# 重新认识Spring Boot

## Spring Boot与Spring Cloud的关系

Spring Cloud是基于Spring Boot提供的一条微服务解决方案，配置中心、服务注册和负 载均衡等，都是在Spring Boot这个框架上来构建的，所以Spring Boot是构建Spring Cloud 生态的基石。学习过Spring Bot的同学都知道，Spring Boot是帮助开发者快速构建一个 Spring Framework及Spring生态体系的应用解决方案，也是Spring Framework对于“约 点优先配置”（Convention over Configuration）理念的最佳实践。

## Spring的诞生

Spring Framework的起源需要追溯到2002年，当时很多知名公司都用Java EE标准及EJB 容器作为主要的软件解决方案，其中EJB是J2EE规范的核心内容，它与Spring Framework 的诞生密切相关。EJB是一种组件模式，使得开发人员只需要关注业务开发，不需要关 心底层的实现机制，所以当时很多大公司都采用EJB来部署自己的系统，但是在使用 EJB之后各种问题接踵而至，比如配置太过繁琐、臃肿、低效等。随后Spring Framework 诞生，Spring是一个轻量级框架，它的主要目的是简化JavaEE的企业级应用开发，而达 到这个目的的关键手段是IOC/DI和AOP。除此之外，Spring能够快速继承和开发，比如 Spring集成MyBatis、Spring集成Structs2等，人们很快就抛弃了繁重的EJB标准，Spring 逐步成了Java EE开发的标准。

## Spring IOC/DI

IoC（Inversion of Control）和DI（Dependency Injection）分别表示控制反转和依赖注入。

1. **IoC（控制反转）**

IoC（控制反转）实际上就是把对象的生命周期托管到Spring容器中，而反转是指对象的获取方式被反转了，传统方式是通过new来创建对象，但这种方式会使代码之间的耦合度非常高。当使用Spring Ioc容器之后，客户端可以不再通过new来创建这些对象，而是直接在IoC容器中获得。在Spring中，主要通过XML的方式来定义Bean，Spring会解析XML文件，把定义的Bean装载到容器中。

1. **DI（依赖注入）**

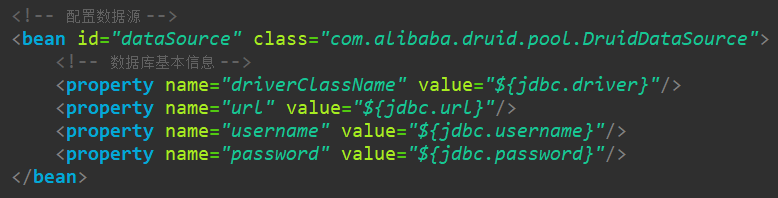
DI（依赖注入）简单理解就是IoC容器在运行期间，动态地把某种依赖关系注入组件中。只需要在Spring的配置文件中描述Bean之间的依赖关系，IoC容器在解析该配置文件的时候，会根据Bean的依赖关系进行注入，这个过程就是依赖注入。实现依赖注入的方式有三中，分别是接口注入、构件方法注入、setter方法注入。不过现在基本上都是基于注入的方式来描述Bean之间的依赖关系，比如@Autowired、@Inject、@Resource，但不管形式怎么变化，本质上还是一样的。

## Bean装配方式的升级

基于XML配置的方式很好地完成了对象生命周期的描述和管理，但是随着项目不断扩 大，XML配置也逐渐增多，使得配置文件难以管理。另一方面，项目中依赖关系越来越 复杂，配置的文件变得难以理解。为此，需要进行解决。

1. 随着JDK1.5带来的注解支持， Spring从2.x开始，可以使用注解的方式来Bean进行声明和注入，大大减少了XML的配置量。

2）以往都是使用XML的方式装配Bean，如下：



Spring升级到3.x后，提供了JavaConfig的能力，可以完全取代XML，通过Java代码的方式来完成Bean的注入。



@Configuration表示这个类是一个配置类，用于将Bean注入到IoC容器中，@Bean 表示这是一个Bean，方法的返回值类型是Bean的类型，默认方法的名称是Bean 的名称，即Bean的id，如果需要注入依赖，在方法的内部通过代码的形式完成。

