<http://blog.csdn.net/w938706428/article/details/40821987>

多线程并发处理起来通常比较麻烦，如果你使用spring容器来管理业务bean，事情就好办了多了。spring封装了java的多线程的实现，你只需要关注于并发事物的流程以及一些并发负载量等特性。具体来说如何使用spring来处理并发事务：

首先编写具体的事务逻辑，实现Runnable接口，比如说

package com.andy.threadDemo;   
  
public class ThreadTransCode implements Runnable{   
      
  @Override   
  public void run() {   
    System.out.println("执行事务");   
  }   
}

然后要做的就是配置spring容器本身提供的线程池任务执行器：

<bean id="taskExecutor"   
    class="org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor">   
    <property name="corePoolSize" value="3" />   
    <property name="keepAliveSeconds" value="200" />   
    <property name="maxPoolSize" value="5" />   
    <property name="queueCapacity" value="25" />   
  </bean>

配置过程中主要关注于四个属性的配置

接下来要做的事就是在应用系统本身的业务bean中使用线程池任务执行器来执行并发事务。

在对应的业务bean中注入线程池任务执行器，像普通的spring bean一样。

<bean id="bizManager"   
      class="com.andy. threadDemo.BizManager" >   
      **<**propertyname**="**taskExecutor**">   
        <**refbean**="**taskExecutor**" />   
      </**property**>**   
</bean>

在业务代码中，通常以for循环的方式执行多个事务

for(int k = 0; k < n; k++) {      
        taskExecutor.execute(new ThreadTransCode());      
    }

其它繁琐的线程管理的事情就交给执行器去管理。

值得注意的事有两点

1, taskExecutor.execute(new ThreadTransCode());  激活的线程都是守护线程，主线程结束，守护线程就会放弃执行，这个在业务中式符合逻辑的，在单元测试中为了看到执行效果，需要自行阻塞主线程。

2, taskExecutor.execute(new ThreadTransCode());    的执行也不是完全安全的，在执行的过程中可能会因为需要的线程查过了线程队列的容量而抛出运行时异常，如有必要需要捕获。