3

第 3 篇

Netty 核心篇

9

第 9 章

Promise 与 Future 双子星的秘密

课程目标

1、深入了解 Netty 的运行机制。

2、掌握 NioEventLoop、Pipeline、ByteBuf 的核心原理。

3、掌握 Netty 常见的调优方案。

内容定位

1、希望深入了解 Netty 源码的人群。

2、未来可能参与中间件开发的人群。

9.1 异步结果 Future

java.util.concurrent.Future 是 Java 提供的接口，表示异步执行的状态，Future 的 get 方法会判断任务是否执行完成，如果完成就返回结果，否则阻塞线程，直到任务完成。

Netty 扩展了 Java 的 Future，最主要的改进就是增加了监听器 Listener 接口，通过监听器可以让异步执行更加有效率，不需要通过 get 来等待异步执行结束，而是通过监听器回调来精确地控制异步执行结束的时间点。

```
public interface Future<V> extends java.util.concurrent.Future<V> {
    boolean isSuccess();
    boolean isCancellable();
    Throwable cause();
    Future<V> addListener(GenericFutureListener<? extends Future<? super V>> listener);
    Future<V> addListeners(GenericFutureListener<? extends Future<? super V>>... listeners);
    Future<V> removeListener(GenericFutureListener<? extends Future<? super V>> listener);
    Future<V> removeListeners(GenericFutureListener<? extends Future<? super V>>... listeners);
    Future<V> sync() throws InterruptedException;

    Future<V> syncUninterruptibly();
    Future<V> await() throws InterruptedException;
    Future<V> awaitUninterruptibly();
    boolean await(long timeout, TimeUnit unit) throws InterruptedException;
    boolean await(long timeoutMillis) throws InterruptedException;
    boolean awaitUninterruptibly(long timeout, TimeUnit unit);
    boolean awaitUninterruptibly(long timeoutMillis);
    V getNow();
    boolean cancel(boolean mayInterruptIfRunning);
}
```

ChannelFuture 接口扩展了 Netty 的 Future 接口，表示一种没有返回值的异步调用，同时和一个 Channel 进行绑定。

```
public interface ChannelFuture extends Future<Void> {
    Channel channel();
    ChannelFuture addListener(GenericFutureListener<? extends Future<? super Void>> listener);
    ChannelFuture addListeners(GenericFutureListener<? extends Future<? super Void>>... listeners);
    ChannelFuture removeListener(GenericFutureListener<? extends Future<? super Void>> listener);
    ChannelFuture removeListeners(GenericFutureListener<? extends Future<? super Void>>... listeners);
    ChannelFuture sync() throws InterruptedException;
    ChannelFuture syncUninterruptibly();
    ChannelFuture await() throws InterruptedException;
    ChannelFuture awaitUninterruptibly();
    boolean isVoid();
}
```

9.2 异步执行 Promise

Promise 接口也扩展了 Future 接口，它表示一种可写的 Future，就是可以设置异步执行的结果。

```
public interface Promise<V> extends Future<V> {
    Promise<V> setSuccess(V result);
    boolean trySuccess(V result);
    Promise<V> setFailure(Throwable cause);
```

```
    boolean tryFailure(Throwable cause);
    boolean setUncancellable();
    Promise<V> addListener(GenericFutureListener<? extends Future<? super V>> listener);
    Promise<V> addListeners(GenericFutureListener<? extends Future<? super V>>... listeners);
    Promise<V> removeListener(GenericFutureListener<? extends Future<? super V>> listener);
    Promise<V> removeListeners(GenericFutureListener<? extends Future<? super V>>... listeners);

    Promise<V> await() throws InterruptedException;
    Promise<V> awaitUninterruptibly();
    Promise<V> sync() throws InterruptedException;
    Promise<V> syncUninterruptibly();
}
```

ChannelPromise 接口扩展了 Promise 和 ChannelFuture，绑定了 Channel，既可写异步执行结构，又具备了监听者的功能，是 Netty 实际编程使用的表示异步执行的接口。

```
public interface ChannelPromise extends ChannelFuture, Promise<Void> {
    Channel channel();
    ChannelPromise setSuccess(Void result);
    ChannelPromise setSuccess();
    boolean trySuccess();
    ChannelPromise setFailure(Throwable cause);
    ChannelPromise addListener(GenericFutureListener<? extends Future<? super Void>> listener);
    ChannelPromise addListeners(GenericFutureListener<? extends Future<? super Void>>... listeners);
    ChannelPromise removeListener(GenericFutureListener<? extends Future<? super Void>> listener);
    ChannelPromise removeListeners(GenericFutureListener<? extends Future<? super Void>>... listeners);
    ChannelPromise sync() throws InterruptedException;
    ChannelPromise syncUninterruptibly();
    ChannelPromise await() throws InterruptedException;
    ChannelPromise awaitUninterruptibly();
    ChannelPromise unvoid();
}
```

DefaultChannelPromise 是 ChannelPromise 的实现类，它是实际运行时的 Promoise 实例。Netty 使用 addListener() 方法来回调异步执行的结果。看一下 DefaultPromise 的 addListener()方法的源码：

```
public Promise<V> addListener(GenericFutureListener<? extends Future<? super V>> listener) {
        checkNotNull(listener, "listener");

        synchronized (this) {
            addListener0(listener);
        }

        if (isDone()) {
            notifyListeners();
        }

        return this;
}

private void addListener0(GenericFutureListener<? extends Future<? super V>> listener) {
        if (listeners == null) {
            listeners = listener;
        } else if (listeners instanceof DefaultFutureListeners) {
            ((DefaultFutureListeners) listeners).add(listener);
        } else {
            listeners = new DefaultFutureListeners((GenericFutureListener<? extends Future<V>>) listeners, listener);
        }
}
```

```
private void notifyListeners() {
        EventExecutor executor = executor();
        if (executor.inEventLoop()) {
            final InternalThreadLocalMap threadLocals = InternalThreadLocalMap.get();
            final int stackDepth = threadLocals.futureListenerStackDepth();
            if (stackDepth < MAX_LISTENER_STACK_DEPTH) {
                threadLocals.setFutureListenerStackDepth(stackDepth + 1);
                try {
                    notifyListenersNow();
                } finally {
                    threadLocals.setFutureListenerStackDepth(stackDepth);
                }
                return;
            }
        }

        safeExecute(executor, new Runnable() {
            @Override
            public void run() {
                notifyListenersNow();
            }
        });
}
```

它会判断异步任务执行的状态，如果执行完成，就理解通知监听者，否则加入到监听者队列通知监听者就是找一个线程来执行调用监听的回调函数。再来看监听者的接口，就一个方法，即等异步任务执行完成后，拿到 Future 结果，执行回调的逻辑。

```
public interface GenericFutureListener<F extends Future<?>> extends EventListener {
    void operationComplete(F future) throws Exception;
}
```