1. 以往在Spring配置文件中使用<context:component-scan base-package=””/>形式扫描指定报路径下带有@Component、@Controller、@Service、@Repository等注解的类，将这些类作为Bean注入到IoC容器中，而现在也可以使用@ComponentScan注解完成。
2. 使用@Import注解可以完成<import resource=””/>的作用，导入其他的配置文件。

虽然可以通过注解的方式来装配Bean，可以在一定程度上减少XML配置带来的问题， 但实际上本质问题并没有得到解决，比如：

·依赖过多。Spring可以整合几乎所有常用的技术框架，如JSON、MyBatis、Redis、

Log等，不同依赖包的版本很容器导致版本兼容的问题。

·配置过多。以Spring使用JavaConfig方式整合MyBatis为例，需要配置注解驱动、 配置数据源、配置MyBatis、配置事务管理器等，这些只是继承一个技术组件需 要的基础配置，在一个项目中这类配置很多，开发者需要做很多类似的重复工作。

·运行和部署很繁琐。需要先把项目打包，再部署到容器上。

知道Spring Boot出现，本质问题才得到了解决。

## Spring Boot的诞生

由于Spring框架所带来的问题以及有人提出了在Spring框架中支持无容器的Web应用 程序体系结构，Spring Boot的诞生。Spring Boot并不是一个新的技术框架，其主要作用 就是简化Spring应用的开发，开发者只要通过少量的代码就可以创建一个产品级的 Spring应用，而达到这一目的的最核心思想就是“约定优于配置（Convention Over Configuration）”。

## 理解约定优于配置

约定优于配置（Convention Over Configuration）是一种软件设计规范，目的在于减少配 置的数量或降低理解难度，从而提升开发效率。它并不是一种新的思想，在接触Java 时就会发现有很多地方都有这种思想的提现，如数据库中的表名的设计对应到Java中 实体类的名字，就是一种约定，我们可以从这个实体类的名知道它对应数据库中哪张表。 再比如，每个公司都会有自己的开发规范，开发者按照开发规范可以在一定程度上减少 Bug的数量，增加可读性和可维护性。

在Spring Boot中，约定由于配置的思想主要体现在以下方面（包括但不限于）：

·Maven目录结构的约定

·Spring Boot默认的配置文件及配置文件中配置属性的约定

·对于Spring MVC的依赖，自动依赖内置的Tomcat容器

·对于Starter组件自动完成装配

## Spring Boot的核心

Spring Boot是基于Spring Framework体系来构建的，所以它并没有什么新的东西，但是 要学好Spring Boot，必须知道它的核心：

·Starter组件，提供开箱即用的组件

·自动装配，自动根据上下文完成Bean的装配

·Actuator，Spring Boot应用的监控

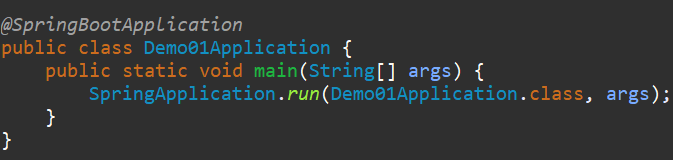
·Spring Boot CLI，基于命令行工具快速构建Spring Boot应用

其中，最核心的部分应该是自动装配，Starter组件的核心部分也是基于自动装配来实现 的。

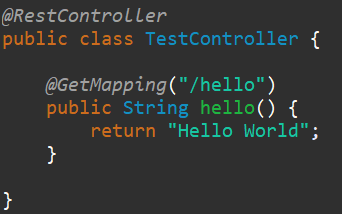
## 构建Spring Boot应用

构建Spring Boot应用的方式有很多，比如在<https://start.spring.io>网站上通过图形界面 来完成创建。如果使用的Intelli IDEA，则可以直接在这个工具上来创建Spring Boot项目， 默认也是使用<https://start.spring.io>来构建的。

