并发编程 Promise, Future 和 Callback

2014/06/22 23:23 - Posted By 起衣 - Tags: callback,java,未来 - Category: Code

1 条评论

在并发编程中，我们通常会用到一组非阻塞的模型：Promise，Future 和 Callback。其中的 Future 表示一个可能还没有实际完成的异步任务的结果，针对这个结果可以添加 Callback 以便在任务执行成功或失败后做出对应的操作，而 Promise 交由任务执行者，任务执行者通过 Promise 可以标记任务完成或者失败。 可以说这一套模型是很多异步非阻塞架构的基础。

这一套经典的模型在 Scala、C# 中得到了原生的支持，但 JDK 中暂时还只有无 Callback 的 Future 出现，当然也并非在 JAVA 界就没有发展了，比如 Guava 就提供了 ListenableFuture 接口，而 Netty 4+ 更是提供了完整的 Promise、Future 和 Listener 机制，在 Netty 的官方文档 Using as a generic library 中也介绍了将 Netty 作为一个 lib 包依赖，并且使用 Listenable futures 的示例。在实际的项目使用中，发现 Netty 的 EventLoop 机制不一定适用其他场景，因此想去除对 EventLoop 的依赖，实现一个简化版本。

参考 Scala 和 Netty 的代码重新定义了接口和实现，先介绍下和 Netty 版本的区别:

1. 去除了对 EventLoop 的依赖，Callback 的执行策略不同：任务未完成时添加的 Callback，会在结束任务的线程执行；任务完成后添加的 Callback 会在添加 Callback 线程立即执行
2. 一个 Callback 执行后会立即被清理
3. Callback 可以根据任务结果添加，支持添加以下三种 Callback: onComplete, onSuccess, onFailure, 不需要和 Netty 的 FutureListener 一样大部分场景下都需要检查 future.isSuccess 等
4. 支持 Callback 的组合，Callback 包含一些函数式的方法，比如 compose 和 andThen 可以用来组合
5. 使用 CountdownLatch 替换掉了 Netty 的 wait/notify 实现
6. 去掉 Netty Future 一些不常使用的方法，同时补充一些模型间关联的方法，比如 Promise.getFuture

然后再介绍几个使用这个 commons-future 的示例:

1. 异步执行任务，获得 Future 后添加 Callback

```
01.  final TaskPromise promise = new DefaultTaskPromise();
02.  final TaskFuture future = promise.getFuture();
03.  final CountDownLatch latch = new CountDownLatch(1);
04.  future.onComplete(new TaskCallback() { // 添加结束 Callback
```

```
    @Override
    public TaskFuture apply(TaskFuture f) {
        latch.countDown();
        return f;
    }
});
new Thread(new Runnable() {
    @Override
    public void run() {
        promise.setSuccess(null);
    }
}).start();
latch.await();
```

2. 异步执行任务，获得 Future 后添加成功结束的 Callback

```
final TaskPromise promise = new DefaultTaskPromise();
final TaskFuture future = promise.getFuture();
final CountDownLatch latch = new CountDownLatch(1);
future.onSuccess(new TaskCallback() { // 添加成功结束 Callback
    @Override
    public TaskFuture apply(TaskFuture f) {
        latch.countDown();
        return f;
    }
});
new Thread(new Runnable() {
    @Override
    public void run() {
        promise.setSuccess(null);
    }
}).start();
```

```
17.  latch.await();
```

3. 异步执行任务，获得 Future 后，添加失败结束的组合 Callback

```
01.  final TaskPromise promise = new DefaultTaskPromise();
02.  final TaskFuture future = promise.getFuture();
03.  final CountDownLatch latch = new CountDownLatch(2);
04.  future.onFailure(new TaskCallback() {
05.      @Override
06.      public TaskFuture apply(TaskFuture f) {
07.          latch.countDown();
08.          return f;
09.      }
10.  }.andThen(new TaskCallback() {
11.      @Override
12.      public TaskFuture apply(TaskFuture f2) {
13.          latch.countDown();
14.          return f2;
15.      }
16.  }));
17.  new Thread(new Runnable() {
18.      @Override
19.      public void run() {
20.          promise.setFailure(new IllegalStateException("cm"));
21.      }
22.  }).start();
23.  latch.await();
```

4. 异步执行任务，获得 Future 后阻塞等待任务完成

```
01.  final TaskPromise promise = new DefaultTaskPromise();
02.  final TaskFuture future = promise.getFuture();
```

```
new Thread(new Runnable() {
    @Override
    public void run() {
        try {
            TimeUnit.SECONDS.sleep(2);
        } catch (InterruptedException e) {
        }
        promise.setFailure(new IllegalStateException("cm"));
    }
}).start();
future.await();
```

代码仓库: https://bitbucket.org/qiyi/commons-future
参考:

- http://docs.scala-lang.org/sips/completed/futures-promises.html
- http://scala-lang.org/
- https://github.com/netty/netty
- http://docs.oracle.com/javase/6/docs/api/java/util/concurrent/CountDownLatch.html
- http://biasedbit.com/countdownlatch-vs-wait-notify/

相关日志 Relate Posts

- 根据条件过滤日志输出
- SpringBoot应用集成etcd配置源
- SpringBoot应用配置项加密
- retrofit 服务自定义签名认证
- retrofit 发起form data请求时使用自定义对象参数

“并发编程 Promise, Future 和 Callback”1条留言

雪鹰领主: September 17, 2015 at 4:53 pm
[围观]。。。。。

Reply

发表留言（Ctrl+Enter提交）

昵称 (必填)

EMail (不公开，必填)

网站 (选填)

发表