<http://blog.csdn.net/qq_24046745/article/details/52816584>

## 加载xml文件

在ApplicationContext.xml文件里面添加

**[java]** [view plain](http://blog.csdn.net/qq_24046745/article/details/52816584) [copy](http://blog.csdn.net/qq_24046745/article/details/52816584)

1. xmlns:task="http://www.springframework.org/schema/task"

xmlns文件并且xsi:schemaLocation中添加

**[java]** [view plain](http://blog.csdn.net/qq_24046745/article/details/52816584) [copy](http://blog.csdn.net/qq_24046745/article/details/52816584)

1. http://www.springframework.org/schema/task

**[java]** [view plain](http://blog.csdn.net/qq_24046745/article/details/52816584) [copy](http://blog.csdn.net/qq_24046745/article/details/52816584)

1. http://www.springframework.org/schema/task/spring-task.xsd

## 在spring中配置Executor

在ApplicationContext.xml文件里面添加

**[java]** [view plain](http://blog.csdn.net/qq_24046745/article/details/52816584) [copy](http://blog.csdn.net/qq_24046745/article/details/52816584)

1. <bean id="taskExecutor" **class**="org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor">
2. <!-- 核心线程数 -->
3. <property name="corePoolSize" value="${task.core\_pool\_size}" />
4. <!-- 最大线程数 -->
5. <property name="maxPoolSize" value="${task.max\_pool\_size}" />
6. <!-- 队列最大长度 -->
7. <property name="queueCapacity" value="${task.queue\_capacity}" />
8. <!-- 线程池维护线程所允许的空闲时间，默认为60s -->
9. <property name="keepAliveSeconds" value="${task.keep\_alive\_seconds}" />
10. </bean>
11. <!-- 注解式 -->
12. <task:annotation-driven />

在dbconfig.properties添加

**[java]** [view plain](http://blog.csdn.net/qq_24046745/article/details/52816584) [copy](http://blog.csdn.net/qq_24046745/article/details/52816584)

1. maxOpenPreparedStatements=20
2. removeAbandoned=**true**
3. removeAbandonedTimeout=1800
4. logAbandoned=**true**

这是分别对线程池做配置

## 添加依赖注入

在所需要的service或者controller类里面添加

**[java]** [view plain](http://blog.csdn.net/qq_24046745/article/details/52816584) [copy](http://blog.csdn.net/qq_24046745/article/details/52816584)

1. @Resource(name = "taskExecutor")
2. **private** TaskExecutor taskExecutor;

## 使用线程池进行并发操作

代码如下

**[java]** [view plain](http://blog.csdn.net/qq_24046745/article/details/52816584) [copy](http://blog.csdn.net/qq_24046745/article/details/52816584)

1. taskExecutor.execute(**new** Runnable() {
3. @Override
4. **public** **void** run() {
5. // TODO Auto-generated method stub
6. **try** {
7. ;span style="white-space:pre">                   </span>//要进行的并发操作
8. } **catch** (Exception e) {
9. // TODO Auto-generated catch block
10. e.printStackTrace();
11. }
12. }
13. });

## 提示

注意在线程中操作变量时候变量的作用域范围。需要在这个controller或者sevice中声明变量如下

**[java]** [view plain](http://blog.csdn.net/qq_24046745/article/details/52816584) [copy](http://blog.csdn.net/qq_24046745/article/details/52816584)

1. @Controller
2. **public** **class** IndexController {
3. **int** studentscount = 0;
4. @RequestMapping(value = "/index.html")
5. **public** ModelAndView goIndex() {
6. logBefore(logger, "列表Center");
7. ModelAndView mv = **this**.getModelAndView();
8. <span style="white-space:pre">        </span>taskExecutor.execute(**new** Runnable() {
10. @Override
11. **public** **void** run() {
12. // TODO Auto-generated method stub
13. // 得到所有学生人数
14. **try** {
15. studentscount = coursesService.getStudentCount(pd);
16. } **catch** (Exception e) {
17. // TODO Auto-generated catch block
18. e.printStackTrace();
19. }
21. }
22. });
23. <span style="white-space:pre">        </span>mv.addObject("studentscount", studentscount);
24. <span style="white-space:pre">        </span>mv.setViewName("common/index");
25. <span style="white-space:pre">        </span>**return** mv;
26. ｝

版权声明：本文为博主原创文章，未经博主允许不得转载。