1. 创建的Spring Boot应用会包含一个启动类

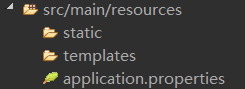


在不做任何改动的情况下，就可以直接运行该启动类中的main方法来启动Spring Boot。由于默认情况下没有任何URI映射，所以看不出效果，我们创建控制层增加 一个Controller定义一个Restful接口，代码如下：



接着，运行启动类，服务启动完成后在浏览器中即可访问刚刚定义的Restful接口。

1. Spring Boot应用会包含一个resources目录，在resources目录中会包含static和templates目录、application.properties文件，分别存放静态资源和前端模板，以及Spring Boot应用配置文件。



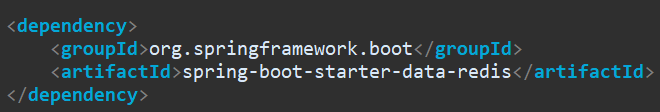
可以看出，以往使用Spring来构建一个Web项目需要很多基础操作，如在web.xml中 配置控制器、配置Spring的XML文件等。而Spring Boot省略了这些繁琐的基础性工作， 使得开发者可以只需要关注业务本身。

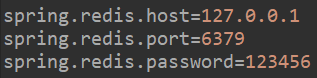
# Spring Boot自动装配的原理

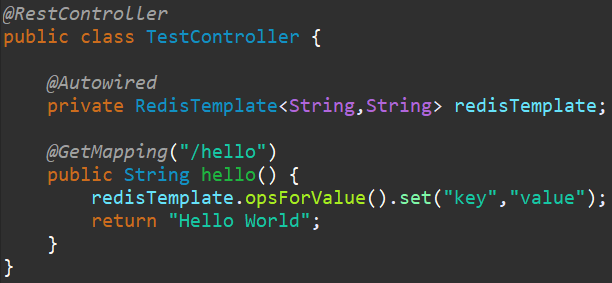
## 什么是自动装配

在Spring Boot中，自动装配是Starter的基础，也是Spring Boot的核心。那么什么是自 动装配呢？简单来说，就是自动将Bean装配到IoC容器中。下面通过一个示例来理解 什么是自动装配。

1. 在Spring Boot应用中添加Starter依赖：



1. 在application.propert ies中配置Redis的数据源：
2. 在TestController类中使用RedisTemplate实现Redis的操作：

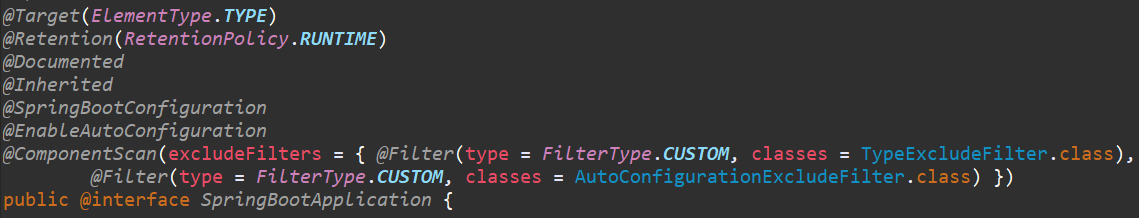


在以上案例中，并没有通过XML形式或者注解形式把RedisTemplate注入IoC容器中， 但是在TestController中却可以直接使用@Autowired来注入redisTemplate实例，这就说 明，IoC容器中已经存在RedisTemplate。这就是Spring Boot的自动装配机制。

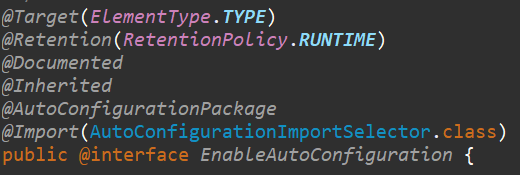
那么如何只添加一个Starter依赖，就能完成该依赖组件相关Bean的自动注入呢？不难 猜出，这个机制的实现一定基于某种约定或者规范，只要Starter组件符合Spring Boot 中自动装配约定的规范，就能实现自动装配。

