[**spring beans源码解读之--总结篇**](https://www.cnblogs.com/davidwang456/p/4213652.html)

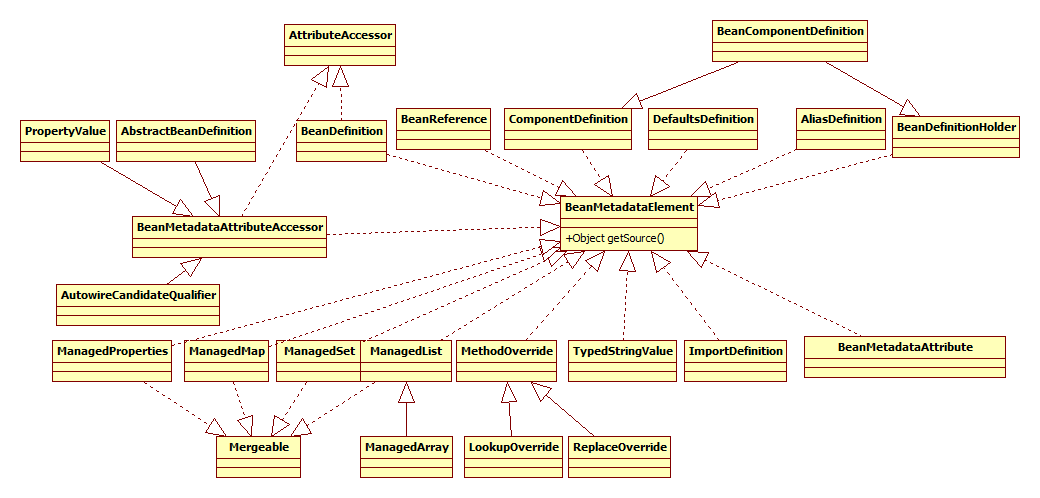
spring beans下面有如下**源文件包**：

org.springframework.beans, 包含了操作java bean的接口和类。  
org.springframework.beans.annotation, 支持包，提供对java 5注解处理bean样式的支持。  
org.springframework.beans.factory, 实现spring轻量级IoC容器的核心包。  
org.springframework.beans.factory.access, 定位和获取bean工程的辅助工具类。  
org.springframework.beans.factory.access.el,从统一样式的EL 获取spring beanFactory的支持类  
org.springframework.beans.factory.annotation, 配置基于注解驱动的bean的支持包。  
org.springframework.beans.factory.config, bean工厂的SPI接口和配置相关的处理类。  
org.springframework.beans.factory.parsing, bean definition解析的支持基础类  
org.springframework.beans.factory.serviceloader, jdk1.6 ServiceLoader基础类的支持包。  
org.springframework.beans.factory.support,org.springframework.beans.factory包的支持类  
org.springframework.beans.factory.wiring, 一种决定一个bean实例的元数据的机制。  
org.springframework.beans.factory.xml, 包含了一个基于xml的beanFactory实现，也包含一个标准的spring-beans的dtd  
org.springframework.beans.propertyeditors, 属性编辑器，用来将string类型的值转换为object类型，例如：java.util.Properties  
org.springframework.beans.support,org.springframework.beans的支持包，像：一组bean的排序和保持工具类等。

1.org.springframework.beans包含的类和接口有：

其中**接口**有：

1.1 BeanInfoFactory：创建java.beans.BeanInfo实例的策略。  
1.2 BeanMetadataElement：传送配置源对象的超级接口。定义了唯一的方法：Object getSource();其子类和子接口如下：



1.3 PropertyAccessor：获取和设置属性(例如对象bean的属性，对象的Field)的通用接口，它用来作为BeanWrapper的基础接口。主要方法有：

Class getPropertyType(String propertyName)

TypeDescriptor getPropertyTypeDescriptor(String propertyName)

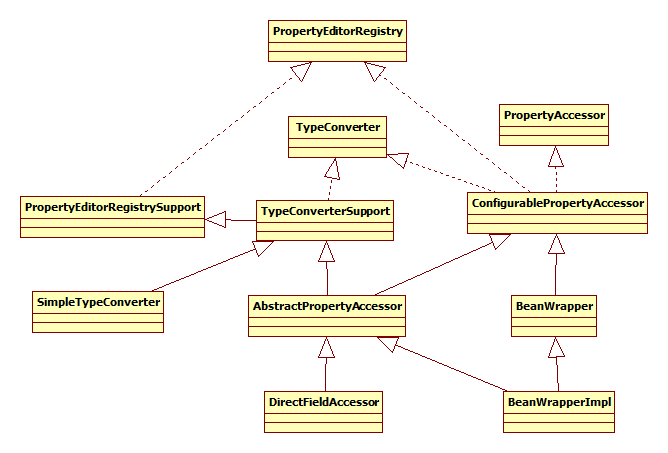
Object getPropertyValue(String propertyName)

boolean isReadableProperty(String propertyName)

boolean isWritableProperty(String propertyName)

void setPropertyValue(PropertyValue pv)

 继承关系：



1.4 BeanWrapper： spring 底层JavaBean公共基础的核心接口。  
1.5 ConfigurablePropertyAccessor： PropertyAccessor配置方法的封装接口。  
1.6 Mergeable 代表一个对象的值组可以和父对象的值组合并的接口。其主要子类有：anagedList, MManagedArray, ManagedMap, ManagedProperties, ManagedSet。为了加深对这个接口的理解，我们以ManagedList为例，看一下它合并的实现方法：

public List<E> merge(Object parent) {

List<E> merged = new ManagedList<E>();

merged.addAll((List<E>) parent);

merged.addAll(this);

return merged;

