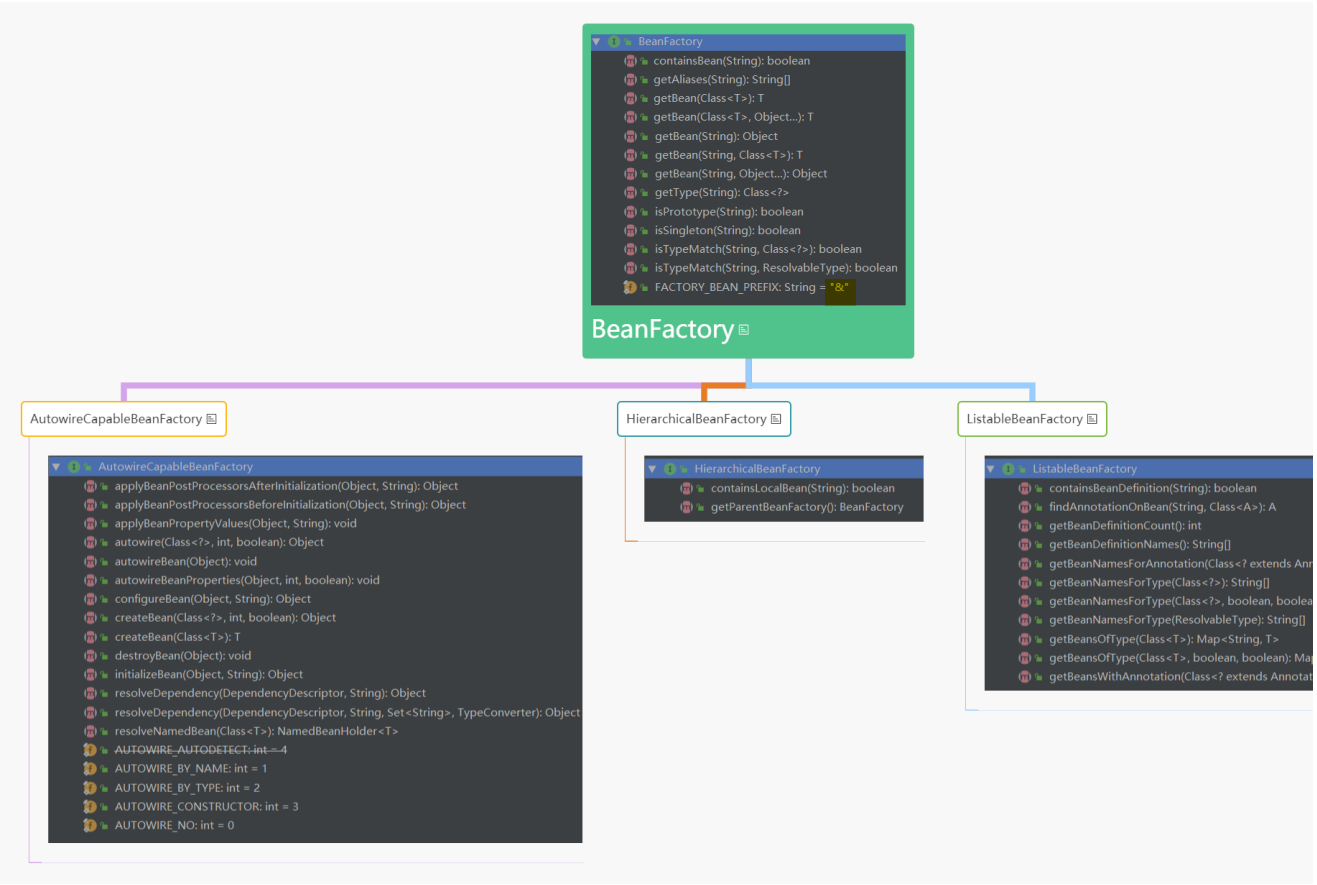
Spring核心API介绍

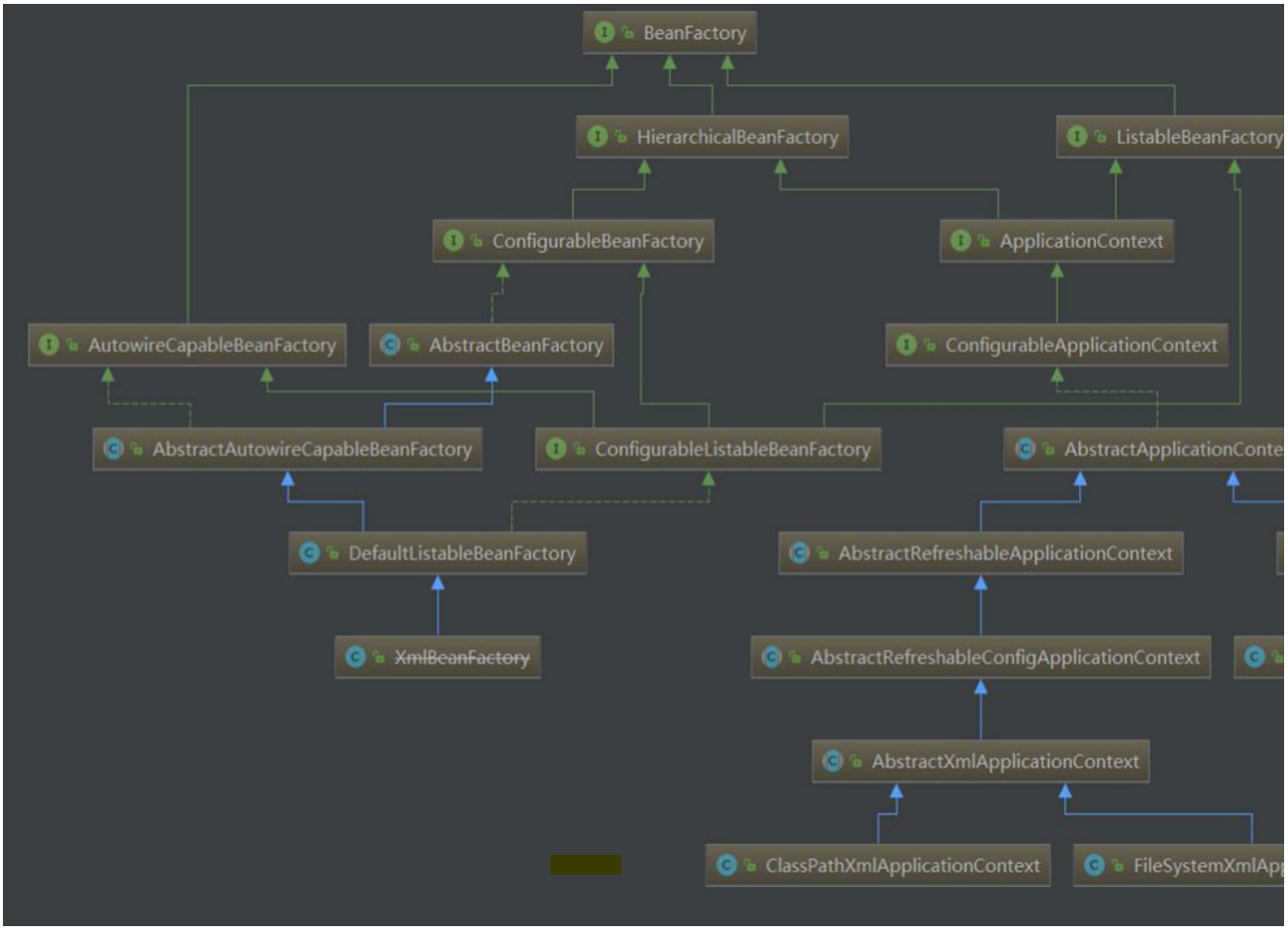
BeanFactory

> org.springframework.beans.factory

BeanFactory是用于访问Spring Bean容器的根接口，典型的工厂模式，用于生产Bean的一个Bean工厂，其提供了生产Bean所需的最基本规则。

BeanFactory的所有方法：





BeanDefinition

功能

BeanDefinition是bean在spring中的描述，有了BeanDefinition我们就可以创建Bean,BeanDefinition是Bean在spring中的定义形态 接下来我们看看BeanDefinition的相关接口与类.

方法

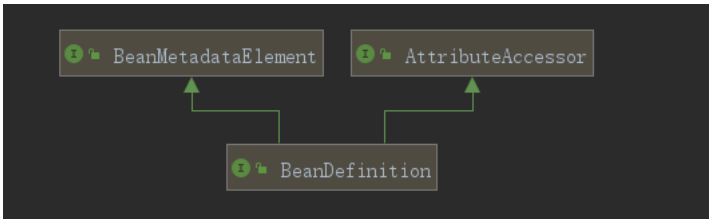
- `String: getBeanClassName`: 返回当前bean definition定义的类名
- `ConstructorArgumentValues: getConstructorArgumentValues`:返回bean的构造函数参数
- `String[]: getDependsOn`:返回当前bean所依赖的其他bean的名称
- `String: getFactoryBeanName`: 返回factory bean的名称
- `String: getFactoryMethodName`: 返回工厂方法的名称
- `BeanDefinition: getOriginatingBeanDefinition`: 返回原始的BeanDefinition,如果不存在返回`null`
- `String: getParentName`: 返回当前bean definition的父definition的名字
- `MutablePropertyValues: getPropertyValues`: 返回一个用于新的bean实例上的属性值
- `String: getScope`: 返回当前bean的目标范围
- `boolean: isAbstract`: 当前bean是否是abstract,意味着不能被实例化
- `boolean: isLazyInit`: bean是否是延迟初始化
- `boolean: isPrimary`: bean是否为自动装配的主要候选bean
- `boolean: isPrototype`: bean是否是多实例
- `boolean: isSingleton`: bean是否是单例
- `void: setAutowiredCandidate(boolean)`: 设置bean是否对其他bean是自动装配的候选bean
- `void: setBeanClassName(String)`: 指定bean definition的类名
- `void: setDependsOn(String ...)`: 设置当前bean初始化所依赖的beans的名称
- `void: setFactoryBeanName(String)`: 如果factory bean的名称
- `void: setFactoryMethodName(String)`: 设置工厂的方法名
- `void: setLazyInit(boolean lazyInit)`: 设置是否延迟初始化
- `void: setParentName(String)`: 设置父definition的名称