## 自动装配的实现

自动装配在Spring Boot中是通过@EnableAutoConfiguration注解开启的，这个注解的声 明在启动类注解@SpringBootApplication中。进入@SpringBootApplication注解中可以发 现：



进入到@EnableAutoConfiguration注解，如下：



可以发现@EnableAutoConfiguration由@AutoConfigurationPackage注解和@Import注 解组成。@AutoConfigurationPackage注解的作用是把使用了该注解的类所在的包及子包 下所有组件扫描到Spring IoC容器中；@Import注解用于导入配置，但在此处它导入的 并不是一个Configuration的配置类，而是一个AutoConfigurationImportSelector类，不 管AutoConfigurationImportSelector是什么，它一定会实现配置类的导入，至于导入的 方式和普通的@Configuration有什么区别，需要我们去分析。由于书上与实际有别，以 下简单来总结一下核心过程：

1. 在@SpringBootApplication注解的中@EnableAutoConfiguration注解中，通过@Import(AutoConfigurationImportSelector.class)实现配置类的导入，但是这里并不是传统意义上的单个配置类装配。
2. AutoConfigurationImportSelector类实现了@ImportSelector接口，重写了方法selectImports，该方法用于实现选择性批量配置类的装配。
3. 通过Spring提供的SpringFactoriesLoader机制，扫描classpath路径下的META-INF/spring.factories，读取需要实现自动装配的配置类。
4. 通过条件筛选（如@EnableAutoConfiguration注解的exclude属性可以指定哪些Bean不进行自动装配）的方式，把不符合条件的配置类移除，最终完成自动装配。

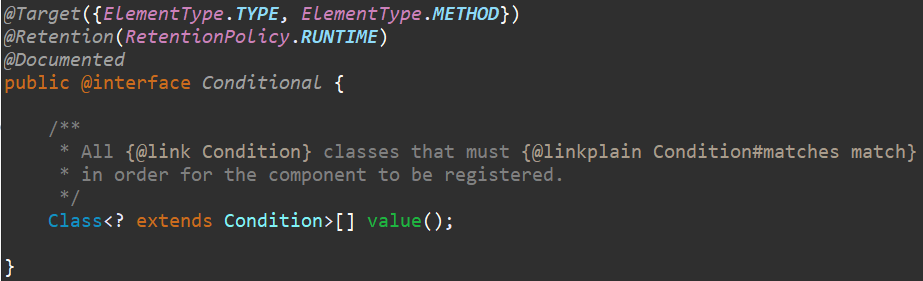
# 条件装配

条件装配指的是根据给定的条件或应用的环境条件来决定是否装配指定的Bean。

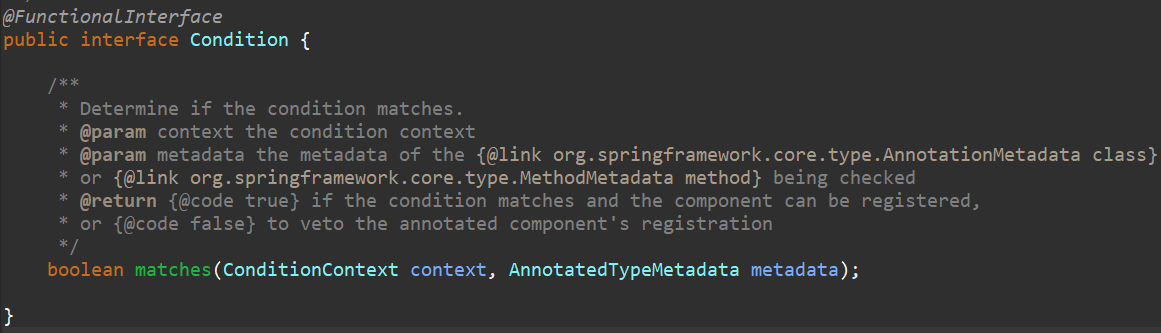
## @Conditional条件装配

@Conditional是Spring Framework提供的一个核心注解，这个注解的作用是提供自动装 配的条件约束，然后根据指定条件来决定是否装配该Bean，一般与@Configuration注解 和@Bean注解配合使用。

