链滴

如何解释 spring 是什么

作者：thas

原文链接：https://ld246.com/article/1591596491519

来源网站：链滴

什么是spring

spring 一般特指 spring-framework. spring 是一个便捷的开发快捷框架，它为企业和个人应用开发供了基础的框架支持。

官方源码仓库

什么是开发框架?

框架是软件项目的架子, 开发框架一般具有支撑性, 约束性, 扩展性.

IT语境中的框架, 特指为解决一个开放性问题而设计的具有一定约束性的支撑结构. 在此结构上可以根具体问题扩展, 安插更多的组成部分, 从而更迅速和方便地构建完整的解决问题的方案.

spring为什么非常流行

一个软件项目总不可避免的要遇上这个几点:

1. 异常和运行日志. 这是最重要也最容易被忽略的, 生产环境并不像开发调试一下, 啥东西都能打印在制台上，也并不是什么bug都能够被复现的, 这时候只能通过日志来定位问题.

spring 最基础的包便是 spring-jcl, 这个包提供了最基本的日志框架(把apache的整个 commons-logging 源码 copy 了过来，其中还集成了对 jul, log4j 和 slf4j 的支持), 但这种模式仅给spring自身使用.

spring-boot 的日志使用的是 slf4j + logback 的模式. 将juc, jcl, jboss-logging 三种日志框架统一换为 slf4j(*-over-slf4j)，最后再使用 logback 来实现 slf4j 的 api，实现统一输出日志。项目pom文中应能看到如下依赖:

```
<dependency>
      <groupId>ch.qos.logback</groupId>
      <artifactId>logback-classic</artifactId>
   </dependency>
   <dependency>
      <groupId>ch.qos.logback</groupId>
      <artifactId>logback-core</artifactId>
   </dependency>
  <!-- log4j通过slf4j来代理 -->
   <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>log4j-over-slf4j</artifactId>
   </dependency>
   <!-- apache commons logging通过slf4j来代理 -->
   <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>jcl-over-slf4j</artifactId>
   </dependency>
   <!-- java.util.logging 通过slf4j来代理 -->
   <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>jul-to-slf4j</artifactId>
   </dependency>
```

java中的异常分为编译时异常, 运行时异常, 前者在java届是争议颇多. 在 spring 框架中将全部的编译异常包装为了运行时异常, 同时 spring 会进行捕获, 确保异常不会被一直向上抛到 jvm, 导致程序终止.

一般会使用拦截器/AOP在目标方法前后进行捕获，然后记录异常，将错误信息转为正常 (httpStatus ode:200) 的输出。在引入了切面编程后, 异常的处理方就变成了本方法, 调用方以及切面. 编译时异常求异常必须由本方法或调用方处理(不会交给切面), 所以并不适用于切面编程.

2. 数据存储及事务. spring-jdbc 对 jdbc 进行了封装 (jdbcTemplate), 当然现在已经很少直接使了. spring-tx 提供了事务支持(将 Connection 对象绑定到线程上, ORM 框架就是根据这个特性支持了pring框架的事务).

3. 高内聚低耦合. spring最受欢迎的原因就是因为这一点, IOC 也好, AOP 也好, 都是为了解耦.

IOC (Inversion Of Control), 控制翻转, 即将创建对象的权利交给 IOC, 用户无须在业务代码中手动去ew 对象, 且无须关注这个对象本身的实现, 只需要关注它所支持的接口即可. 这样主程序只依赖了抽象依赖倒置原则), 即使实现发生了变化, 业务无须做修改.

早期的避免业务耦合具体实现的方式便是工厂模式, 通过工厂去创建对象, 再将对象交给主程序，无须系如何new出对象. IOC 则更高级, 它本质就是将是将所有的工厂管理起来, 用户使用 BeanFactory 用BanName 或者类型 (策略模式) 就可以自动创建并获得实例。

IOC 是一种设计思想, DI/CI 是实现方式, Spring 对这两种方式都做了实现.

DL (DependencyLookup), 依赖查找, 主动到 IOC 容器中寻得依赖, 依赖 IOC 容器提供的 API.