}

1.7 PropertyEditorRegistrar： 用一个属性编辑注册器(PropertyEditorRegistry)注册一个自定义的属性编辑器(PropertyEditor)的策略接口。

1.8 PropertyEditorRegistry： 封装了javabean属性编辑器的注册方法。  
1.9 PropertyValues：包含了一个或者多个PropertyValue对象，通常用作特定的一个目的bean的属性更新，主要方法是：PropertyValues changesSince(PropertyValues old);  
1.10 TypeConverter：定义了类型转换方法的接口。

 其中**类**有：

1.11  AbstractPropertyAccessor： PropertyAccessor接口的抽象实现.

1.12 BeanMetadataAttribute：bean definition定义中属性键值对容器.   
1.13 BeanMetadataAttributeAccessor：AttributeAccessorSupport的扩展,为了追踪对象定义源 holding attributes as BeanMetadataAttribute objects in order to keep track of the definition source.   
1.14 BeanUtils： javabean的静态工具方法：用来初始化bean、检查bean的属性，复制bean的属性等等。  
1.15 BeanWrapperImpl：BeanWrapper的默认实现，能够适用于绝大多数应用场合。  
1.16 CachedIntrospectionResults：.内部类，缓存一个类的javabean的属性描述信息。使用情境：BeanWrapperImpl.java

public PropertyDescriptor[] getPropertyDescriptors() {

return getCachedIntrospectionResults().getPropertyDescriptors();

}

1.17 DirectFieldAccessor PropertyAccessor的实现类，直接获取实例的field。   
1.18 ExtendedBeanInfoFactory BeanInfoFactory接口实现，评估bean的class是否有不满足javaBean规范的set方法，因而是否满足spring ExtendedBeanInfo作为内省的候选。  
1.19 MutablePropertyValues Default implementation of the PropertyValues interface.

1.20 PropertyAccessorFactory：获取PropertyAccessor实例的简单工厂门面, 特别是获取BeanWrapper实例。PropertyAccessorFactory.java

public static BeanWrapper forBeanPropertyAccess(Object target) {

return new BeanWrapperImpl(target);

}

1.21 PropertyAccessorUtils：工具方法或者工具类，根据PropertyAccessor 获取bean属性。  
1.22 PropertyEditorRegistrySupport：PropertyEditorRegistry接口的基本实现类。.  
1.23 PropertyValue：一个对象，它保存一个bean的单独属性的值信息。  
1.24 PropertyValuesEditor Editor for a PropertyValues object.   
1.25 SimpleTypeConverter TypeConverter接口的简单实现类，它不会针对一个特定的目标对象进行操作。   
1.26 TypeConverterSupport TypeConverter接口的基本实现类,使用包私有的代理

**异常处理类**：

BeanInstantiationException： bean初始化失败时抛出的异常。  
BeansException：在beans包和子包中所抛出的所有异常的抽象超类。  
ConversionNotSupportedException：当一个bean的属性没有适合的编辑器或者转换器时所抛出的异常。   
FatalBeanException：在bean包和子包中遇到的不可恢复问题抛出的异常，例如：当引用一个无效的bean属性时抛出的InvalidPropertyException异常。   
MethodInvocationException：当一个bean属性的get和set方法抛出异常时抛出该异常，类似于InvocationTargetException。  
NotReadablePropertyException：当试图获取一个不可读属性的属性值时抛出的异常，典型场景bean属性没有get方法。   
NotWritablePropertyException：当试图向一个不可写属性写入属性值时抛出的异常，典型场景bean属性没有set方法。 .   
NullValueInNestedPathException：当获取一个内嵌属性路径的属性时遇到空指针异常时抛出的异常。  
PropertyAccessException：获取属性相关的异常超类，例如类型不匹配或者目标调用异常。   
PropertyBatchUpdateException组合异常，由多个的单个PropertyAccessException实例组成。  
TypeMismatchException：当试图设置bean属性时抛出的类型不匹配异常。

2. org.springframework.beans.annotation包含的类有：

AnnotationBeanUtils：javabean类型注解的通用工具方法。方法有：

public static void copyPropertiesToBean()

静态方法，拷贝指定Annotation的属性到给定的目标bean。

3. org.springframework.beans.propertyeditors属性编辑器，用来将string类型的值转换为object类型，例如：java.util.Properties。包含的类有：

ByteArrayPropertyEditor.java,CharacterEditor.java,CharArrayPropertyEditor.java,CharsetEditor.java,ClassArrayEditor.java,ClassEditor.java,CurrencyEditor.java,CustomBooleanEditor.java,CustomCollectionEditor.java,CustomDateEditor.java,CustomMapEditor.java,CustomNumberEditor.java,FileEditor.java,InputSourceEditor.java,InputStreamEditor.java,LocaleEditor.java,PatternEditor.java,PropertiesEditor.java,ResourceBundleEditor.java,StringArrayPropertyEditor.java,StringTrimmerEditor.java,TimeZoneEditor.java,URIEditor.java,URLEditor.java,UUIDEditor.java,ZoneIdEditor.java  
它们的共同特点是实现了PropertyEditorSupport，实现了setAsText，getAsText