@Conditional的注解类代码如下：



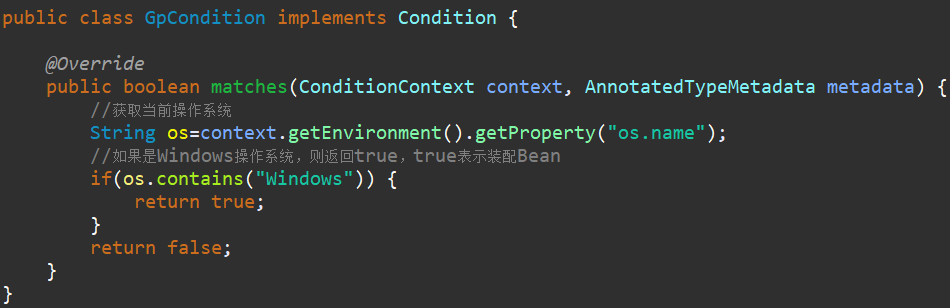
该注解可以接收一个Condition接口的实现类类型的数组。查看Condition接口，如下：



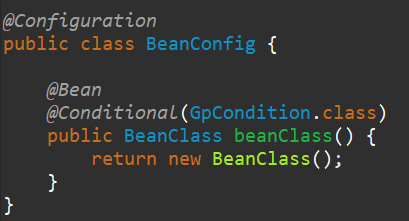
Condition是一个函数式接口，只提供了一个matches方法，该方法用于提供条件匹 配规则，若方法返回true则表示可以注入Bean，反之则不注入。

示例如下：

1. 自定义一个类，实现Condition接口，覆写matches方法用于提供匹配条件，如果当前操作系统是Windows，则返回true，否则返回false



1. 创建一个配置类BeanConfig，装载一个BeanClass类型的Bean（BeanClass是自定义的一个类）



在BeanClass的Bean声明方法中使用@Conditional ( GbpConditaion.class )，其中具 体的条件是我们自定义的GpCondition类，如果GpCondition类中的matchs方法返 回true， 则将BeanClass装载到Spring IoC容器中。

1. 进行测试，测试是否装载BeanClass这个Bean



如果当前操作系统是Windows操作系统，则该Bean不为null，如果当前操作系统 不是Windows操作系统，应用会启动失败，提示没有发现BeanClass这个Bean。

## Spring Boot中的条件装配注解

在Spring Boot中，针对@Conditional做了扩展，提供了更简单的使用形式，这些注解如 下：

**@ConditionalOnBean/@ConditionalOnMissingBean**

当IoC容器中存在某个Bean或者不存在某个Bean时才进行Bean的装载。

**@ConditionalOnClass/ConditionalOnMissingClass**

当classpath下存在指定类或者不存在指定类时才进行Bean的装载

**@ConditionalOnCloudPlatform**

只有运行在指定的云平台上才进行Bean的装载

**@ConditionalOnExpression**

基于SpEl表达式的条件判断，根据判断结果来决定是否进行Bean的装载

**@ConditionalOnJava**

只有运行指定版本的Java才会进行Bean的装载

**@ConditionalOnJndi**

只有指定的资源通过JNDI加载后才进行Bean的装载

**@ConditionalOnWebApplication/ConditionalOnNotWebApplication**

如果是Web应用或者不是Web应用时才进行Bean的装载

**@ConditionalOnProperty**

系统中指定的对应的属性是否有对应的值，如果有，则进行Bean的装配

**@ConditionalOnResource**

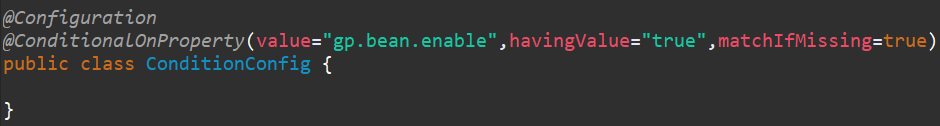
当装载的Bean依赖的指定资源在classpath中存在，才进行Bean的装载。

