Future

1. 功能概述

Future是一个方便通过异步的方式执行耗时工作的类，能够很高的提升程序的并发性。

在进行耗时工作时，通过立即返回一个Future对象，后台实际执行耗时任务，最后通过返回的Future对象获取异步任务的执行结果。

Future Pattern包括三个组成部分：Future、Core、Promise（为Core的封装类）。

1. 功能细分
2. 耗时接口立即返回Future对象，耗时操作在线程里面执行。后续通过返回的Future对象获取异步耗时操作的结果，通过Promise辅助。
3. then(callbackfunc)函数，该函数在Future对象代表的异步操作完成的时候然后立即执行then里面的callbackfunc函数（在主线程内执行），这种方式下Future通过关联的Promise对象通知操作是否完成。并且此函数返回值为Future对象，所以可以链式调用then函数。比如Future.then(a).then(b).then(c)。
4. collectAll和CollectAny两个函数都是用来等待Future集合的，分别用来等待Future集合内所有的Future对象完成和等待Future集合内任意一个Future完成。
5. via函数，这个函数用来控制Future中callback函数能够在制定线程内执行，通过传递一个Executor对象返回一个Future对象，然后在调用then函数，此时then里面设置的callback函数就会在Executor内执行。比如Future.via(executor1).then(a).via(executor2).then(b)，此时a和b分别在executor1和executor2线程内执行，而不像默认的then函数一样在主线程内执行。
6. onError函数，用来处理Future内抛出的任何异常，并且支持chaining-together，就像一个try后面跟多个catch一样（这通常是不必要的）
7. ensure函数，这个函数类似于java里面的finally块。无论Future的执行结果是包含一个期望值（value）还是一个异常（exception），都会保证ensure内的callback函数一定会被执行。
8. map（futures,mapFunction）函数，这个函数是Future内的higher-order-function map（类似于MapReduce内的map函数），用来对一组输入集通过指定一个functin，从而映射到另外一组对应的结果集（按照输入集的顺序依次调用这个指定的function映射函数）。
9. reduce(futures, initialvalue, reduceFunction(reduced value, next result from the futures))函数，这个函数是Future内的higher-order-function fold（类似于MapReduce内的reduce函数），用来对经过map处理过的结果集进行规约处理（也是按照结果集的顺序进行），规约函数通过reduce内的callback函数指定。用于对结果集中的每个结果和规约的结果进行规约处理。最后返回对结果集进行规约后的结果。

For instance，if you have a collection of Future<A> and you want to have Future<bool> that container if all the input ints are equal to zero, You might write:

|  |
| --- |
| reduce(fs, true, [](bool b, int i){  // You could also return a Future<bool> if you needed to  return b && (i == 0);  })  .then([](bool result){  // result is true if all inputs were zero  });  // You could use onError or Try here in case one of your input Futures  // contained an exception or if your reducing function threw an exception |

1. Executor implementation包括

（1）ThreadPoolExecutor(CPUThreadPoolExecutor,IOThreadPoolExecutor)，

（2）ManualExecutor

（3）InlineExecutor

（4）QueueImmediateExecutor

（5）ScheduledExecutor

（6）ThreadManager

（7）FutureExecutor

1. TimeOuts
2. 实现

首先应该对Future、Core、Promise三者的职责定位明确。

1. Future：具有get方法获取Core里面的数据，实现线程同步。主要关注于线程同步，对Core内的数据进行相关处理（map、reduce等）。
2. Core：实际的共享数据保存在Core里面，对外提供setValue和getValue的方法，isReady等方法，不具备线程同步的功能，主要关注于数据存取方面。
3. Promise：为对Core的封装，并关联到Future。对外提供统一入口（通过创建Promise对象可以获取到一个和它关联的Future对象）。

Core类设计:

Core存储实际数据，其存储数据类型通过模板指定。

类成员：

结果存储：result\_,

状态：FSM fsm\_

线程：excutor\_,

回调：callback\_,

锁控制线程安全：lock,

活动状态标记变量：active\_

方法：

1. setResult()
2. getResult()
3. setCallback()
4. doCallback()
5. .......
6. 功能细分第一点

Future和Promise需要共享一个代表操作结果的变量，当异步操作执行完之后，

Promise通过设置该变量使得Future得知异步操作完成。伪代码如下：

|  |
| --- |
| …….  int main()  {  Future<T> fut = expensiveProcessFunc(…);.  //do something else rather than wait future to complete  T result = fut.get();  //……..  }  Future<T> expensiveProcessFunc(…)  {  Promise<T> promise;  Future<T> fut = promise.getFuture();  Callback cb(promise);  asyncProcessFunc(cb);  return fut;  }  template <typename Promise>  Class Callback  {  Public:  Callback(Promise promise) : promise\_(promise) {….}  Void callback()  {  promise\_.setValue();  }  Private:  Promise promise\_;  } |

Git地址：

https://github.com/facebook/folly/tree/master/folly/futures