4. org.springframework.beans.support包含的类有：

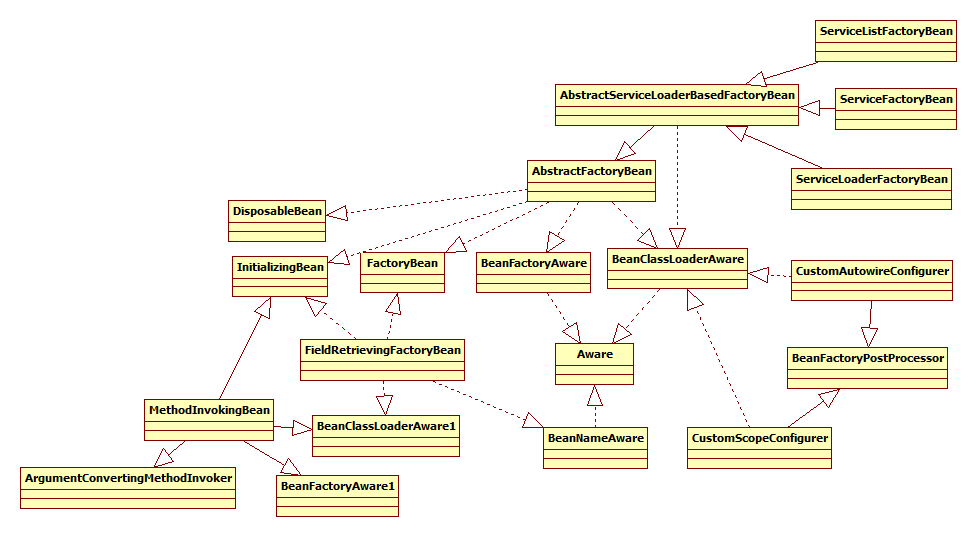
ArgumentConvertingMethodInvoker：MethodInvoker的子类，通过TypeConverter将真实目标方法的指定参数进行转换。   
MutableSortDefinition：SortDefinition接口的可变实现。  
PagedListHolder<E>：PagedListHolder是一个简单的状态保持，它处理一组对象，将它们分页。   
PropertyComparator：比较对象为两个bean，通过BeanWrapper来比较指定bean的属性。   
ResourceEditorRegistrar：继承了PropertyEditorRegistrar，使用资源编辑器来填充指定的PropertyEditorRegistry。

5.org.springframework.beans.factory包含类有：

其中**接口**有：

Aware：它是一个标签，内部没有任何属性或者方法。它是一个超级接口，实现它的子接口的bean标志着在spring容器中可以被特定框架的对象通知到，通过回调方式的方法来完成这种通知。

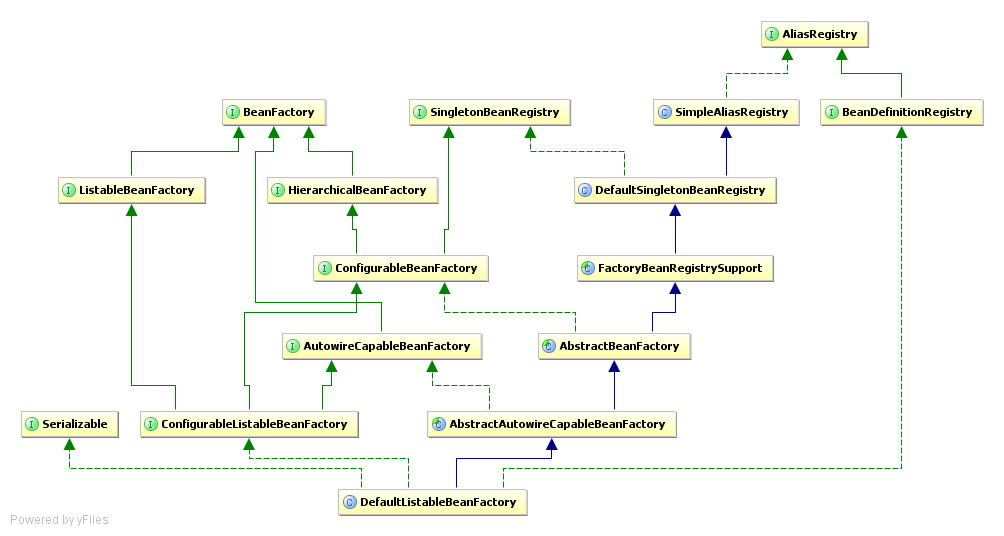
继承关系：



BeanClassLoaderAware 允许一个获取它的classLoader(即当前bean factory加载bean类使用的class loader)的回调类，实现了void setBeanClassLoader(ClassLoader classLoader);方法

void setBeanClassLoader(ClassLoader classLoader);

BeanFactory：获取spring bean容器的根接口。



    (图片来源：http://www.myexception.cn/software-architecture-design/925888.html)

BeanFactoryAware实现此接口的bean可以获取到它们自己的Beanfactory。

void setBeanFactory(BeanFactory beanFactory) throws BeansException;

BeanNameAware实现此接口的bean可以在一个bean factory中获取到它们的bean名称。

void setBeanName(String name);

DisposableBean：实现此接口的bean希望在销毁方法中释放资源。

void destroy() throws Exception;

FactoryBean<T>：实现了此接口的bean不能看做一个通常意义上的bean，一个FactoryBean虽然以bean的形式来定义，但它暴露的对象(getObject())通常是它创建的对象。

T getObject() throws Exception;

Class<?> getObjectType();

boolean isSingleton();