**@ConditionalOnSingleCandidate**

只有在确定了给定Bean类的的那个候选项时才会装载Bean。

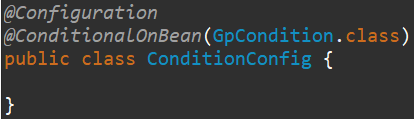
这些注解只需要添加到@Configuration配置类的类级别或者方法级别，然后根据每个注 解的作用来传参就行。下面演示几种注解的使用。

1）@ConditionalOnProperty



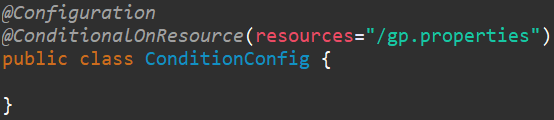
在applicaion.properties或application.yml文件中当gp.bean.enable=true时才会加载 ConditioinConfig这个Bean，如果没有匹配到gp.bean.enable属性也会加载 ConditionConfig这个Bean，因为matchIfMissing=true，默认值是false。

2）@ConditionalOnBean



当Spring IoC容器中存在GpCondition这个类型的Bean时，才会装载ConditionConfig。

3）@ConditionalOnResource



当classpath下存在gp.properties，则装配ConditionConfig这个Bean。

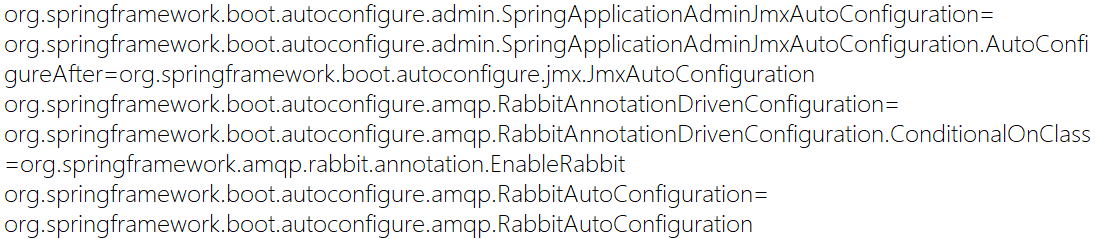
## 通过文件实现条件装配

除了@Conditional注解类和Spring Boot中提供的条件装配注解外，在Spring Boot中还 提供了spring-autoconfigure-metadata.properties文件来实现批量自动装配条件配置。它 的作用和@Conditional和Spring Boot中提供的条件装配注解的作用是一样的，只是将这 些条件配置放在了配置文件中。并且这种配置方法的好处在于，它可以有效地降低 Spring Boot的启动时间，通过这种过滤方式可以减少配置类的加载数量，因为这个过滤 发生在配置类的装载之前，所以它可以降低Spring Boot启动时装载Bean的耗时。 通 过这种配置的方式来实现条件过滤必须要遵循两个条件：

·配置文件的路径名称必须是/META-INF/spring-autoconfigure-metadata.properties

·配置文件中的key的配置格式：自动配置类的类全路径名.条件=值

如spring-boot-autoconfigure.jar包中的 /META-INF/spring-autoconfigure-metadata.pro perties文件



# 实现Starter组件

## 为什么要实现Starter组件

对于自动装配的原理进行分析之后，我们可以基于这个机制来实现一个Starter组件， 以便加深大家对自动装配及Starter组件的理解。同时Spring Boot官方提供的Starter并 不能囊括所有的技术组件，在工作中，如果自己的项目需要支持Spring Boot，也需要开 发Stater组件。从Spring Boot官方提供的Starter的作用来看，Starter组件主要有三个 功能：