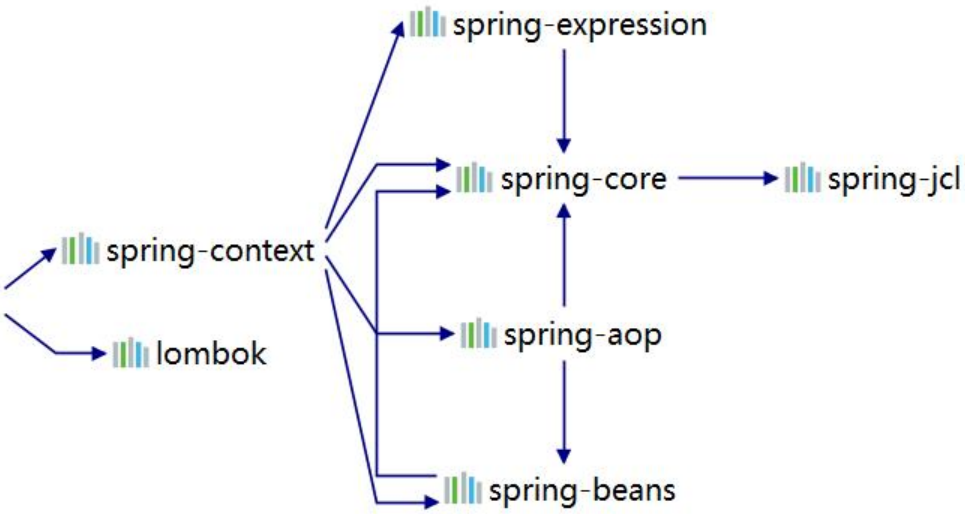
DI (Dependency Injection), 依赖注入. IOC 中实例创建时(构造器注入)或创建后(属性和setter注入)动注入依赖, 无需在业务代码中手动去依赖查找.

Spring 还提供了另一种方式: 接口回调注入, 这种方式并不常用, 会造成 Spring 对业务代码的重度侵入. 这种方式可以用来获得 Spring 的内建依赖 (游离对象, 如 BeanFactory, ApplicationContext) 或 Bean 的信息. Spring 中的内建依赖不能被依赖查找, 但是可以被依赖注入. 而接口回调可以在 Spring 启动期的时候确保业务对象能够获得内建依赖 (依赖注入 (@Autowire 注解) 必须在 BeanPostProcessor 始化完成后才能实现).

AOP 切面编程也是为了实现高内聚低耦合. 将业务流程中的主要业务和次要业务 (不影响业务流程的) 分，核心业务流程只包含主要业务, 次要业务放入切面中. 切面编程可以用来实现纵向业务的代码复用继承是横向/同级别业务的代码复用).

4. 可扩展性. spring本身就考虑了第三方的集成，比如mybatis、hibernate等数据库访问框架，及struts等web框架。最能体现扩展性还是 spring-boot.

spring组成



spring 最常用的包便是 spring-context 包，但是其实它还依赖另外5个包 spring-jcl, spring-core, spring-expression, spring-beans, spring-aop.

1. spring-jcl, spring 的日志框架, copy 自 Apache 的 commons-logging.

2. spring-core, spring 的核心类库, 不包含具体的业务实现, 主要包括 annotation 注解 , 各种常量, codec 编解码, convert 类型转换, env 环境配置, io 操作, log 日志, serializer 序列化工具 (使用 java 自的), style 字符串输出美化, task (定时/异步/同步) 任务调度, type (Java 反射类型).

3. spring-expression, spring自己实现的 EL 表达式解析工具, 与之类似的还有 JEL.