HierarchicalBeanFactory：子接口，实现此接口的bean factory具有层次结构即可以获取父BeanFactory。

BeanFactory getParentBeanFactory();

boolean containsLocalBean(String name);

InitializingBean：实现此接口的bean在BeanFactory设置为它们的属性时只需要执行一次，例如：初始化定制或者仅仅检查必须存在的属性是否已经设置完成等等。

void afterPropertiesSet() throws Exception;

ListableBeanFactory：beanFactory接口的实现，实现此接口的beanFactory能遍历他们内部的所有bean实例，而不用根据客户请求通过名字一个一个的去搜索bean。

String[] getBeanDefinitionNames();

String[] getBeanNamesForType(Class<?> type);

NamedBean：对应BeanNameAware接口，返回bean的名称。

/\*\*

\* Return the name of this bean in a Spring bean factory.

\*/

String getBeanName();

ObjectFactory<T>：一个对象工厂，当触发时会返回所有对象的实例(可以是共享的或者独立的);

T getObject() throws BeansException;

SmartFactoryBean<T>：FactoryBean接口的扩展实现。

boolean isPrototype();

boolean isEagerInit();

 其中**类**有：  
BeanFactoryUtils：beanFactory上的操作方法工具类，特别是在ListableBeanFactory接口上。

**异常处理类**：

BeanCreationException：beanfactory在试图通过bean definition创建一个bean时遇到错误而抛出的异常。  
BeanCreationNotAllowedException：若当前不允许创建一个bean时(例如在beanFactory关闭过程中)而试图去获取bean的请求时抛出的异常。  
BeanCurrentlyInCreationException：一个bean的引用当前正在创建过程中抛出的异常。   
BeanDefinitionStoreException：beanFactory遇到一个无效的bean definition抛出的异常.   
BeanExpressionException：获取一个表达式的值失败时抛出的异常。  
BeanInitializationException：bean初始化异常时抛出  
BeanIsAbstractException：当试图获取一个定义为abstract的bean definition时抛出的异常。   
BeanIsNotAFactoryException：当一个bean不是工厂，但用户试图通过给定bean名称访问工厂。  
BeanNotOfRequiredTypeException：当一个bean的类型和期望类型不匹配时抛出的异常。   
CannotLoadBeanClassException：BeanFactory不能加载指定bean的class类时抛出的异常。  
FactoryBeanNotInitializedException：工厂bean没有初始化异常。  
NoSuchBeanDefinitionException：没有该bean definition异常  
NoUniqueBeanDefinitionException：返回多个bean definition异常。   
UnsatisfiedDependencyException：当有依赖检查时，在bean factory定义中，bean的依赖或者属性没有指定时抛出的异常

6.org.springframework.beans.factory.annotation

参见：<http://www.cnblogs.com/davidwang456/p/4199459.html>

其中接口有：

AnnotatedBeanDefinition：BeanDefinition接口的扩展，它暴露了它的bean 类的AnnotationMetadata，不需要加载类。

AnnotationMetadata getMetadata();

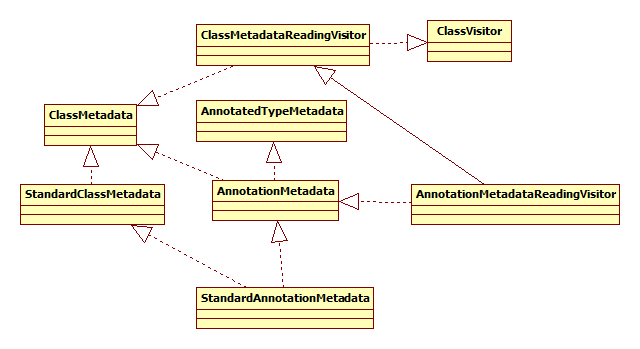
其中类有：  
AnnotatedGenericBeanDefinition：GernericBeanDefinition接口的扩展，通过暴露AnnotatedBeanDefinition接口来增加对注解元数据的支持。

public AnnotatedGenericBeanDefinition(Class<?> beanClass) {

setBeanClass(beanClass);

this.metadata = new StandardAnnotationMetadata(beanClass, true);

}



AnnotationBeanWiringInfoResolver继承自BeanWiringInfoResolver，使用配置的注解来查找哪些类需要自动绑定。实现了解析方法：

BeanWiringInfo resolveWiringInfo(Object beanInstance);

AutowiredAnnotationBeanPostProcessor:实现了BeanPostProcessor接口，它自动绑定注解的field，set方法和任意配置方法。

BeanFactoryAnnotationUtils：关联注解的bean的查询的工具方法，例如spring的@Qualifier注解。

public static <T> T qualifiedBeanOfType()

CustomAutowireConfigurer：继承了BeanFactoryPostProcessor，它使自定义的自动绑定qualifier类型的注册更便利。.   
InitDestroyAnnotationBeanPostProcessor继承了BeanPostProcessor，实现了注解触发init和destroy方法。

InjectionMetadata：管理注入元数据的内部类。

QualifierAnnotationAutowireCandidateResolver：实现了AutowireCandidateResolver，功能是将qualifier注解要自动绑定的field或者参数和bean definition qualifier相匹配。  
RequiredAnnotationBeanPostProcessor：实现了BeanPostProcessor，对配置了Required注解的javaBean属性进行强制检查。

