任务

1 异步任务

1.1 什么是异步

异步相对于同步而言的。同步就是按顺序执行任务，一个任务做完了，才做下一件任务。而异步则相反。

1.2 Demo

这里通过代码举个例子：

写一个线程：

```
@Service
public class AsyncService {
    public void hello(){
        try {
            Thread.sleep(3000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("数据正在处理。。。");
    }
}
```

写一个跳转的Controller：

```
@RestController
public class AsyncController {
    @Autowired
    private AsyncService asyncService;

    @RequestMapping("/hello")
    public String hello() {
        asyncService.hello();//停止三秒
        return "ok";
    }
}
```

这时，我们进入网页http://localhost:8080/hello，我们发现会“转圈圈”三秒，因为我们上面设置跳转前先等待三秒。这就是同步的概念，正常状态下就是同步。

但是，我们可以用Spring提供的注解解决：

【这是一个SpringBoot项目】

- 在"等待三秒"的方法上加上注解@Async，告诉Spring这是一个异步的方法：

```
//告诉Spring这是一个异步的方法
@Async
public void hello(){
    try {
        Thread.sleep( millis: 3000);
```

- 在主启动方法上添加注解@EnableAsync，代表开启异步注解

```
//开启异步注解功能
@EnableAsync
@SpringBootApplication
public class Springboot09TaskApplication {

    public static void main(String[] args) { Spr
```

- 这时我们再启动，输入http://localhost:8080/hello，就直接跳转了，没有进行等待三秒，这就是异步。

当然，这只是个简单的引导思路，开发中可远远复杂多了。

1.3 使用异步的优点

如果没有异步的存在，就会出现用户长时间等待，并且由于当前任务还未完成，所以这时候所有的其他操作都会无响应。

1.4 异步和多线程的关系

异步其实是一个思想，而多线程是实现这个思想的方法，上面的注解也是方法。

2 邮件任务

所谓的邮件任务就是发送邮件。

- 导入依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-mail</artifactId>
</dependency>
```

父依赖中有这个东西：

```
</dependency>
<dependency>
  <groupId>com.sun.mail</groupId>
  <artifactId>jakarta.mail</artifactId>
  <version>1.6.4</version>
  <scope>compile</scope>
</dependency>
```

所以，核心还是熟悉的javax.mail

- 进行配置【这里只给出QQ邮箱】

  application.properties:

```
spring.mail.username=1084987683@qq.com
spring.mail.password=**************
spring.mail.host=smtp.qq.com
# 开启加密验证【QQ邮箱有的】
spring.mail.properties.mail.smtp.ssl.enable=true;
```

这里我们使用QQ邮箱。

password的获取方法：

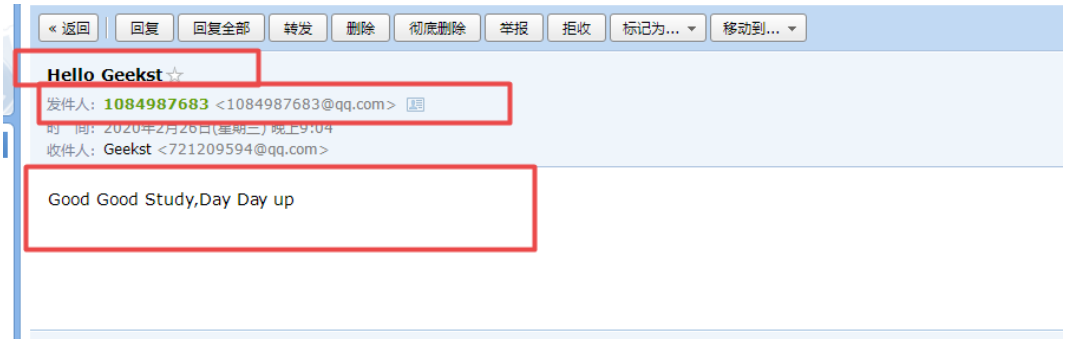
QQ邮箱---》设置--》账户---》开启SMTF服务---》验证后，会受到一串编码

- 测试类

```java
package com.kuang;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.mail.SimpleMailMessage;
import org.springframework.mail.javamail.JavaMailSenderImpl;
import org.springframework.mail.javamail.MimeMessageHelper;

import javax.mail.MessagingException;
import javax.mail.internet.MimeMessage;
import java.io.File;

@SpringBootTest
class SpringbootO9TaskApplicationTests {
    @Autowired
    JavaMailSenderImpl mailSender;