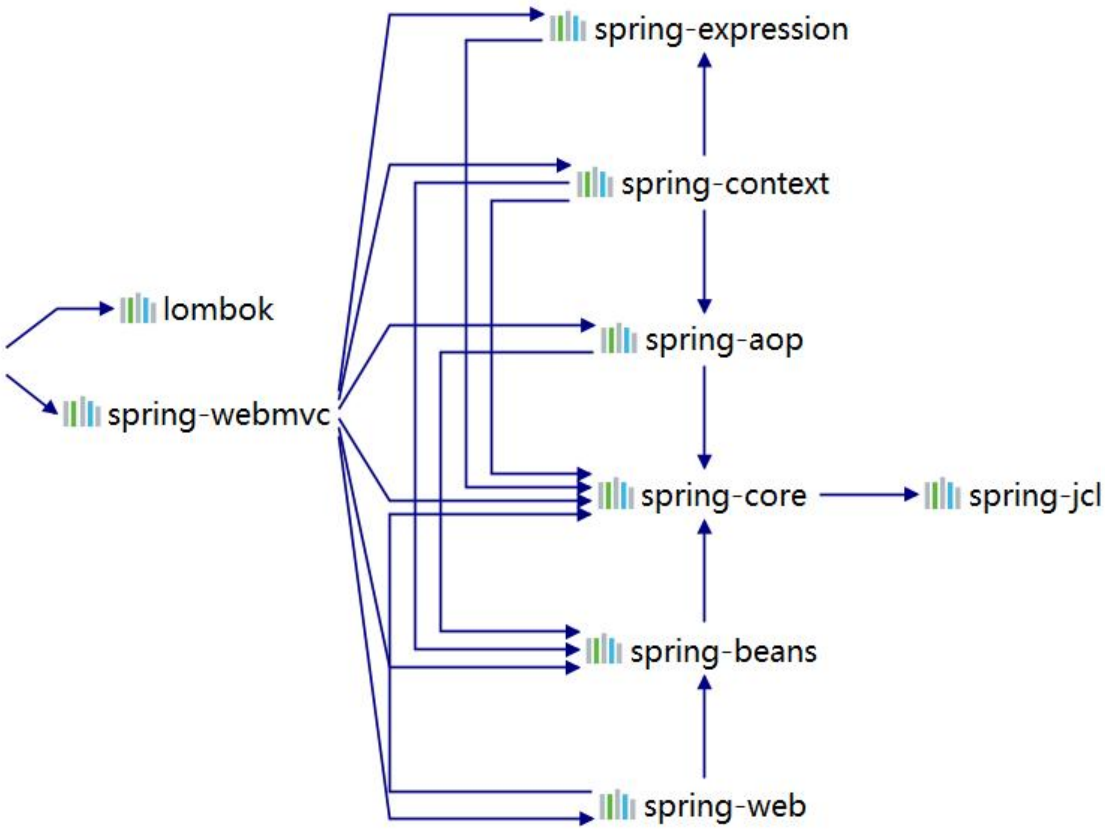
4. spring-aop , 切面相关的工具.

5. spring-beans, 所有跟 Bean相关的, IOC 容器和DL/DI的实现都在这里面. 如果只是想使用 IOC 容这个功能, 只依赖这个包就足够了 (得手动注册 BeanDefinition).

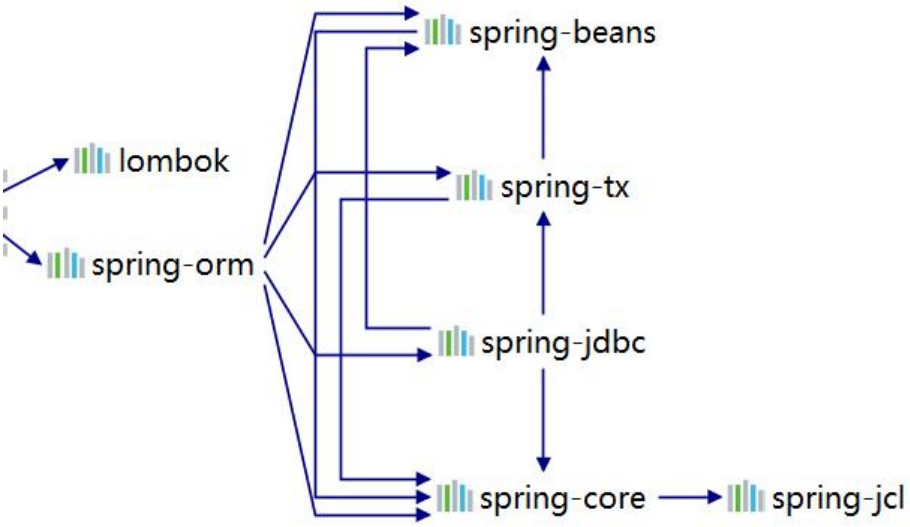
6. spring-context, 最常用的一个包了. context 是上下文的意思，可以理解为贯穿某个生命周期的一变量. spring-context 就是一个辅助开发的工具人, 还没上升到项目架构的高度. 如果想不仅仅满足于本的 IOC容器, 还需要使用注解注册 Bean, 条件注册 Bean, 包扫描, xml 注册 Bean 这些更方便的功能, 那么就要依赖 spring-context 这个包了. 除了容器相关的, spring-context 还提供了事件, 国际化, 校验 外部化配置, 环境配置, EL 解析, 异步任务调度相关的工具.

