如何使用ThreadPool

givemepass.blogspot.tw/2015/10/threadpool.html

Handler、HanlderThread介紹單一Thread如何運作,
如果想要讓多個Thread並行, 可以使用ThreadPool。
Java的Executor框架為Thread以及使用的資源有更進一步的控制,
Executor提供一個簡單的interface, 目標是將任務的建立跟執行分開。

```
public interface Executor{
    void execute(Runnable runnable);
}

public class SimpleExecutor extends Executor{
    public void execute(Runnable runnable){
        new Thread(runnable).start();
    }
}
```

不過一般我們不會自己設計Executor，Java提供了ThreadPoolExecutor讓我們使用, 有幾個優點:
* Thread能保持存活, 等待新任務, 不會隨著任務建立再銷毀。
* Thread Pool限制最大Thread數量, 避免系統浪費。
* Thread的生命週期被Thread Pool控制。
ThreadPoolExecutor可以自行定義一些設定。

```
ThreadPoolExecutor executor = new ThreadPoolExecutor(
    int corePoolSize,
    int maxPoolSize,
    long keepAliveTime,
    TimeUnit unit,
    BlockingQueue<Runnable> workQueue
);
```

- core pool size(核心緩衝池數量):
  Thread pool的Thread下限, Thread數量不會低於這個數字,
- maxumum pool size(最大緩衝池數量):
  Thread pool的Thread最大數量, 如果Thread都被執行, 則Task會被塞進Queue直到有空閒的Thread出現為止。
  比較好的做法是根據底層硬體來決定數量。

```
int N = Runtime.getRuntime().availableProcessors();
```

- keep-alive time(最大閒置時間):
  如果超過閒置時間, 則系統會回收core Thread數量以上的Thread。
- task queue type(任務佇列類型):
  被加進來的Task, 根據策略不同, 所用的演算法也會不同。

利用allowCoreThreadTimeOut(true)這個方法, 可以在Core Thread閒置的時候, 讓系統回收。

```
int N = Runtime.getRuntime().availableProcessors();
ThreadPoolExecutor executor = new ThreadPoolExecutor(
```

```
    N,
    N * 2,
    60L,
    TimeUnit.SECONDS,
    new LinkedBlockingQueue<Runnable>());
executor.allowCoreThreadTimeOut(true);
```

這邊有一個重點可能需要注意一下。
如果一開始

```
BlockingQueue<Runnable> rList = new LinkedBlockingQueue<Runnable>();
rList.add(new Runnable(){
    @Override
    public void run() {
        //run 1
    }
});
rList.add(new Runnable(){
    @Override
    public void run() {
        //run 2
    }
});
ThreadPoolExecutor executor = new ThreadPoolExecutor(
    1,
    2,
    1,
    TimeUnit.SECONDS,
    rList);
```

一開始Thread Pool剛建立Thread, 此時並沒有任何Thread可以執行Task,
因此所有的Task將會被丟進Queue內等待, 直到有新的Task後來又被加入,
才會連同之前等待的Task一起執行。
如果要解決這個問題可以預先建立core thread。

```
executor.prestartCoreThread();
//or
executor.prestartAllCoreThreads();
```

呼叫其中一個方法可以先建立core thread。
另外還有一個陷阱就是如果core thread設定為0,
那麼講無論如何就必須等有其他的task再被加到Thread Pool內才會一起把Queue內的task執行。

```
ThreadPoolExecutor executor = new ThreadPoolExecutor(
    0,
    2,
    1,
    TimeUnit.SECONDS,
    rList);
```

如果你不想要自建Excutor, Java有內建一些Executor。

```
ExecutorService fixExecutor = Executors.newFixedThreadPool(2);
ExecutorService cacheExecutor = Executors.newCachedThreadPool();
ExecutorService singleExecutor = Executors.newSingleThreadExecutor();
```

分別是

- 固定尺寸執行器(newFixedThreadPool)
  只要固定好數量, 假設是2, 那就代表Thread永遠固定是2個, 特性是用完就丟, 執行新任務會再開新的Thread。
- 動態尺寸執行器(newCachedThreadPool)
  Thread會隨著任務多寡, 新增或刪除, 假設一個Thread被閒置60秒, 系統則會進行移除。
- 單一執行緒執行器(newSingleThreadExecutor)
  這個執行器最多只會有一個Thread, 因此是執行緒安全的, 但是相對效率會下降, 因為Task會被阻塞。

透過Callable搭配Future可以讓我們更方便管理執行緒。

```
ExecutorService executor = Executors.newSingleThreadExecutor();
Future<Object> future = executor.submit(new Callable<Object>(){
        public Object call() throws Exception{
            Object obj = doLongTask();
        return obj;
    }
});

//wait future
while(future.isDone());
Object result = future.get();
```

與Runnable不同的是Callable可以回傳結果,
透過blocking直到long task完成, 回傳物件。
Executor提供同時多個Thread並行的操作。
InvokeAll : 同時並行多個Thread, 並且透過blocking來取回每一個Task的結果。
InvokeAny: 同時並行多個Thread, 只要有一個回傳成功, 則終止剩下的Task。
也可以利用ExecutorCompletionService來查詢所完成的任務,
他會將完成任務的結果放置BlockingQueue內, 透過polling的方式, 來查詢任務是否完成。