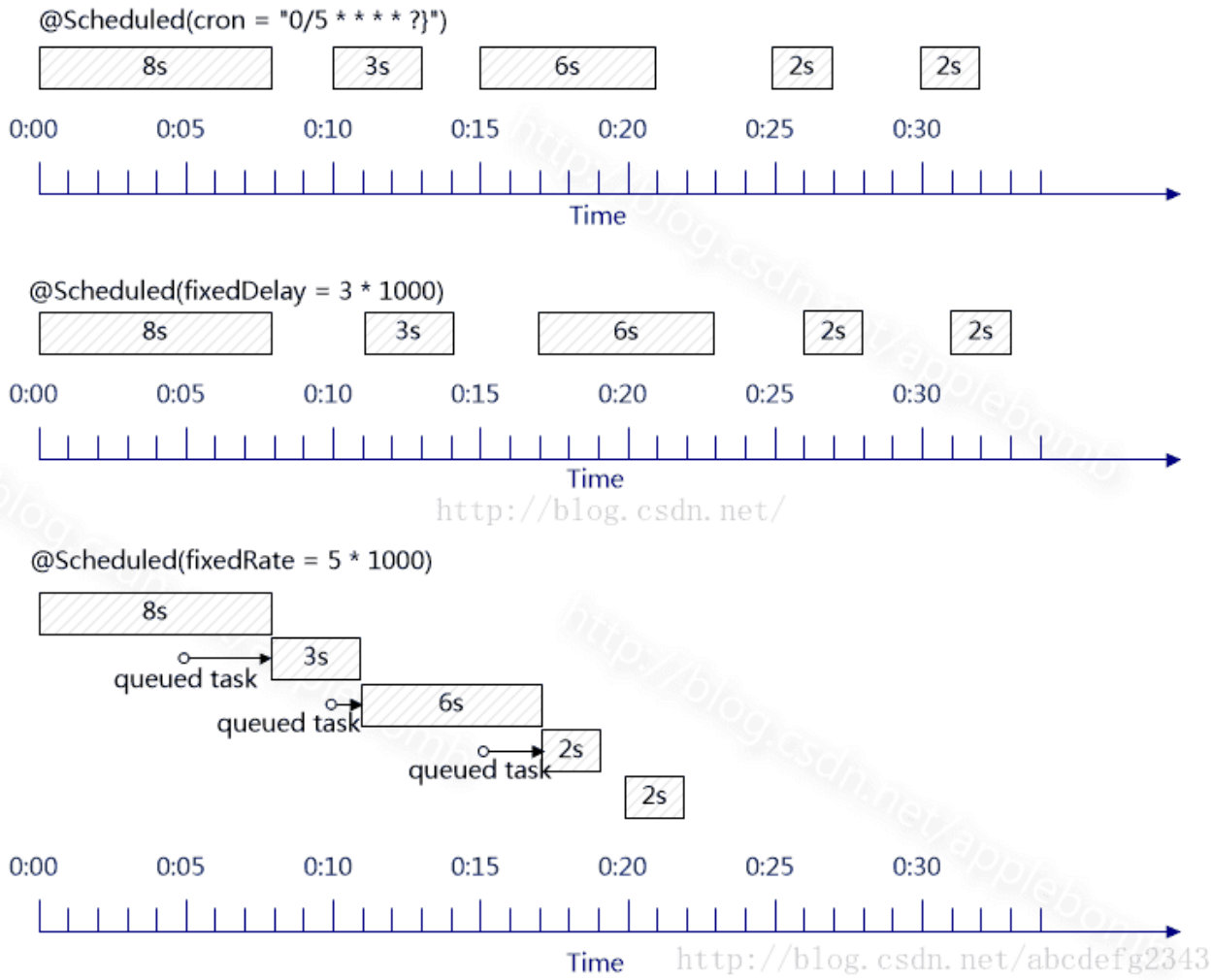
Spring定时任务

一张图来说明（任务执行长度超过周期的情况）：



虽然定时任务可以嵌入到web应用程序和WAR文件中，但下面演示一种更简单的方法创建了一个独立的应用程序。您将所有的内容打包在一个单一的、可执行的JAR文件中，用一个传统Java main()方法驱动。这也就是springboot的启动类。

1入门案例

1.1 主程序

```
package com.qianfeng.test.task;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.scheduling.annotation.EnableScheduling;

@SpringBootApplication
```

```
@EnableScheduling
public class Application {

    public static void main(String[] args) throws Exception {
        SpringApplication.run(Application.class);
    }
}
```

1.2任务类代码

```
@Component
public class MyScheduled {
    //每个5秒执行一次
    @Scheduled(fixedRate = 5000)
    public void test1() {
        System.out.println("定时任务触发了");
    }

}
```

