1. 添加Quartz包
2. 新建类，初始化QuartZ.

public class DispatchingManager

{

public async static Task Init()

{

#region 创建单元

StdSchedulerFactory stdSchedulerFactory = new StdSchedulerFactory();

IScheduler schedule = await stdSchedulerFactory.GetScheduler();

await schedule.Start();

#endregion

//监听事件

schedule.ListenerManager.AddJobListener(new CustomJobListener());

schedule.ListenerManager.AddTriggerListener(new CustomTriggerListener());

schedule.ListenerManager.AddSchedulerListener(new CustomSchedulerListener());

//作业

IJobDetail jobDetail = JobBuilder.Create<SendMessageJob>()

.WithIdentity("SendMessageJob", "group1")//使用给定的名称和组来标识作业详细信息

.WithDescription(" DispatchingManager.Init ")//设置作业的给定（人类有意义的）描述。

.Build();

//WithIntervalInSeconds(5)//5秒执行一次 WithRepeatCount(3):执行3次

//时间策略

ITrigger trigger = TriggerBuilder.Create().WithIdentity("SendMessageTrigger", "group1").StartNow().WithSimpleSchedule(x => x.WithIntervalInSeconds(5).WithRepeatCount(3)).Build();

//把作业和时间策略承载到单元上

await schedule.ScheduleJob(jobDetail, trigger);

}

}

//需要执行的操作类(无状态)

public class SendMessageJob : IJob

{

public async Task Execute(IJobExecutionContext context)

{

await Task.Run(()=> {

//业务代码

Console.WriteLine("This is SendMessageJob Execute ");

});

}

}

1. 传参方式
2. jobDetail.JobDataMap.Add("params1","zhangsan");//传递参数

var p1 = context.JobDetail.JobDataMap.GetString("params1");// SendMessageJob中获取参数