枚举类：

Autowire：决定自动绑定状态的枚举，即一个bean的依赖是否由spring容器使用setter方式自动注入。这个是spring DI的核心概念。

注解类：

Autowired：public @interface Autowired：可以对成员变量、方法和构造函数进行标注，来完成自动装配的工作。  
Configurable：@Configurable 注解中的autowire属性就可以让Spring来自动装配  
Qualifier：@Qualifier(value = "限定标识符") 。

Required：public @interface Required 依赖检查；

Value：public @interface Value：用于注入SpEL表达式，可以放置在字段方法或参数上。

7. org.springframework.beans.factory.config包含的类有：

<http://www.cnblogs.com/davidwang456/p/4187012.html>

<http://www.cnblogs.com/davidwang456/p/4192318.html>

其中接口有：

AutowireCapableBeanFactory：beanFactory的扩展接口，实现了自动绑定功能。这个接口的两个主要方法是：

Object autowire(Class<?> beanClass, int autowireMode, boolean dependencyCheck) throws BeansException;

　　　void autowireBeanProperties(Object existingBean, int autowireMode, boolean dependencyCheck)throws BeansException;

autowire：使用给定的自动绑定策略，来给一个类的新实例进行初始化。  
autowireBeanProperties：使用名称或者类型来自动绑定给定bean实例的属性。

BeanDefinition；一个bean实例的描述，它含有属性值，构造参数值。它的实现子类还提供更多的信息。

BeanExpressionResolver：通过计算一个表达式来解析为值的策略接口。

Object evaluate(String value, BeanExpressionContext evalContext) throws BeansException;

BeanFactoryPostProcessor：允许对一个applicationContext中的bean definition进行定制修改，修改context内含的bean factory中bean的属性值。

void postProcessBeanFactory(ConfigurableListableBeanFactory beanFactory) throws BeansException;

BeanPostProcessor：允许对一个新的bean实例进行定制修改的工厂钩子。

Object postProcessBeforeInitialization(Object bean, String beanName) throws BeansException;

Object postProcessAfterInitialization(Object bean, String beanName) throws BeansException;

BeanReference：暴露了bean名称的引用接口。

String getBeanName();

ConfigurableBeanFactory：将会被大部分beanFactory实现的配置接口。

ConfigurableListableBeanFactory：将会被大部分ListablebeanFactory实现的配置接口。  
DestructionAwareBeanPostProcessor：BeanPostProcessor的子接口，它增加了一个销毁前回调方法。

void postProcessBeforeDestruction(Object bean, String beanName) throws BeansException;

InstantiationAwareBeanPostProcessor：BeanPostProcessor的子接口，它增加了一个初始化前回调方法，还有一个在初始化后但显式设置属性或者自动绑定发生前的回调方法。

Object postProcessBeforeInstantiation(Class<?> beanClass, String beanName) throws BeansException;

boolean postProcessAfterInstantiation(Object bean, String beanName) throws BeansException;

 PropertyValues postProcessPropertyValues(PropertyValues pvs, PropertyDescriptor[] pds, Object bean, String beanName)throws BeansException;

Scope：ConfigurableBeanFactory使用的策略接口，代表了bean实例所在的目标域。我们来从ConfigurableBeanFactory来看Scope的使用情况：

String SCOPE\_SINGLETON = "singleton";

String SCOPE\_PROTOTYPE = "prototype";

void registerScope(String scopeName, Scope scope);

String[] getRegisteredScopeNames();

Scope getRegisteredScope(String scopeName);

从上述代码中我们可以得知，默认有两种策略，singleton 单例即只有一个bean实例；prototype 原型即多个bean实例。如果想增加scope，可以通过registerScope来注册新的scope。

SingletonBeanRegistry：定义了共享bean实例的注册接口。

void registerSingleton(String beanName, Object singletonObject);

注意：prototype 实例是通过PrototypeAspectInstanceFactory来产生，而它由LazySingletonAspectInstanceFactoryDecorator来进行包装，保证仅返回一个aspect。

SmartInstantiationAwareBeanPostProcessor：InstantiationAwareBeanPostProcessor的扩展接口，它增加了对一个处理过bean的最终类型进行预测的回调方法。

Class<?> predictBeanType(Class<?> beanClass, String beanName) throws BeansException;

