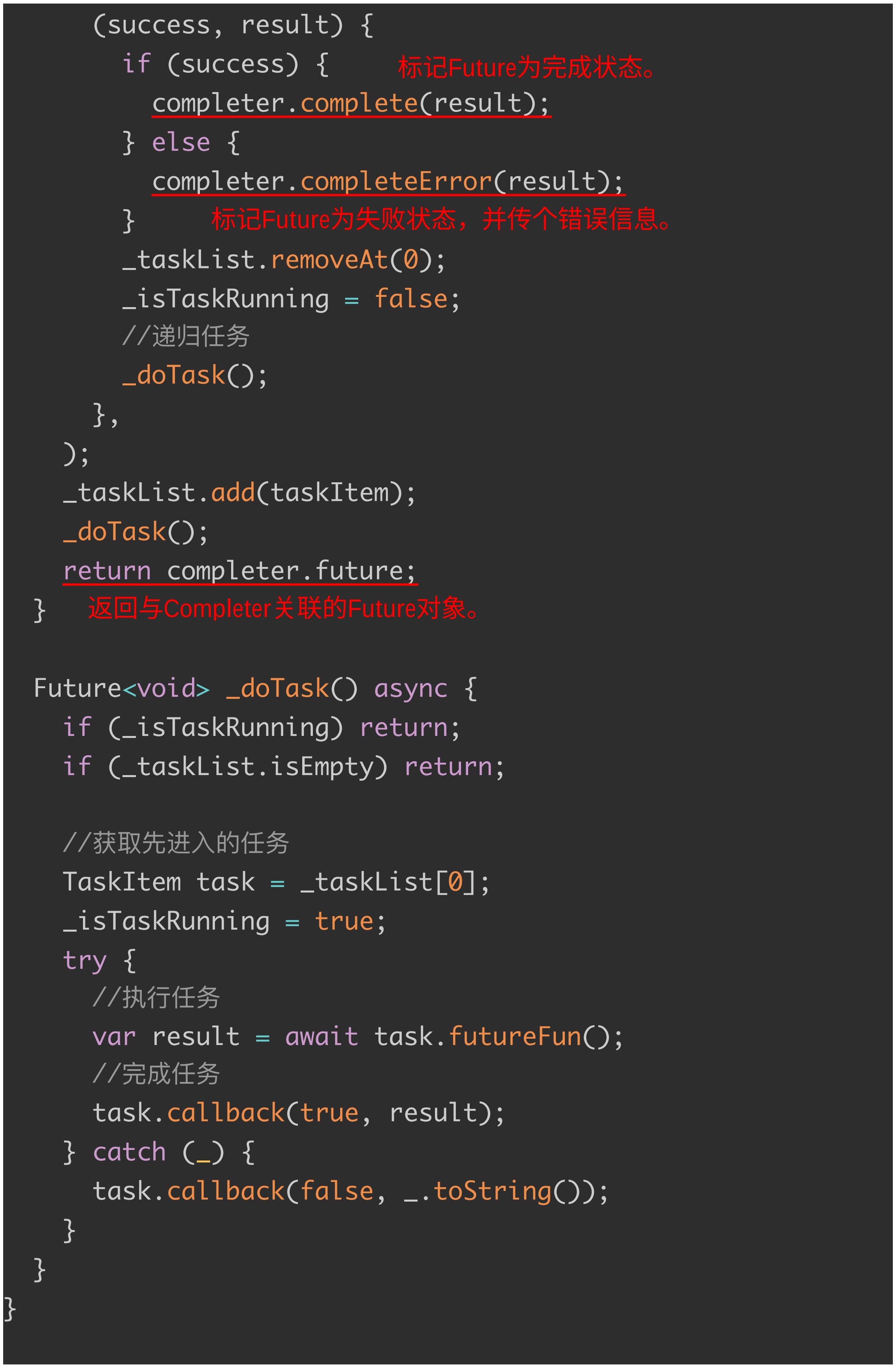
```
import 'dart:async';

typedef TaskCallback = void Function(bool success, dynamic
result);
typedef TaskFutureFuc = Future Function();

///队列任务，先进先出，一个个执行
class TaskQueueUtils {
  bool _isTaskRunning = false;
  List<TaskItem> _taskList = [];

  bool get isTaskRunning => _isTaskRunning;

  Future addTask(TaskFutureFuc futureFunc, {dynamic param})
{
    Completer completer = Completer();
    TaskItem taskItem = TaskItem(
      futureFunc,
```



```
      (success, result) {
        if (success) {     标记Future为完成状态。
          completer.complete(result);
        } else {
          completer.completeError(result);
        }     标记Future为失败状态，并传个错误信息。
        _taskList.removeAt(0);
        _isTaskRunning = false;
        //递归任务
        _doTask();
      },
    );
    _taskList.add(taskItem);
    _doTask();
    return completer.future;
  }   返回与Completer关联的Future对象。

  Future<void> _doTask() async {
    if (_isTaskRunning) return;
    if (_taskList.isEmpty) return;

    //获取先进入的任务
    TaskItem task = _taskList[0];
    _isTaskRunning = true;
    try {
      //执行任务
      var result = await task.futureFun();
      //完成任务
      task.callback(true, result);
    } catch (_) {
      task.callback(false, _.toString());
    }
  }
}
```

```
///任务封装
class TaskItem {
  final TaskFutureFuc futureFun;
  final TaskCallback callback;

  const TaskItem(
    this.futureFun,
    this.callback,
  );
}
```

使用方式：

```
main() {
  Future task1() {
    return Future(() async {
      print("start task1");
      await Future.delayed(Duration(seconds: 3));
      return "end task1";
    });
  }

  Future task2() {
    return Future(() async {
      print("start task2");
      await Future.delayed(Duration(seconds: 1));
      return "end task2";
    });
  }

  TaskQueueUtils queueUtils = TaskQueueUtils();
  queueUtils.addTask(task1).then((result) {
    print(result);
    return Future.value(result);
  });
  queueUtils.addTask(task2).then((result) {
    print(result);
    return Future.value(result);
  });
}
```

task1，task2 为模拟的耗时任务。

打印结果如下：

```
start task1
end task1
start task2
end task2
```