# Executor Future FutureTask

## Future RunnableFuture FutureTask

**Future是一个接口，用来取异步计算的结果V，可以终止执行**

public interface **Future<V>** {

boolean cancel(boolean mayInterruptIfRunning);

boolean isDone();

V get() throws InterruptedException, ExecutionException;

V get(long timeout, TimeUnit unit)

throws InterruptedException, ExecutionException, TimeoutException;

}

**RunnableFuture也是接口，继承了Runnable和Future接口**

public interface **RunnableFuture<V>** extends Runnable, Future<V> {

void run();

}

**FutureTask是实现类，实现了Future和Runnable接口中的方法**

public class **FutureTask<V>** implements RunnableFuture<V> {

private Callable<V> callable;

private volatile Thread runner;

// 使用Runnable构造

public FutureTask(Runnable runnable, V result) {

this.callable = Executors.callable(runnable, result);

this.state = NEW; // ensure visibility of callable

}

// 使用Callable构造

public FutureTask(Callable<V> callable) {

if (callable == null)

throw new NullPointerException();

this.callable = callable;

this.state = NEW; // ensure visibility of callable

}

// 执行Thread的interruput

public boolean cancel(boolean mayInterruptIfRunning) {

Thread t = runner;

t.interrupt();

}

// 执行Callable的call，并得到结果

public void run() {

Callable<V> c = callable;

result = c.call();

}

}

## Executor ExecutorService AbstractExecutorService ThreadPoolExecutor

Executor可以直接调用execute执行一个runnable

ExecutorService可以调用submit执行，返回一个Future，用来取执行的结果

**Executor是一个接口，包含唯一的execute方法**

public interface Executor {

void execute(Runnable command);

}

**ExecutorService也是接口，新增了shutdown，submit**

public interface ExecutorService extends Executor {

void shutdown();

List<Runnable> shutdownNow();

boolean isShutdown();

boolean isTerminated();

<T> Future<T> submit(Callable<T> task);

<T> Future<T> submit(Runnable task, T result);

Future<?> submit(Runnable task);

invokeAll(…)

…

}

**AbstractExecutorService是抽象的类，其功能是把submit的Runnable转换成FutureTask后执行**

public abstract class **AbstractExecutorService** implements ExecutorService {

public <T> Future<T> submit(Runnable task, T result) {

if (task == null) throw new NullPointerException();

RunnableFuture<T> ftask = new FutureTask<T>( task, result);

execute(ftask);

return ftask;

}

}

**ThreadPoolExecutor是具体实现类，使用线程池实现了execute和shutdown等**

public class **ThreadPoolExecutor** extends AbstractExecutorService {

public void execute(Runnable command) {

}

public void shutdown() {

}

}

// 使用示例

mWorkQueue = new PriorityBlockingQueue<Runnable>(Const.WORK\_QUEUE\_MAX\_COUNT);

mExecutor = new ThreadPoolExecutor(Const.THREAD\_DEF\_WORDER\_COUNT,

Const.THREAD\_MAX\_WORKER\_COUNT, Const.THREAD\_KEEP\_ALIVE\_TIME,

TimeUnit.SECONDS, mWorkQueue, new ThreadFactory() {

private final AtomicInteger mCount = new AtomicInteger(1);

public Thread newThread(Runnable r) {

return new Thread(r, "HttpHandlerImpl ConnectTask #"

+ mCount.getAndIncrement());

}

}, new RejectedExecutionHandler() {

public void rejectedExecution(Runnable r, ThreadPoolExecutor e) {

if (!e.isShutdown()) {

Runnable runnable = e.getQueue().poll();

e.execute(r);

}

}

}); // 或者new ThreadPoolExecutor.DiscardOldestPolicy()

# Syncronized ReentrantLock

# LinkedHashMap Iterator

next

remove