 其中类有：

AbstractFactoryBean<T>：实现了FactoryBean的简单模板超类，它根据标志来决定是创建一个单例还是一个prototype对象。  
BeanDefinitionHolder：使用名称或者别名来保存BeanDefinition。  
BeanDefinitionVisitor：遍历BeanDefinition对象的参观者类，特别是也遍历bean中的属性值和构造参数值，解析bean的元数据值。  
BeanExpressionContext：计算一个BeanDefinition内部的表达式的容器对象。  
BeanReferenceFactoryBean：过期。 它是一个通过不同名称来暴露一个bean对象的工厂bean。从spring 3.2以后，开始使用别名(从spring 2.5开始支持解析占位符)来替代它  
CommonsLogFactoryBean：过期。它是org.apache.commons.logging.Log实例的工厂bean，用来在多个bean对象中共享log实例，而不是每一个类使用一个log实例。从spring 3.2后，使用org.apache.commons.logging.LogFactory来代替了。  
ConstructorArgumentValues：保存构造方法的参数值，特别是作为Beandefinition的一部分。  
ConstructorArgumentValues.ValueHolder：保存构造方法的一个参数值。  
CustomEditorConfigurer：继承自BeanFactoryPostProcessor，给自定义属性编辑器的注册提供了便利的方法。  
CustomScopeConfigurer：BeanFactoryPostProcessor的简单实现，给自定义Scope的注册提供了便利的方法，上面提到ConfigurableBeanFactory提供了Scope的注册。  
DependencyDescriptor：将要注入的特定依赖的描述。  
DeprecatedBeanWarner：继承自BeanFactoryPostProcessor，记录@Deprecated bean的报警信息。  
FieldRetrievingFactoryBean：检索静态或者非静态Field的值的工厂bean。  
InstantiationAwareBeanPostProcessorAdapter：实现了SmartInstantiationAwareBeanPostProcessor所有方法的适配器，它没有任何操作，不会改变容器对bean进行初始化的处理过程。  
ListFactoryBean：共享一组(list)实例的简单工厂bean。  
MapFactoryBean：共享一组(map)实例的简单工厂bean。  
MethodInvokingFactoryBean：一个有返回值的工厂bean，它返回触发一个静态或者实例的方法的结果。  
ObjectFactoryCreatingFactoryBean：一个有返回值的工厂bean，它返回ObjectFactory。   
PlaceholderConfigurerSupport：属性资源配置器的抽象基类，它解析BeanDefinition中属性值的占位符。  
PreferencesPlaceholderConfigurer：PropertyPlaceholderConfigurer的子类，支持JDK1.4中的Preferences API (java.util.prefs)  
PropertiesFactoryBean：它支持从classpath位置的文件中读取属性实例的工厂bean。   
PropertyOverrideConfigurer：属性资源配置器，它支持在applicationContext中重写一个bean的属性值。  
PropertyPathFactoryBean：通过给定目标对象计算属性路径的工厂bean。  
PropertyPlaceholderConfigurer：PlaceholderConfigurerSupport的子类，它解析本地属性或者系统属性或者环境变量定义的占位符(以${}描述)。  
PropertyResourceConfigurer：支持从一个属性资源中对单个bean的属性值进行配置。  
ProviderCreatingFactoryBean：返回一个JSR-330 Provider的工厂bean，JSR-330 Provider反过来从beanFactory返回一个bean资源。  
RuntimeBeanNameReference：固定占位符类，当在beanfactory中作为另外一个bean名称的引用时，作为属性值对象，将在运行时进行解析。   
RuntimeBeanReference：固定占位符类，当在beanfactory中作为另外一个bean的引用时，作为属性值对象，将在运行时进行解析。   
ServiceLocatorFactoryBean：继承自FactoryBean，在beanFactory中创建一个动态代理，来代理一个具有一个或者多个服务或者类型的接口。  
SetFactoryBean;共享set实例的简单工厂bean  
TypedStringValue：保存一个类型的属性值。

上述类信息，可以参考FactoryBean的实现接口或者类。

8. org.springframework.beans.factory.parsing 解析beanDefinition所需的基本类和接口。

包含的接口有：

ComponentDefinition：描述在同一配置的Context中一组BeanDefinition和BeanReference的逻辑视图的接口。  
DefaultsDefinition：一个默认definition标识接口，继承了BeanMetadataElement，没有实现任何方法。   
ParseState.Entry：ParseState的入口标识接口   
ProblemReporter：spi接口，支持tool或者外部进程处理在beanDefinition解析期间报出的错误或者异常。.   
ReaderEventListener：接受在读取BeanDefinition进程中注册组件、别名、import时的回调接口。   
SourceExtractor：简单策略接口，允许工具控制source元数据关联到bean definition元数据。

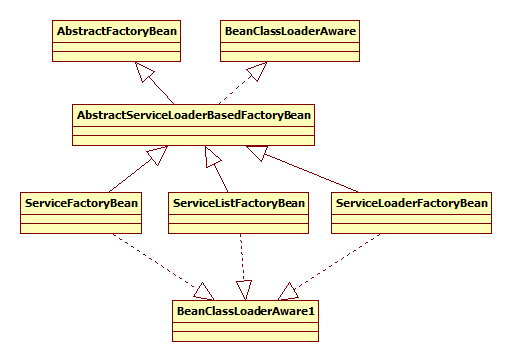
包含的类有：

AbstractComponentDefinition：ComponentDefinition接口的基本实现，提供了AbstractComponentDefinition.getDescription()来代理ComponentDefinition.getName()方法。  
AliasDefinition：代表在解析进程中一个别名已经被注册。  
BeanComponentDefinition：基于一个标准BeanDefinition的ComponentDefinition，暴露指定bean的指定beanDefinition，内部BeanDefinition和BeanReference。   
BeanEntry：代表了一个BeanDefinition。   
CompositeComponentDefinition：保存了一个或者多个内嵌ComponentDefinition实例的ComponentDefinition实现，它把这些ComponentDefinition实例聚合成具有命名的组。  
ConstructorArgumentEntry：代表了构造参数。  
EmptyReaderEventListener：ReaderEventListener接口的空实现，所有回调方法都没有提供可执行操作。  
FailFastProblemReporter：ProblemReporter接口的简单实现，当遇到错误发生时展示fail-fast行为。   
ImportDefinition：在解析进程中，代表一个import已经被处理。  
Location：模型接口，一个资源位置的模型。  
NullSourceExtractor：SourceExtractor接口的简单实现，返回null作为source元数据。   
ParseState：在解析进程中作为一个简单的基于栈结构的追踪逻辑位置类。   
PassThroughSourceExtractor：SourceExtractor的简单实现，它通过一个attachment来传递备选的源数据类型对象。  
Problem：代表了一个beanDefinition配置问题。   
PropertyEntry：代表了一个javaBean的属性。   
QualifierEntry：代表了一个自动绑定的备选qualifier。   
ReaderContext：bean definition读取进程中传递的一个Context，封装了所有相关的配置，包括状态。

