* 学习目标

* 能够分析SpringBoot的starter配置

* 能够掌握SpringBoot的SpringBootApplication注解

* 能够掌握Spring的Conditional相关的注解

* 能够掌握SpringBoot启动流程

----

* 回顾

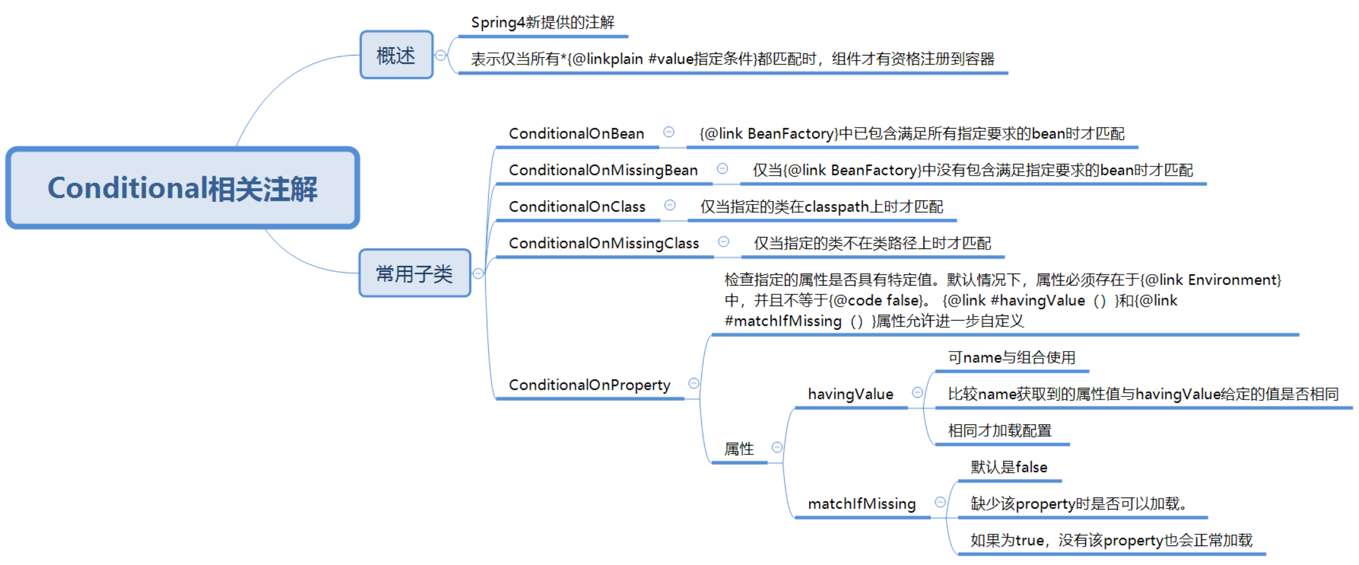
* 能够分析SpringBoot的starter配置

```
* pom:spring-boot-starter-parent extends spring-boot-dependencies
* spring-boot-dependencies
 * 统一了版本号
 * 引入常见的依赖
  * junit
* spring-boot-starter-web extends spring-boot-starters
  * spring-boot-starter
  * spring-boot-starter-json
  * spring-boot-starter-tomcat
  * spring-boot-starter-validation
  * spring-web
  * spring-webmvc
* spring-boot-starters extends spring-boot-parent
                       extends spring-boot-dependencies
* spring-boot-starter  extends spring-boot-starters
  * spring-boot
  * spring-boot-autoconfigure
  * spring-boot-starter-logging
  * jakarta.annotation-api
  * spring-core
  * snakeyaml

 * 总结：每个层次都可以点进去看看，这里就不记录了
```