    /**
     * 简单的邮件发送
     */
    @Test
    void contextLoads() {
        SimpleMailMessage mailMessage = new SimpleMailMessage();
        mailMessage.setSubject("Hello Geekst");//邮件的主题
        mailMessage.setText("Good Good Study,Day Day up");//文本
        mailMessage.setTo("721209594@qq.com");//收件人
        mailMessage.setFrom("1084987683@qq.com");//发件人
        mailSender.send(mailMessage);
    }

    /**
     * 复杂的邮件发送
     */
    @Test
    void contextLoads2() throws MessagingException {
        MimeMessage mimeMessage = mailSender.createMimeMessage();

        MimeMessageHelper helper = new MimeMessageHelper(mimeMessage,
true);
        helper.setSubject("复杂的邮件");//主题
        helper.setText("<p style='color:red'>Good Good Study! </p>",
true);//邮件的内容，true代表支持识别HTML

        //附件
        helper.addAttachment("1.jpg", new
File("C:\\Users\\10849\\Desktop\\1.jpg"));

        helper.setTo("721209594@qq.com");//收件人
        helper.setFrom("1084987683@qq.com");//发件人

        mailSender.send(mimeMessage);
    }
}
```

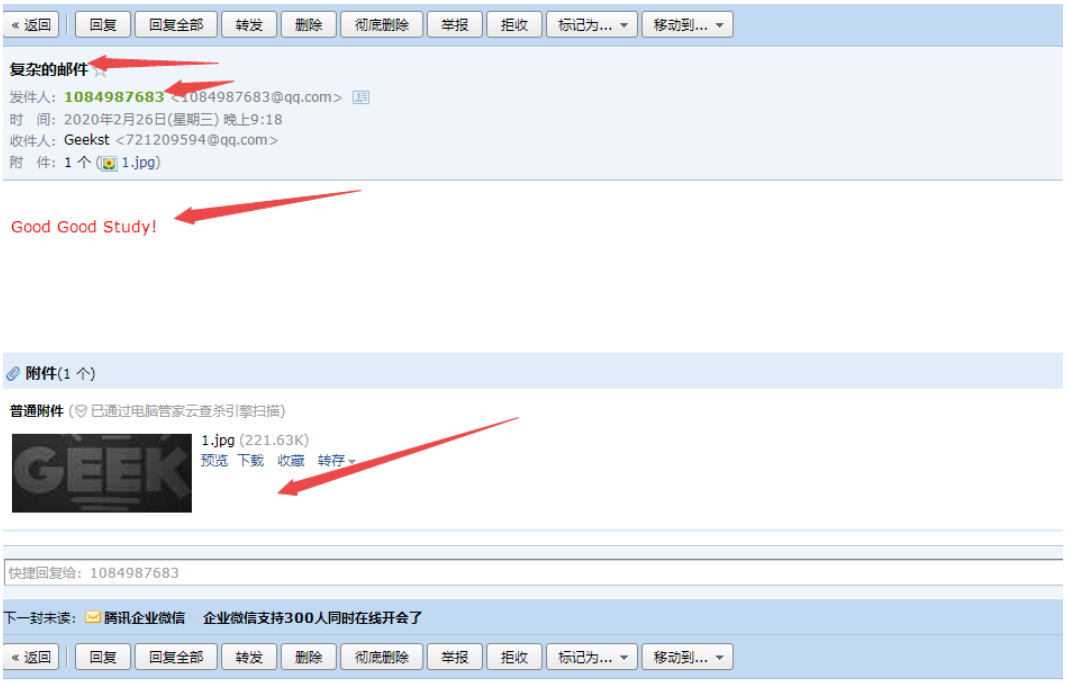
这里我们测试了两种邮件的发送：

- 简单邮件【没有附件】



- 复杂邮件【有附件】

  当然，附件可以不止一个，多写几个helper.addAttachment()方法就好。



3 定时任务

