# [Fork/Join框架详解](https://www.cnblogs.com/senlinyang/p/7885964.html)

　　Fork/Join框架是Java 7提供的一个用于并行执行任务的框架，是一个把大任务分割成若干个小任务，最终汇总每个小任务结果后得到大任务结果的框架。Fork/Join框架要完成两件事情：

　　1.任务分割：首先Fork/Join框架需要把大的任务分割成足够小的子任务，如果子任务比较大的话还要对子任务进行继续分割

　　2.执行任务并合并结果：分割的子任务分别放到双端队列里，然后几个启动线程分别从双端队列里获取任务执行。子任务执行完的结果都放在另外一个队列里，启动一个线程从队列里取数据，然后合并这些数据。

　　在Java的Fork/Join框架中，使用两个类完成上述操作

　　1.ForkJoinTask:我们要使用Fork/Join框架，首先需要创建一个ForkJoin任务。该类提供了在任务中执行fork和join的机制。通常情况下我们不需要直接集成ForkJoinTask类，只需要继承它的子类，Fork/Join框架提供了两个子类：

　　　　a.RecursiveAction：用于没有返回结果的任务

　　　　b.RecursiveTask:用于有返回结果的任务

　　2.ForkJoinPool:ForkJoinTask需要通过ForkJoinPool来执行

　　任务分割出的子任务会添加到当前工作线程所维护的双端队列中，进入队列的头部。当一个工作线程的队列里暂时没有任务时，它会随机从其他工作线程的队列的尾部获取一个任务(工作窃取算法)。

Fork/Join框架的实现原理

　　ForkJoinPool由ForkJoinTask数组和ForkJoinWorkerThread数组组成，ForkJoinTask数组负责将存放程序提交给ForkJoinPool，而ForkJoinWorkerThread负责执行这些任务。

　　ForkJoinTask的Fork方法的实现原理：  
　　当我们调用ForkJoinTask的fork方法时，程序会把任务放在ForkJoinWorkerThread的pushTask的**workQueue**中，异步地执行这个任务，然后立即返回结果，代码如下：

[IMG_256](https://www.cnblogs.com/senlinyang/p/javascript:void(0);)

public final ForkJoinTask<V> fork() {

Thread t;

if ((t = Thread.currentThread()) instanceof ForkJoinWorkerThread)

((ForkJoinWorkerThread)t).workQueue.push(this);

else

ForkJoinPool.common.externalPush(this);

return this;

}

[IMG_257](https://www.cnblogs.com/senlinyang/p/javascript:void(0);)

　　pushTask方法把当前任务存放在ForkJoinTask数组队列里。然后再调用ForkJoinPool的signalWork()方法唤醒或创建一个工作线程来执行任务。代码如下：

[IMG_258](https://www.cnblogs.com/senlinyang/p/javascript:void(0);)

final void push(ForkJoinTask<?> task) {

ForkJoinTask<?>[] a; ForkJoinPool p;

int b = base, s = top, n;

if ((a = array) != null) { // ignore if queue removed

int m = a.length - 1; // fenced write for task visibility

U.putOrderedObject(a, ((m & s) << ASHIFT) + ABASE, task);

U.putOrderedInt(this, QTOP, s + 1);

if ((n = s - b) <= 1) {

if ((p = pool) != null)

p.signalWork(p.workQueues, this);

}

else if (n >= m)

growArray();

}

}

[IMG_259](https://www.cnblogs.com/senlinyang/p/javascript:void(0);)

**ForkJoinTask的join方法实现原理**

　　Join方法的主要作用是阻塞当前线程并等待获取结果。让我们一起看看ForkJoinTask的join方法的实现，代码如下：

[IMG_260](https://www.cnblogs.com/senlinyang/p/javascript:void(0);)

public final V join() {

int s;

if ((s = doJoin() & DONE\_MASK) != NORMAL)

reportException(s);

return getRawResult();

}

[IMG_261](https://www.cnblogs.com/senlinyang/p/javascript:void(0);)

　　它首先调用doJoin方法，通过doJoin()方法得到当前任务的状态来判断返回什么结果，任务状态有4种：已完成（NORMAL）、被取消（CANCELLED）、信号（SIGNAL）和出现异常（EXCEPTIONAL）。

　　如果任务状态是已完成，则直接返回任务结果。

　　如果任务状态是被取消，则直接抛出CancellationException

　　如果任务状态是抛出异常，则直接抛出对应的异常

　　让我们分析一下doJoin方法的实现

[IMG_262](https://www.cnblogs.com/senlinyang/p/javascript:void(0);)

private int doJoin() {

int s; Thread t; ForkJoinWorkerThread wt; ForkJoinPool.WorkQueue w;

return (s = status) < 0 ? s :

((t = Thread.currentThread()) instanceof ForkJoinWorkerThread) ?