* 能够掌握SpringBoot的SpringBootApplication注解

```
* @SpringBootApplication
 * 这个注解等于：@Configuration，@ComponentScan，@EnableAutoConfiguration
* @EnableAutoConfiguration
 * 启用Spring Application Context的自动配置
 * 尝试猜测和配置您可能需要的bean
 * 通常根据您的类路径和定义的bean来应用自动配置类。
 * 例如，如果您的类路径上有{@code tomcat-embedded.jar}，
   * 则可能需要一个{@link TomcatServletWebServerFactory}
   *（除非您已定义了自己的{@link ServletWebServerFactory} bean
 * @AutoConfigurationPackage
 * @Import(AutoConfigurationImportSelector.class)

* @AutoConfigurationPackage
  * 指示包含带注解的类的包应向 {@link AutoConfigurationPackages}注册
  * @Import(AutoConfigurationPackages.Registrar.class)
  * 测试可以断点调试Registrar的registerBeanDefinitions方法，查看效果
    * 发现：com.lg包注册了

* AutoConfigurationImportSelector
 * 处理自动配置
 * 断点调试：getCandidateConfigurations--SpringFactoriesLoader.loadFactoryNames
   * 发现它们会去加载：项目中引进来依赖中有META-INF/spring.factories文件
   * 这次断点调试发现加载了129自动配置
* 查看spring-boot-autoconfigure:spring.factories文件
   * 发现里面一堆自定配置类
    * AopAutoConfiguration
      * 演示禁止AOP的例子：spring.aop.auto=false
    * HttpEncodingAutoConfiguration
    * WebMvcAutoConfiguration
    * HttpMessageConvertersAutoConfiguration
    * TransactionAutoConfiguration

* 总结：
  * SpringBootApplication
    * 三个注解构成：@Configuration，@ComponentScan，@EnableAutoConfiguration
  * EnableAutoConfiguration
    * @AutoConfigurationPackage
    * @Import(AutoConfigurationImportSelector.class)
  *AutoConfigurationImportSelector
    * getCandidateConfigurations
    * SpringFactoriesLoader:EnableAutoConfiguration.class
       * meta-inf/spring.factories
    * spring.factories
      * AopAutoConfiguration
      * ...
      * @Conditional
```

* 能够掌握Spring的Conditional相关的注解


Conditional相关注解
概述
Spring4新提供的注解
表示仅当所有*{@linkplain #value指定条件}都匹配时，组件才有资格注册到容器
常用子类
ConditionalOnBean
{@link BeanFactory}中已包含满足所有指定要求的bean时才匹配
ConditionalOnMissingBean
仅当{@link BeanFactory}中没有包含满足指定要求的bean时才匹配
ConditionalOnClass
仅当指定的类在classpath上时才匹配
ConditionalOnMissingClass
仅当指定的类不在类路径上时才匹配
ConditionalOnProperty
检查指定的属性是否具有特定值。默认情况下，属性必须存在于{@link Environment}中，并且不等于{@code false}。 {@link #havingValue （）}和{@link #matchIfMissing （）}属性允许进一步自定义
属性
havingValue
可name与组合使用
比较name获取到的属性值与havingValue给定的值是否相同
相同才加载配置
matchIfMissing
默认是false
缺少该property时是否可以加载。
如果为true，没有该property也会正常加载