```
记住两个重要的类：
- TaskScheduler ——  任务调度者
- TaskExecutor  ——   任务执行者

两个注解：
- @EnableScheduling【开启定时功能】
- @Scheduled【什么时候执行】

一种表达式：
- Cron表达式
```

- 在主启动类上加 开启功能的注解

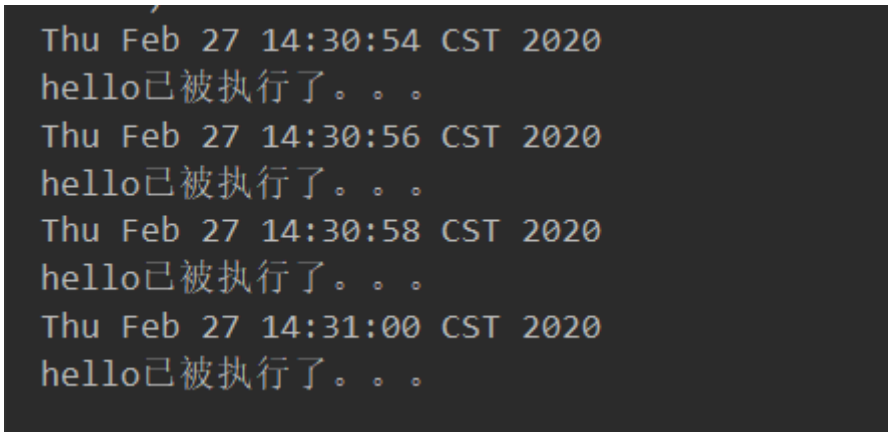
```java
@EnableScheduling//开启定时功能的注解
@SpringBootApplication
public class Springboot09TaskApplication {

    public static void main(String[] args) {
        SpringApplication.run(Springboot09TaskApplication.class, args);
    }

}
```

- 直接在方法上加注解

```java
@Service
public class ScheduledService {

    /**
     * 在一个特定的时间执行这个方法
     * 秒    分    时    日    月    周几
     */
    @Scheduled(cron = "0/2 * * * * ?")//表示每两秒执行一次任务
    public void hello() {
        System.out.println(new Date());
        System.out.println("hello已被执行了。。。");
    }
}
```

- 测试运行

  我们直接启动主启动类就好了，因为这是异步的任务，SpringBoot会帮我们直接运行。



- cron表达式

  【如果觉得表达式难以理解，那么直接去网站生成吧：
  http://www.bejson.com/othertools/cron/】

  从左到右（用空格隔开）：秒 分 时 日 月 周几 年

字段	允许值	允许的特殊字符
秒（Seconds）	0~59的整数	, - * / 四个字符
分（*Minutes*）	0~59的整数	, - * / 四个字符
小时（*Hours*）	0~23的整数	, - * / 四个字符
日期（*DayofMonth*）	1~31的整数（但是你需要考虑你月的天数）	,- * ? / L W C 八个字符
月份（*Month*）	1~12的整数或者 JAN-DEC	, - * / 四个字符
星期（*DayofWeek*）	1~7的整数或者 SUN-SAT （1=SUN）	, - * ? / L C # 八个字符
年(可选，留空)（*Year*）	1970~2099	, - * / 四个字符

（1）*：表示匹配该域的任意值。假如在Minutes域使用*, 即表示每分钟都会触发事件。

（2）?：只能用在DayofMonth和DayofWeek两个域。它也匹配域的任意值，但实际不会。因为DayofMonth和DayofWeek会相互影响。例如想在每月的20日触发调度，不管20日到底是星期几，则只能使用如下写法： 13 13 15 20 * ?, 其中最后一位只能用?，而不能使用*，如果使用*表示不管星期几都会触发，实际上并不是这样。

（3）-：表示范围。例如在Minutes域使用5-20，表示从5分到20分钟每分钟触发一次

（4）/：表示起始时间开始触发，然后每隔固定时间触发一次。例如在Minutes域使用5/20,则意味着5分钟触发一次，而25，45等分别触发一次.

（5）,：表示列出枚举值。例如：在Minutes域使用5,20，则意味着在5和20分每分钟触发一次。

（6）L：表示最后，只能出现在DayofWeek和DayofMonth域。如果在DayofWeek域使用5L,意味着在最后的一个星期四触发。

（7）W:表示有效工作日(周一到周五),只能出现在DayofMonth域，系统将在离指定日期的最近的有效工作日触发事件。例如：在 DayofMonth使用5W，如果5日是星期六，则将在最近的工作日：星期五，即4日触发。如果5日是星期天，则在6日(周一)触发；如果5日在星期一到星期五中的一天，则就在5日触发。另外一点，W的最近寻找不会跨过月份 。

（8）LW:这两个字符可以连用，表示在某个月最后一个工作日，即最后一个星期五。

（9）#:用于确定每个月第几个星期几，只能出现在DayofMonth域。例如在4#2，表示某月的第二个星期三。