1. trigger.JobDataMap.Add("params2", "lisi");//传递参数

var p2 = context.Trigger.JobDataMap.GetString("params2");// SendMessageJob中获取参数

1. var p3= context.MergedJobDataMap.GetString("params2");// SendMessageJob中获取参数

如果key相同，就获取最新的一个不为空或者不为0值，key不同，获取最新的一个值

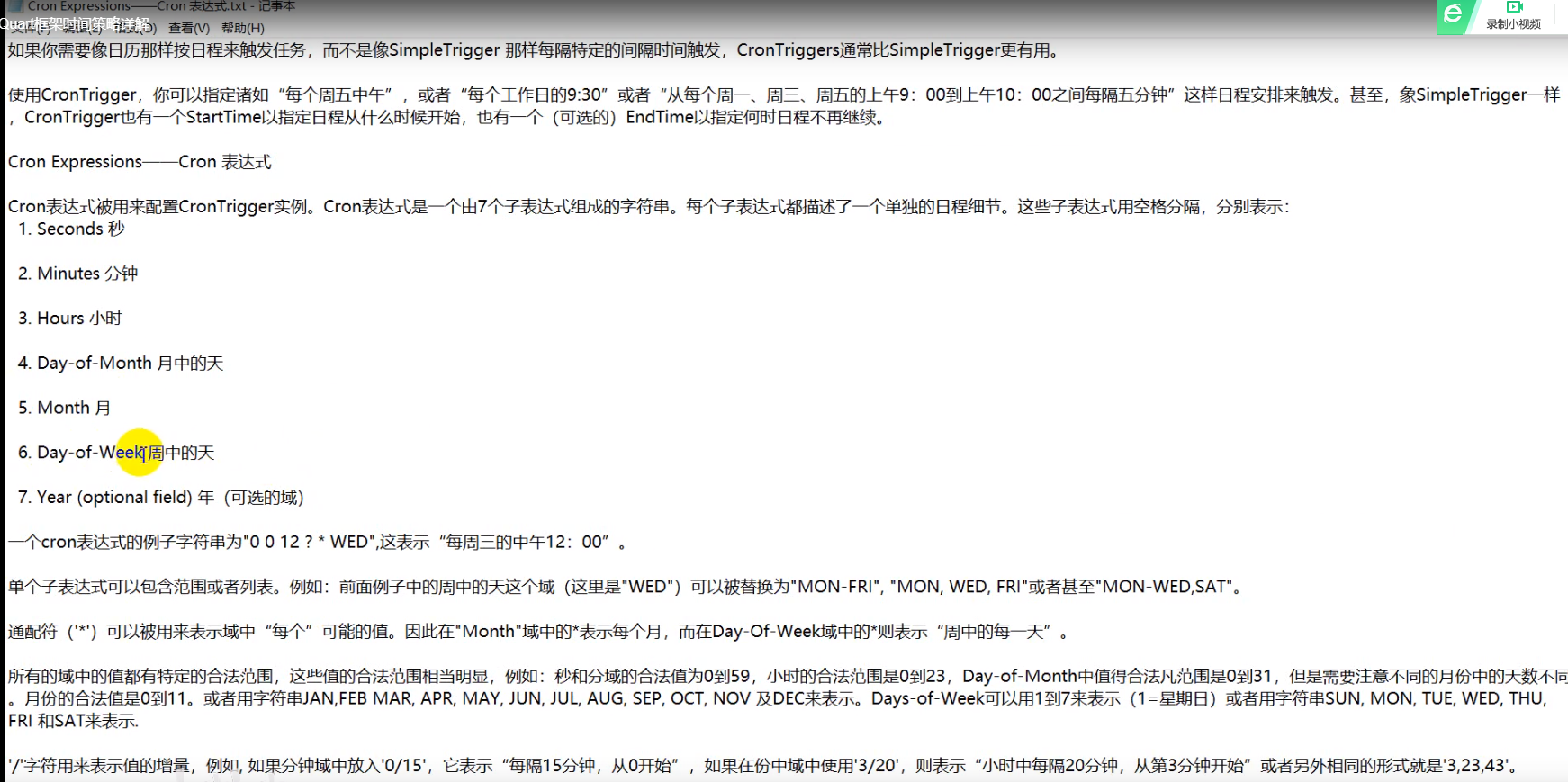
1. context.JobDetail.JobDataMap.Put("paramsint1", pi1+1);//在SendMessageJob类上添加[PersistJobDataAfterExecution]特性，可以从上次运行中获取变量：pi1第一次运行是10，第二次运行为11
2. 在SendMessageJob类上添加[DisallowConcurrentExecution]//必须在执行完本次操作以后，才能执行下一次操作,防止超时以后，又执行了SendMessageJob里面的操作
3. 时间策略

ITrigger trigger = TriggerBuilder.Create().WithIdentity("SendMessageTrigger", "group1")

.StartAt(new DateTimeOffset(DateTime.Now.AddSeconds(10)))

.WithCronSchedule("5/10 \* \* \* \* ?")//从5开始，每隔10秒运行一次

.Build();





1. 监听事件
2. //可以做日志记录
3. public class CustomJobListener : IJobListener
4. {
5. public string Name => "CustomJobListener";
6. public async Task JobExecutionVetoed(IJobExecutionContext context, CancellationToken cancellationToken = default)
7. {
8. await Task.Run(() =>
9. {
10. Console.WriteLine("取消或发生异常，走这个函数");
11. });
12. }
13. public async Task JobToBeExecuted(IJobExecutionContext context, CancellationToken cancellationToken = default)
14. {
15. await Task.Run(() =>
16. {
17. Console.WriteLine("schedule执行开始之前，走这个函数");
18. });
19. }
20. public async Task JobWasExecuted(IJobExecutionContext context, JobExecutionException jobException, CancellationToken cancellationToken = default)
21. {
22. await Task.Run(() =>
23. {
24. Console.WriteLine("schedule执行开始之后，走这个函数");
25. });
26. }
27. }