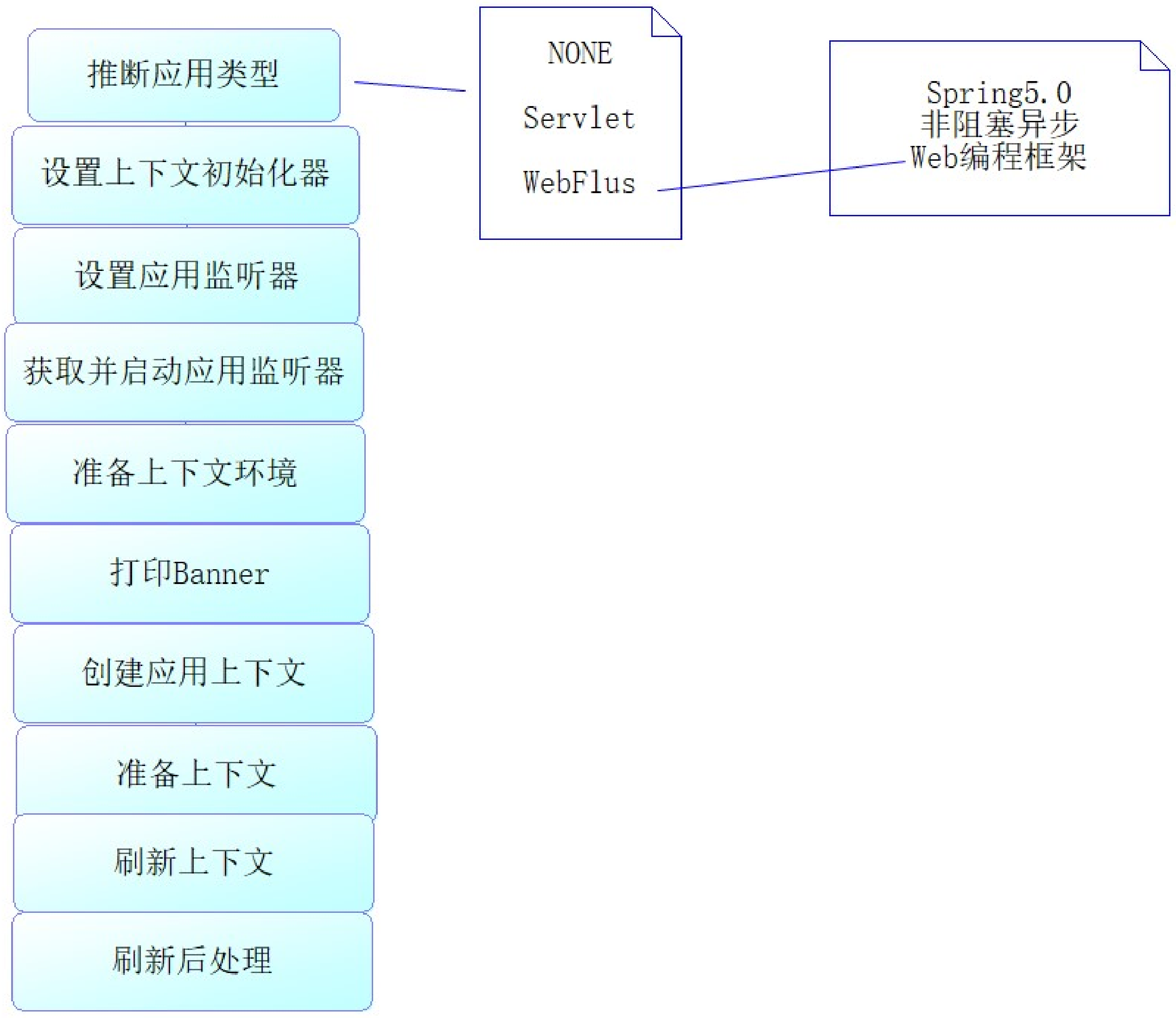
```
* 案例一（Conditional）
 * 需求：根据当前操作系统来注入User实例，
        windows下注入xiaohei，linux下注入xiaobai
* 代码
public abstract class BaseCondition implements Condition {
    @Override
    public boolean matches(ConditionContext context, AnnotatedTypeMetadata meta
        // 获得运行的环境
        Environment environment = context.getEnvironment();
        // 获得运行的系统
        String property = environment.getProperty("os.name");
        return property.contains(getPlatformName());
    }
```

```java
    protected abstract String getPlatformName();
}
public class WindowCondition extends BaseCondition {

    @Override
    protected String getPlatformName() {
        return "Windows";
    }
}
public class LinuxCondition extends BaseCondition {
    @Override
    protected String getPlatformName() {
        return "Linux";
    }
}

@Configuration
public class WebConfig {
    @Conditional(WindowCondition.class)
    @Bean
    public User user1(){
        //-ea
        User user=new User();
        user.setId(1);
        user.setUsername("xiaohei");
        user.setSex("男");
        user.setPsw("123");
        return user;
    }
    @Conditional(LinuxCondition.class)
    @Bean
    public User user2(){
        User user=new User();
        user.setId(2);
        user.setUsername("xiaobai");
        user.setSex("女");
        user.setPsw("456");
        return user;
    }
}
```

```
* 测试
AnnotationConfigApplicationContext context
=new AnnotationConfigApplicationContext(WebConfig.class);
@Test
public void test6(){
  Map<String, User> map = context.getBeansOfType(User.class);
  System.out.println(map);
}
 * 当注入的bean不标注解，都会放到容器了
 * 当使用注解时会根据运行的环境放到不同的bean到容器里
 * 可以通过修改VM-options:-Dos.name=Linux,模拟环境
案例二：（多个条件）
public class ACondition implements Condition {
    @Override
    public boolean matches(ConditionContext context,
    AnnotatedTypeMetadata metadata) {
        return false;
    }
}
public class BCondition implements Condition {
    @Override
    public boolean matches(ConditionContext context,
        AnnotatedTypeMetadata metadata) {
        return true;
    }
}
@Conditional({ACondition.class, BCondition.class})
@Bean
public User user3(){
  User user=new User();
  user.setId(3);
  user.setUsername("xiaoming");
  user.setSex("女");
  user.setPsw("456");
  return user;
}
* 测试：必须两个条件都满足，才把user3放到容器里
```

* 能够掌握SpringBoot启动流程

* SpringBoot 启动流程图



```
* 启动流程源码分析
 * 通过断点调试方式分析
  * SpringApplication
    * 构造器
    * run方法
```