2参数介绍

fixedRate

fixedRate表示从调用开始每次延迟多少毫秒继续调用 用法@Scheduled(fixedRate=5000)，5000的单位是毫秒，也就是间隔时间是5秒。

fixedDelay

fixedDelay表示从调用开始延时多少毫秒继续调用下一个周期 用法@Scheduled(fixedDelay=5000)，5000的单位是毫秒，也就是间隔时间是5秒。

initialDelay

fixedDelay表示在第一次执行fixedRate()或fixedDelay()任务之前延迟的毫秒数。 用法@Scheduled(fixedDelay=5000, initialDelay=10000)，单位是毫秒，表示第一次执行fixedDelay()任务之前先延迟10秒。

@Scheduled(cron="0 0 * * * *")

cron表达式相比于前几个是比较复杂的。 该模式是6个（或者7个）单独的空间分隔字段的列表：表示秒、分钟、小时、日、月、星期、（年）。

> 秒（0~59） 分钟（0~59） 小时（0~23） 天（月）（0~31，但是你需要考虑你月的天数） 月（0~11） 星期（1~7 1=SUN 或 SUN，MON，TUE，WED，THU，FRI，SAT） 年份（1970－2099）（可选）

取值可以是，*？*，*8-10*，*8/10*，/10，MON-FRI等类似语法，有什么区别呢?

其中每个元素可以是一个值(如6),一个连续区间(9-12),一个间隔时间(8-18/4)(/表示每隔4小时),一个列表(1,3,5),通配符。由于"月份中的日期"和"星期中的日期"这两个元素互斥的,必须要对其中一个设置?。现在明白了吧!

比如一下几个例子

```
"0 0 * * * *" = 每天每时整点
"*/10 * * * * *" = 每十秒（10:20:00, 10:20:10, 10:20:20 ...）触发
"0 0 8-10 * * *" = 每天早上8:00、9:00 和 10:00 触发
"0 0 6,19 * * *" = 每天6:00 和 19:00 触发
"0 0/30 8-10 * * *" = 每天8:00, 8:30, 9:00, 9:30, 10:00 和 10:30 触发
"0 0 9-17 * * MON-FRI" = 朝九晚五（周一至周五9:00-17:00的整点）触发
"0 0 0 25 12 ?" = 圣诞节（每年的12月25日00:00）触发
"0 15 10 L * ?" = 每月最后一日的上午10:15触发
"0 15 10 ? * 6L" = 每月的最后一个星期五上午10:15触发
"0 15 10 ? * 6L 2017-2027" = 2017年至2027年的每月的最后一个星期五上午10:15触发
"0 15 10 ? * 6#3" = 每月的第三个星期五上午10:15触发 1234567891011
```

字段	允许值	允许的特殊字符
秒	0-59	, - * /
分	0-59	, - * /
小时	0-23	, - * /
天	1-31	, - * ? / L W C
月	1-12 或者 JAN-DEC	, - * /
星期	1-7 或者 SUN-SAT	, - * ? / L C #
年（可选）	留空, 1970-2099	, - * /

3 并行执行任务

> 上面的方式中,如果我们开启多个定时任务,他们会使用同一个线程开启任务,可能会因为某个任务阻塞而导致执行失败,所以我们可以使用多线程并行执行任务,只需要添加一个线程池即可

4.1 并行类

```java
@Configuration
public class ScheduledConfig implements SchedulingConfigurer {


    /**
     * 创建一个长度为3的线程池对象
     * @return
     */
    @Bean(destroyMethod="shutdown")
    public Executor setTaskExecutors(){
        return Executors.newScheduledThreadPool(3); // 3个线程来处理。
    }

    /**
     * 注册线程池
     * @param scheduledTaskRegistrar
     */
    public void configureTasks(ScheduledTaskRegistrar scheduledTaskRegistrar) {
```

```
            scheduledTaskRegistrar.setScheduler(setTaskExecutors());
        }

    }
```

4.2 任务类

```
@Component
public class MyScheduled {
    @Scheduled(fixedRate = 5000)
    public void test1() {
        System.out.println(Thread.currentThread().getName()+"定时任务111触发了");
    }
    @Scheduled(fixedRate = 5000)
    public void test2() {
        System.out.println(Thread.currentThread().getName()+"定时任务222触发了");
    }
    @Scheduled(fixedRate = 5000)
    public void test3() {
        System.out.println(Thread.currentThread().getName()+"定时任务3333触发了");
    }
    @Scheduled(fixedRate = 5000)
    public void test4() {
        System.out.println(Thread.currentThread().getName()+"定时任务4444触发了");
    }
}
```

> 扫描当前配置类,启动程序即可发现多个任务由多个线程执行