- void: setPrimary(boolean): 设置是否主要的候选bean
- void: setScope(String): 设置bean的范围,如:单例,多实例

定义

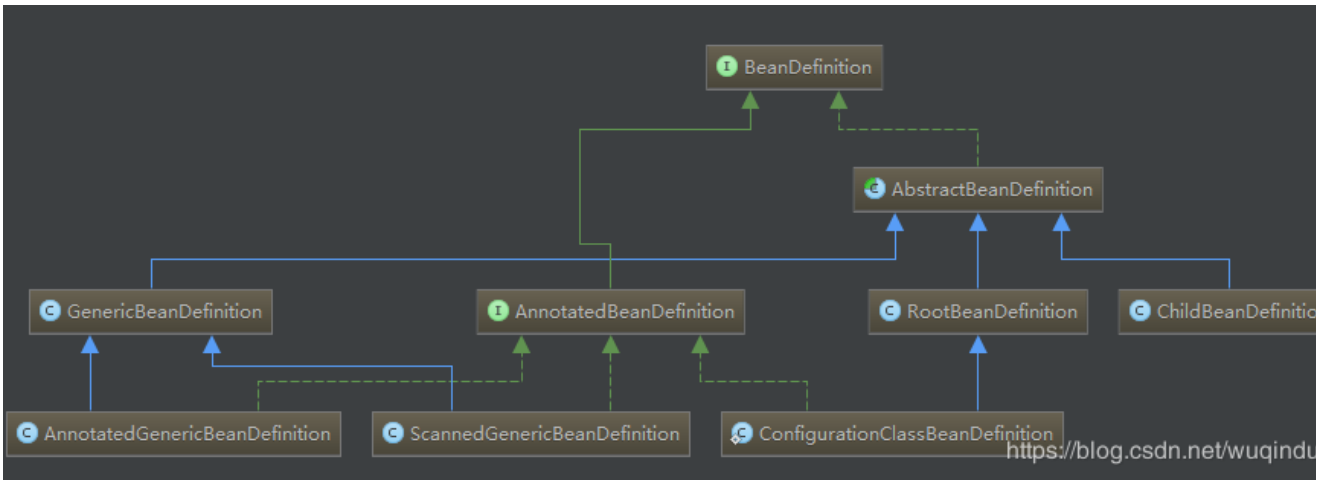
- **BeanDefinition接口**: 顶级基础接口,用来描述Bean,里面存放Bean元数据，比如Bean类名、scope、属性、构造函数参数列表、依赖的bean、是否是单例类、是否是懒加载等一些列信息。

向上



- **BeanMetadataElement接口**：BeanDefinition元数据，返回该Bean的来源
- **AttributeAccessor接口**：提供对BeanDefinition属性操作能力，

向下



- **AbstractBeanDefinition类**：抽象类统一实现了BeanDefinition定义的一部分操作，可以说是定义了BeanDefinition很多默认的属性。 正是在AbstractBeanDefinition基础上， Spring衍生出了一些列BeaDefinition。