BeanDefinitionValidationException：一个bean definition验证失败时抛出异常的异常类。

9. org.springframework.beans.factory.serviceloader包含的类有：

AbstractServiceLoaderBasedFactoryBean：FactoryBean的抽象基类，它是操作JDK1.6 ServiceLoader的基础工具。   
ServiceFactoryBean：暴露指定配置的服务类的基础服务的FactoryBean，通过JDK1.6 serviceLoader基础类来获取这些服务。  
ServiceListFactoryBean：暴露配置的服务类的所有基础服务的FactoryBea，表现为一组服务对象，可以通过JDK1.6 serviceLoader基础类来获取这些服务。  
ServiceLoaderFactoryBean：暴露指定配置服务类的JDK1.6 serviceLoader的FactoryBean。



10.org.springframework.beans.factory.support：

其中接口有：  
AutowireCandidateResolver：策略接口，对特定的依赖，这个接口决定一个特定的bean definition是否满足作为自动绑定的备选项。   
BeanDefinitionReader：bean definition解析器的简单接口。  
BeanDefinitionRegistry：持有beanDefinition的注册接口，例如RootBeanDefinition和ChildBeanDefinition实例。   
BeanDefinitionRegistryPostProcessor：标准BeanFactoryPostProcessor SPI的扩展接口，允许在常规BeanFactoryPostProcessor检查之前注册更多bean definition。  
BeanNameGenerator：对beanDefinition产生bean名称的策略接口。   
InstantiationStrategy：负责根据相应的根bean definition创建实例的接口。   
MergedBeanDefinitionPostProcessor：后处理回调接口，在运行时合并bean definition。  
MethodReplacer：一个可以重新定义Ioc容器对象的所有方法的接口：方法注入是依赖注入的一种形式。   
SecurityContextProvider：运行在beanFactory中的安全Context的提供者。

其中类有：  
AbstractAutowireCapableBeanFactory：抽象beanFactory的超类，它使用指定的RootBeanDefinition类的所有方法实现创建默认bean。   
AbstractBeanDefinition：具体、成熟BeanDefinition类的基类，构造出RootBeanDefinition和ChildBeanDefinition的通用属性。   
AbstractBeanDefinitionReader：实现了BeanDefinitionReader接口，是beanDefinitionReader的抽象基类。   
AbstractBeanFactory：BeanFactory的抽象基类实现，提供ConfigurableBeanFactory SPI的全部功能。  
AutowireCandidateQualifier解析自动绑定备选项Qualifier。  
BeanDefinitionBuilder：使用建造者模式构建BeanDefinition的编程方法。   
BeanDefinitionDefaults：简单保持BeanDefinition的默认属性类。  
BeanDefinitionReaderUtils：BeanDefinitionReader实现使用的工具方法类。  
CglibSubclassingInstantiationStrategy：BeanFactory默认对象初始化策略。  
ChildBeanDefinition：从父类继承各种设置的bean的beanDefinition。   
DefaultBeanNameGenerator：BeanNameGenerator接口的默认实现，代理BeanDefinitionReaderUtils.generateBeanName(BeanDefinition, BeanDefinitionRegistry)方法。  
DefaultListableBeanFactory：ListableBeanFactory接口和BeanDefinitionRegistry接口的默认实现：基于beanDefinition对象的一个成熟的beanFactory。  
DefaultSingletonBeanRegistry：共享bean实例的通用注册，实现了SingletonBeanRegistry。   
FactoryBeanRegistrySupport：一个支持单例注册（需要处理FactoryBean实例）的基类，集合了DefaultSingletonBeanRegistry对单例的管理功能。  
GenericBeanDefinition：通用BeanDefinition是一站式的标准bean Definition。  
LookupOverride：表示可以重写一个在同一个Ioc上下文中查找对象的方法。  
ManagedArray：集合类，用来保存它所管理的array元素，它可以包含运行时期的bean引用(将被解析为bean对象)  
ManagedList<E> 集合类，用来保存它所管理的List元素，它可以包含运行时期的bean引用(将被解析为bean对象).   
ManagedMap<K,V> 集合类，用来保存它所管理的array值，它可以包含运行时期的bean引用(将被解析为bean对象)   
ManagedProperties：表示一个spring管理的属性实例，它支持父/子 definition的合并。  
ManagedSet<E> 集合类，用来保存它所管理的set值，它可以包含运行时期的bean引用(将被解析为bean对象)   
MethodOverride：表示重写对象，它针对Ioc容器所管理对象的方法的重写。  
MethodOverrides：一组方法重写，决定了在运行时期对spring ioc容器管理对象的重写方法(如果存在的话)。   
PropertiesBeanDefinitionReader：简单属性格式的BeanDefinitionReader。  
ReplaceOverride：MethodOverride的扩展，表示一个Ioc容器方法的任意重写。  
RootBeanDefinition：根BeanDefinition，表示在运行时期的spring BeanFactory中一个合并的BeanDefinition。   
SimpleAutowireCandidateResolver：继承自AutowireCandidateResolver，当出现不支持的注解时使用。   
SimpleBeanDefinitionRegistry：BeanDefinitionRegistry接口的简单实现。  
SimpleInstantiationStrategy：BeanFactory中简单对象的初始化策略。  
SimpleSecurityContextProvider：SecurityContextProvider的简单扩展。  
StaticListableBeanFactory：静态BeanFactory的实现，用来编程实现注册已经存在的单例实例。

