# Toolkit5.5定时任务

## 简介

由于.net core的Web服务提供了各式各样的插件。因此，基于.net core的Toolkit5.5提供了定时任务允许定时执行一个任务。

为了执行效率，定时任务将在工作线程中执行，因此，在定时任务中不得有任何Web上下文的概念，如尝试访问HttpContext，HttpRequest都将导致错误。

由于系统启动有大量的初始化工作要做，为了保证插件等一定都被正确加载，因此，定时任务的Timer将在系统启动后5分钟才会被启动。默认的，定时任务的Timer的间隔时间是1分钟，也就是说，它将每分钟扫描一次，有没有需要执行的任务，如果有，将该任务丢到工作线程的队列中，工作线程在执行完毕后，再根据该任务的时间间隔，重新设定下一个该任务的时间点。举个例子，比如系统提供的回收过期缓存的任务，它的时间间隔为5分钟。那么，如果第一次，Timer启动后，发现可以执行回收缓存的任务，它就让工作线程执行这个任务，工作线程执行完后，基于当前时间再加上5分钟，作为下一个时间点。一般来说，这个回收过程会比较短暂，通常不超过1秒，但是也需要时间，和执行的时间点相比，会延迟一点点。那么定时任务的Timer在5分钟后扫描，不会认为需要启动回收缓存的任务，而在下一分钟才会再次启动回收缓存的任务。也就是说，虽然回收过期缓存的任务间隔设定为5分钟，但是实际情况下，会间隔6分钟才会执行。

通过上述事例，我们可以了解到，Timer是不会执行任务，因此它的执行时间会很短。而工作线程才是真正执行任务的地方，如果任务很多，将不得不排队执行。

## 定时任务的开发

### ITimingJob

系统定义了ITimingJob的接口来执行相应的任务，接口定义如下：

public interface ITimingJob

{

TimeSpan Interval { get; }

void Process();

}

Interval是间隔时间。

Process方法是定时任务需要执行的工作。

### 具体实现

以定期回收过期缓存为例，下面是一个定时任务的实现：

[TimingJob(Author = "YJC", CreateDate = "2019-11-06", Description = DESCRIPTION)]

internal class CleanCacheJob : ITimingJob

{

private const string DESCRIPTION = "清理过期缓存";

private static readonly TimeSpan INTERVAL = TimeSpan.FromMinutes(5);

public TimeSpan Interval { get => INTERVAL; }

public void Process()

{

CacheManager.CleanCache();

}

public override string ToString() => DESCRIPTION;

}

这里，Interval设定为5分钟。Process就是执行清理过期缓存的API。这里重载了ToString()方法，为的是让控制台上的日志能够显示出更加友好的名字。

需要注意的是，这个类标记了TimingJobAttribute，因此它会自动注册到定时任务中。系统启动后，就会按照这个规则定时执行。

下图为执行的日志：