这里我们可以关注下重写的equals(),hashcode(), toString()方法

此外initMethodName属性，destroyMethodName 属性， 这两个属性bean的生命周期有关，此处只提一句，后续讲解。

接下来。我们看看从AbstractBeanDefinition上衍生出来的几个类

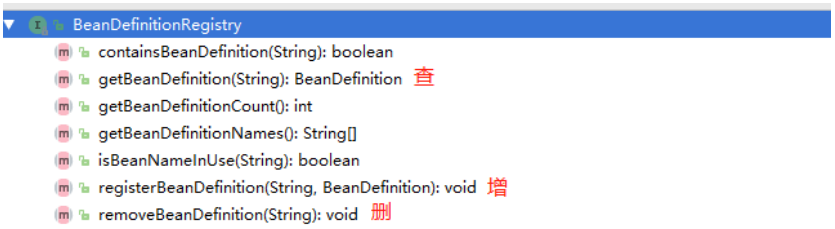
- **RootBeanDefinition**: 代表一个xml,java Config来的BeanDefinition
- **ChildBeanDefinition**: 从Spring2.5开始，ChildBeanDefinition已经不再使用，取而代之的是GenericBeanDefinition。
- **GenericBeanDefinition:** spring2.5后注册bean首选的是GenericBeanDefinition。GenericBeanDefinition允许动态的设置父bean.GenericBeanDefinition可以作为RootBeanDefinition与ChildBeanDefinition的替代品。
- **AnnotatedBeanDefinition接口**： 表示注解类型BeanDefinition。有两个重要的属性，AnnotationMetadata，MethodMetadata分别表示BeanDefinition的注解元信息和方法元信息 实现了此接口的BeanDefinition可以获取到注解元数据和方法元数据。
- **AnnotatedGenericBeanDefinition类**: 表示@Configuration注解注释的BeanDefinition类
- **ScannedGenericBeanDefinition类**: 表示@Component、@Service、@Controller等注解注释的Bean类

操作

动作也可分为两种： 一种是针对自身的操作: 自提提供给外部的可以操作其本身的动作 另一种是外部对BeanDefinition的操作

- **BeanDefinitionRegistry接口**：具有增，查，删BeanDefinition的能力。一次只能注册一个BeanDefinition.

实现类SimpleBeanDefinitionRegistry，DefaultListableBeanFactory，GenericApplicationContext等 一般实现类里都都有一个 private final Map<String, BeanDefinition> beanDefinitionMap = new ConcurrentHashMap()来存储BeanDefinition.



- **BeanDefinitionReader接口**： 既可以使用BeanDefinitionRegistry构造。也可以通过loadBeanDefinitions把配置加载为多个BeanDefinition并注册到BeanDefinitionRegistry中。 可以说是高效版本的BeanDefinitionRegistry. 实现类有 XmlBeanDefinitionReader从xml中读取BeanDefinition；PropertiesBeanDefinitionReader从Properties文件读取BeanDefinition
- **AnnotatedBeanDefinitionReader类** 对带有注解的BeanDefinition进行注册
- **ClassPathBeanDefinitionScanner类**： 可以扫描到@Component @Repository @Service @Controller的BeanDefinition注册到容器中。

其他形态

- BeanDefinitionHolder： BeanDefinition包装类。

Bean

Bean是我们需要的对象，是我们从spring内得到的结果，也就是对象实例

定义

从定义层面看.Bean其实就是我们需要的对象.

操作

我们来看看Bean在spring有哪些操作相关的接口或类。

- SingletonBeanRegistry接口：与BeanDefinition的注册相应的。Bean的操作也有一个类似的接口来操作Bean.SingletonBeanRegistry接口提供了对Bean的注册，获取，存在性判断等功能。
- InitializingBean:对于实现 InitializingBean的Bean，它将执行 afterPropertiesSet(); 在所有的 bean 属性被设置之后。
- InstantiationStrategy:提供 Bean实例化的策略接口，
- DisposableBean:对于 实现了DisposableBean的Bean ， 它将运行 destroy()在 Spring 容器释放该 bean 之后
- FactoryBean: 生产Bean的Bean.

其他形态

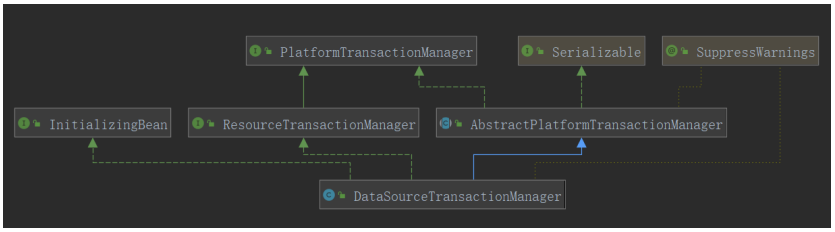
- BeanWrapper: 对Bean的包装.BeanWrapper可以看作是一个从 BeanDefinition 到 Bean 过程中间的产物，可以称为”低级 bean “，在一般情况下，我们不会在实际项目中用到它。BeanWrapper 是 Spring 框架中重要的组件类，它就相当于一个代理类，Spring 委托 BeanWrapper 完成 Bean 属性的填充工作。在 bean 实例被 InstantiatioonStrategy 创建出来后，Spring 容器会将 Bean 实例通过 BeanWrapper 包裹起来，是通过 BeanWrapper.setWrappedInstance() 完成的

总结：

BeanDefinition是物料，Bean是成品，理解BeanDefinition与Bean的关系是理解spring的基础。

事务

DataSourceTransactionManager:



文档：Spring核心API介绍.note

链接：http://note.youdao.com/noteshare?id=c9bde74fde820328f08b053548d5d3e8&sub=66DF3F42D04A4D1CA9FB6729AA0D7639