·设计相关组件的Jar包依赖

·自动实现Bean的装配

·自动声明并且加载application.properties文件中的属性配置

## Starter的命名规范

Starter的命名主要分为两类，一类是官方命名，另一类是自定义组件命名。

·官方命名的格式为：spring-boot-starter-模块名称，比如spring-boot-starter-web

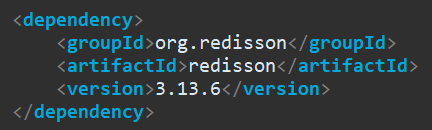
·自定命名的格式为：模块名称-spring-boot-starter，比如mybatis-spring-boot-starter

简单来说，官方命名模块名放在最后，自定义组件中模块名放在最前面。这种命名格式并不是强制性的，是一种约定俗称的方式，可以让开发者更容易识别。

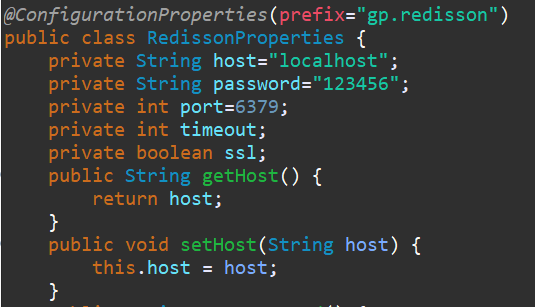
## 实现基于Redis的Starter

虽然Spring Boot官方提供了spring-boot-starter-data-redis组件来实现RedisTemplate的 自动装配，但是我们仍然基于前面学到的思想来实现一个基于Redis简化版本的Starter 组件。

1. 创建一个Spring Boot工程，命名为redis-spring-boot-starter
2. 添加Jar包，Ression提供了在Java中操作Redis的功能，并且基于Redis的特性封装了很多可直接使用的常见，比如分布式锁



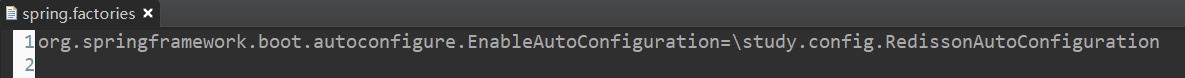
1. 定义属性类，实现在application.properties中配置Redis的连接参数，由于只是一个简单版本的Demo，所以只简单定义了一些必要参数。另外，@ConfigurationProperties这个注解的作用是把当前类中的属性和配置文件（properties/yml）中的配置进行绑定，并且前缀是gp.redission



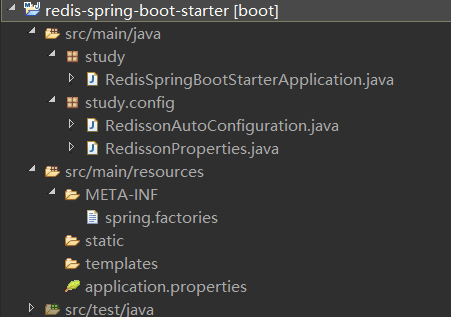
1. 定义需要自动装配的配置类，主要就是把RedisClient装配到IoC容器中，值得注意的是@ConditionalOnClass，它表示一个条件，在当前场景中表示的是：在classpath下存在Redission这个类的时候，RedissionAutoConfiguration才会实现自动装配。另外，这里只演示了一种单机的配置模式，除此之外，Redisson还支持集群、主从、哨兵等模式的配置。大家有兴趣的话可以基于案例去扩展，建议使用config.fromYAML方式，直接加载配置完成不同模式的初始化，这会比根据不同模式的判断来实现配置化的方式更加简单。