其他还有:

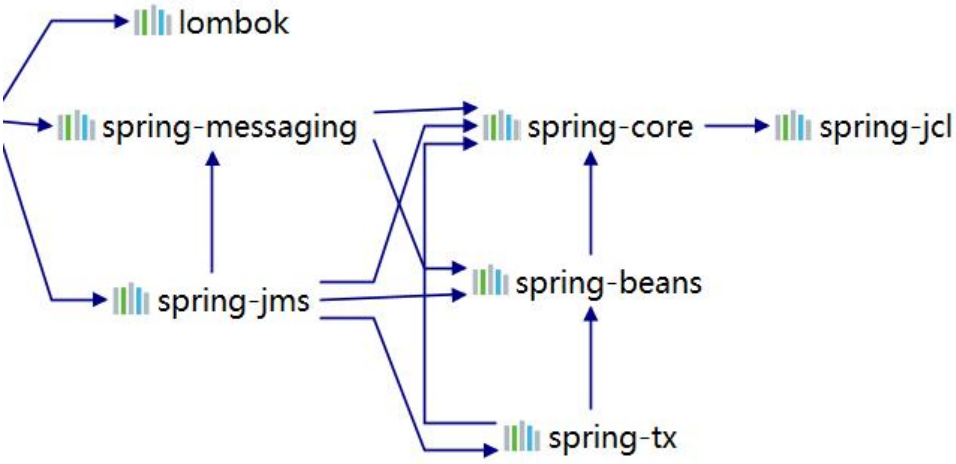
1. web 相关的



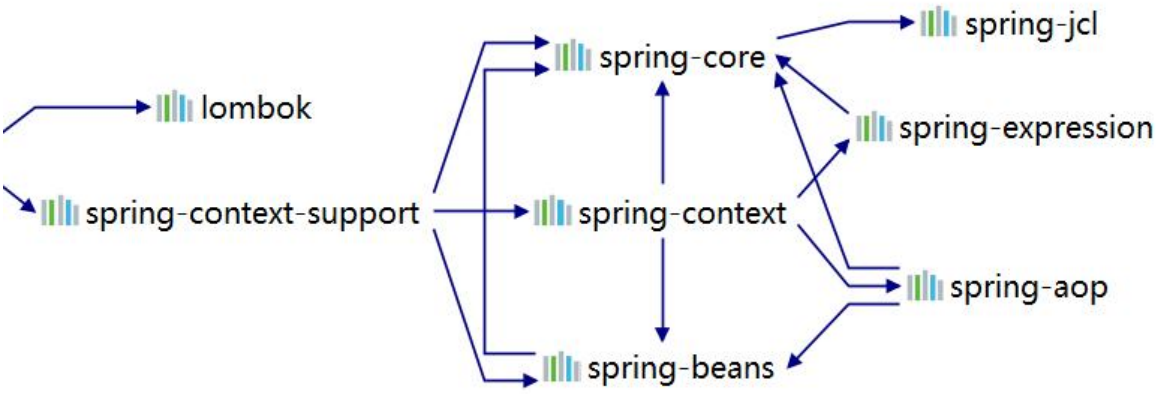
2. orm 相关的



3. jms 相关的



4. spring-context 增强, 缓存, 邮件, 任务调度, freemarker模板引擎



原文链接：如何解释 spring 是什么