(w = (wt = (ForkJoinWorkerThread)t).workQueue).

tryUnpush(this) && (s = doExec()) < 0 ? s :

wt.pool.awaitJoin(w, this, 0L) :

externalAwaitDone();

}

[IMG_263](https://www.cnblogs.com/senlinyang/p/javascript:void(0);)

[IMG_264](https://www.cnblogs.com/senlinyang/p/javascript:void(0);)

final int doExec() {

int s; boolean completed;

if ((s = status) >= 0) {

try {

completed = exec();

} catch (Throwable rex) {

return setExceptionalCompletion(rex);

}

if (completed)

s = setCompletion(NORMAL);

}

return s;

}

[IMG_265](https://www.cnblogs.com/senlinyang/p/javascript:void(0);)

　　在doJoin()方法里，首先通过查看任务的状态，看任务是否已经执行完成，如果执行完成，则直接返回任务状态；如果没有执行完，则从任务数组里取出任务并执行。如果任务顺利执行完成，则设置任务状态为NORMAL，如果出现异常，则记录异常，并将任务状态设置为EXCEPTIONAL。

**Fork/Join框架的异常处理**

　　ForkJoinTask在执行的时候可能会抛出异常，但是我们没办法在主线程里直接捕获异常，所以ForkJoinTask提供了isCompletedAbnormally()方法来检查任务是否已经抛出异常或已经被取消了，并且可以通过ForkJoinTask的getException方法获取异常。使用如下代码：

if(task.isCompletedAbnormally())

{

System.out.println(task.getException());

}

　　getException方法返回Throwable对象，如果任务被取消了则返回CancellationException。如果任务没有完成或者没有抛出异常则返回null。

[IMG_266](https://www.cnblogs.com/senlinyang/p/javascript:void(0);)

public final Throwable getException() {

int s = status & DONE\_MASK;

return ((s >= NORMAL) ? null :

(s == CANCELLED) ? new CancellationException() :

getThrowableException());

}

[IMG_267](https://www.cnblogs.com/senlinyang/p/javascript:void(0);)

**使用例子：**  
　　我们通过一个简单的例子来介绍一下Fork/Join框架的使用。需求是求1+2+3+4的结果

　　使用Fork/Join框架首先要考虑到的是如何分割任务，如果希望每个子任务最多执行两个数的相加，那么我们设置分割的阈值是2，由于是4个数字相加，所以Fork/Join框架会把这个任务fork成两个子任务，子任务一负责计算1+2，子任务二负责计算3+4，然后再join两个子任务的结果。因为是有结果的任务，所以必须继承RecursiveTask，实现代码如下：

[IMG_268](https://www.cnblogs.com/senlinyang/p/javascript:void(0);)

public class CountTask extends RecursiveTask<Integer>{

private static final int THREAD\_HOLD = 2;

private int start;

private int end;

public CountTask(int start,int end){

this.start = start;

this.end = end;

}

@Override

protected Integer compute() {

int sum = 0;

//如果任务足够小就计算

boolean canCompute = (end - start) <= THREAD\_HOLD;

if(canCompute){

for(int i=start;i<=end;i++){

sum += i;

}

}else{

int middle = (start + end) / 2;

CountTask left = new CountTask(start,middle);

CountTask right = new CountTask(middle+1,end);

//执行子任务 left.fork();

right.fork();

//获取子任务结果

int lResult = left.join();

int rResult = right.join();

sum = lResult + rResult;

}

return sum;

}

public static void main(String[] args){

ForkJoinPool pool = new ForkJoinPool();

CountTask task = new CountTask(1,4);

Future<Integer> result = pool.submit(task);

try {

System.out.println(result.get());

} catch (InterruptedException e) {

e.printStackTrace();

} catch (ExecutionException e) {

e.printStackTrace();

}

}

}

[IMG_269](https://www.cnblogs.com/senlinyang/p/javascript:void(0);)

　　通过这个例子，我们进一步了解ForkJoinTask，ForkJoinTask与一般任务的主要区别在于它需要实现compute方法，在这个方法里，首先需要判断任务是否足够小，如果足够小就直接执行任务。如果不足够小，就必须分割成两个子任务，每个子任务在调用fork方法时，又会进入compute方法，看看当前子任务是否需要继续分割成子任务，如果不需要继续分割，则执行当前子任务并返回结果。使用join方法会等待子任务执行完并得到其结果。