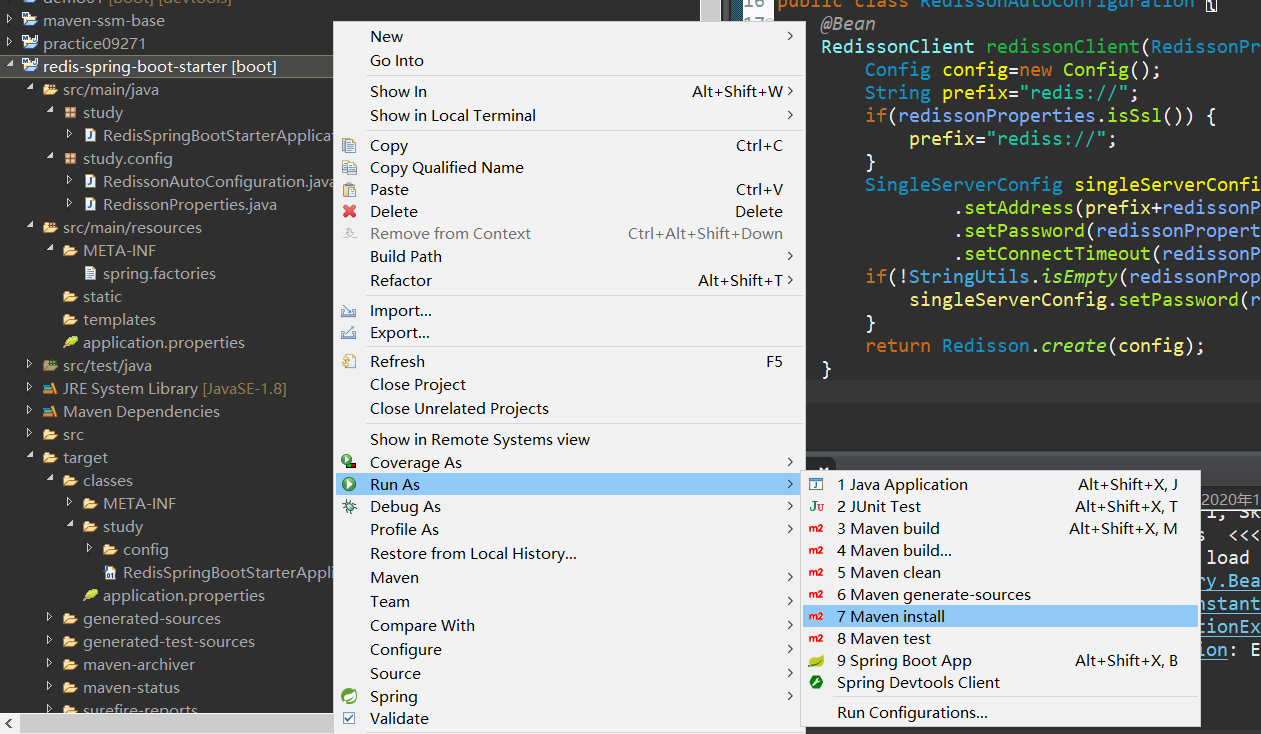
1. 在resources下创建META-INF/spring.factories文件，使得Spring Boot程序可以扫描到该文件完成自动装配，key和value对应如下：



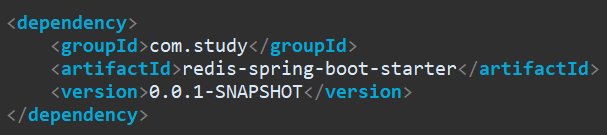
\study.config.RedissonAutoConfiguration是自定义的包类路径，最后整合项目的目录 结构如下：



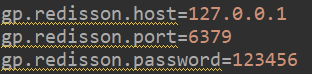
1. 将redis-spring-boot-starter打包install安装到仓库中，供其他项目引用。



1. 最后一步，在其他项目中引用该Starter，只需要做两个步骤：添加Starter依赖，设置属性配置：



在application.propertes中配置host和port，这个属性会自动绑定到RedissonProperties类中定义的属性上（依靠的是@ConfigurationProperties注解来实现这样的功能）。



8）进行测试，查看RedissonClient是否注入成功：



至此，一个非常简易的手写Starter就完成了。