11.org.springframework.beans.factory.wiring包含的类有；

BeanWiringInfoResolver：策略接口，能够根据给定的一个新初始化的bean对象解析bean的名称信息。  
BeanConfigurerSupport：配置bean的便利基类，它可以对对象进行依赖注入。   
BeanWiringInfo：保存特定类的bean的绑定元数据信息。  
ClassNameBeanWiringInfoResolver：BeanWiringInfoResolver的简单默认实现，查找一个和全限定名同名的bean的名称。

12.org.springframework.beans.factory.xml

其中，接口有：  
BeanDefinitionDecorator：DefaultBeanDefinitionDocumentReader用来处理自定义的，内嵌的<bean>标签的接口   
BeanDefinitionDocumentReader：解析包含spring BeanDefinition的xml文件的SPI.   
BeanDefinitionParser：DefaultBeanDefinitionDocumentReader用来处理自定义的，高层的<beans>标签的接口   
DocumentLoader：加载xml文件的策略接口。   
NamespaceHandler：DefaultBeanDefinitionDocumentReader用来处理在spring xml配置文件中的自定义命名空间的基础接口。  
NamespaceHandlerResolver：DefaultBeanDefinitionDocumentReader用来定位NamespaceHandler的接口，NamespaceHandler实现了特定的URI命名空间。

其中，类有：  
AbstractBeanDefinitionParser：BeanDefinitionParser的抽象实现，提供了许多便利方法和模板方法，模板方法需要在子类进行重写来提供具体的逻辑实现。  
AbstractSimpleBeanDefinitionParser：AbstractBeanDefinitionParser的简单实现，当将要解析元素的属性名和配置类的属性名一一对应的时候，可以用到该类。   
AbstractSingleBeanDefinitionParser：BeanDefinitionParser的基类，需要解析和定义单独的BeanDefinition  
BeanDefinitionParserDelegate：状态代理类，用来解析xml BeanDefinition。  
BeansDtdResolver：spring bean dtd解析器EntityResolver的实现，用来从classpath或者jar文件加载dtd。  
DefaultBeanDefinitionDocumentReader：BeanDefinitionDocumentReader接口的默认实现。  
DefaultDocumentLoader：spring 的DocumentLoader默认实现。  
DefaultNamespaceHandlerResolver：NamespaceHandlerResolver接口的默认实现。   
DelegatingEntityResolver：EntityResolver的实现，分别代理了dtd的BeansDtdResolver和xml schemas的 PluggableSchemaResolver。   
DocumentDefaultsDefinition：简单的javaBean,它保存标准的spring xml文件中<bean>级别的属性，如：default-lazy-init, default-autowire等等  
NamespaceHandlerSupport：支持实现自定义的NamespaceHandler。   
ParserContext：传递一个beanDefinition 解析进程到此上下文，封装了所有的相关配置也包括状态。   
PluggableSchemaResolver：继承自EntityResolver，使用一系列map文件将schema url解析到本地classpath资源。   
ResourceEntityResolver：继承自EntityResolver，通过ResourceLoader来解析实体的引用。  
SimpleConstructorNamespaceHandler：单NamespaceHandler实现，它将自定义属性直接映射到bean属性。   
SimplePropertyNamespaceHandler：简单NamespaceHandler实现，它将自定义属性直接映射到bean属性。   
UtilNamespaceHandler：工具命名空间的NamespaceHandler  
XmlBeanDefinitionReader  
Bean definition reader for XML bean definitions.   
XmlBeanFactory 过期。从spring 3.2 后推荐使用DefaultListableBeanFactory和XmlBeanDefinitionReader来代替它。  
XmlReaderContext：ReaderContext的扩展，一般和XmlBeanDefinitionReader一起使用。

13. org.springframework.beans.factory.access包含的类有：

BeanFactoryLocator：定义一个BeanFactory及其子类(如ApplicationContext)的查询、使用、释放的协议。  
BeanFactoryReference：用来追踪一个通过BeanFactoryLocator获得的BeanFactory的引用。   
SingletonBeanFactoryLocator：BeanFactoryLocator的keyed-singleton实现，它可以访问共享的spring beanFactory实例。

14. org.springframework.beans.factory.access.el

SimpleSpringBeanELResolver：SpringBeanELResolver的简单实现，代理一个使用解析器构造的BeanFactory。  
SpringBeanELResolver：通用EL的ELResolver，代理一个spring BeanFactory，将名称引用解析为spring定义的bean。

小结：

spring beans 是Ioc容器实现的最初形态，深入理解spring beans模块有助于加深对IoC容器的认识，而基于web应用的IoC容器Application又是构建与BeanFactory之上的，这样的话，我怎么强调它的重要性都不为过吧。

spring beans最重要的两个东西要搞定即BeanFactory的继承关系和FactoryBean的继续关系，在阅读源码中，你会发现应用了很多设计模式，如工厂模式，建造者模式，适配器模式，模板方法模式，单例模式、代理模式等等，我就不一一列举了，自己去探索吧。

注：这篇博客我写了8天才基本的东西写完，差点坚持不住呀，几次想放弃，后续要继续丰富内容，为了有写继续写下去的动力，就直接发上去了。