IOC 概述：

IoC is also known as *dependency injection* (DI). It is a process whereby objects define their dependencies, that is, the other objects they work with, only through constructor arguments, arguments to a factory method, or properties that are set on the object instance after it is constructed or returned from a factory method. The container then *injects* those dependencies when it creates the bean. This process is fundamentally the inverse, hence the name *Inversion of Control* (IoC), of the bean itself controlling the instantiation or location of its dependencies by using direct construction of classes, or a mechanism such as the *Service Locator* pattern.

IOC 也称为依赖注入(dependency injection, DI)。它是一个过程，对象定义它们的依赖项，也就是说，它们使用的其他对象，仅通过构造函数参数、工厂方法的参数或对象实例在构造或从工厂方法返回后设置的属性。然后容器在创建 bean 时注入这些依赖项。这个过程从根本上说是 bean 本身的逆过程，也就是名称控制反转(IOC)，它通过使用类的直接构造或一种机制(如服务定位器模式)来控制依赖项的实例化或位置。

一般来说，IOC 是一种可以帮助我们解耦各业务对象间依赖关系的对象绑定方式，而 Spring 官方为我们提供了两种类型的容器来支持 IOC 的方式。

关于 Spring 提供的两种 IOC 容器：

第一种为：BeanFactory,通过对官方文档的简单解读，BeanFactory 是 Spring 提供的最基础的 IOC 容器，提供完整的 IOC 支持。

第二种为：ApplicationContext,如果把 BeanFactory 比作 SpringIOC 的核心的话,那么 ApplicationContext 继承了 BeanFactory 的所有功能，并且在此之上扩展了提供国际化等许多高级的扩展实现。也是官方较为推荐使用的 IOC 容器。

至于 BeanFactory 和 ApplicationContext 的差别，可以从官方文档中看出一部分。

Feature	BeanFactory	ApplicationContext
Bean instantiation/wiring	Yes	Yes
Integrated lifecycle management	No	Yes
Automatic BeanPostProcessor registration	No	Yes
Automatic BeanFactoryPostProcessor registration	No	Yes
Convenient MessageSource access (for internalization)	No	Yes
Built-in ApplicationEvent publication mechanism	No	Yes

Beanfactory 容器:

通过类名可以大致看出来，Beanfactory 就像是一个生产 Bean 的工厂，BeanFactory API 为 Spring 的 IoC 功能提供了基础,实际上 Beanfactory 只是一个接口，定义了最基本的方法，例如：

```
获得 Bean
 Object getBean(String name) throws BeansException;

//判断 Bean 是否存在
boolean containsBean(String name);

//是否为单例
boolean isSingleton(String name) throws NoSuchBeanDefinitionException;

//名称类型是否匹配
boolean isTypeMatch(String name, ResolvableType typeToMatch) throws
NoSuchBeanDefinitionException;

//获取别名
String[] getAliases(String name);
```

通过测试发现，最终使用的工厂类是：

```
15      @Autowired
16      private  BeanFactory beanFactory;
17
18      @Test
19      public void contextLoads() {
20
21          System.out.println("beanFactory:"+beanFactory.getClass());
22      }
23
```

SpringdemoApplicationTests › contextLoads()

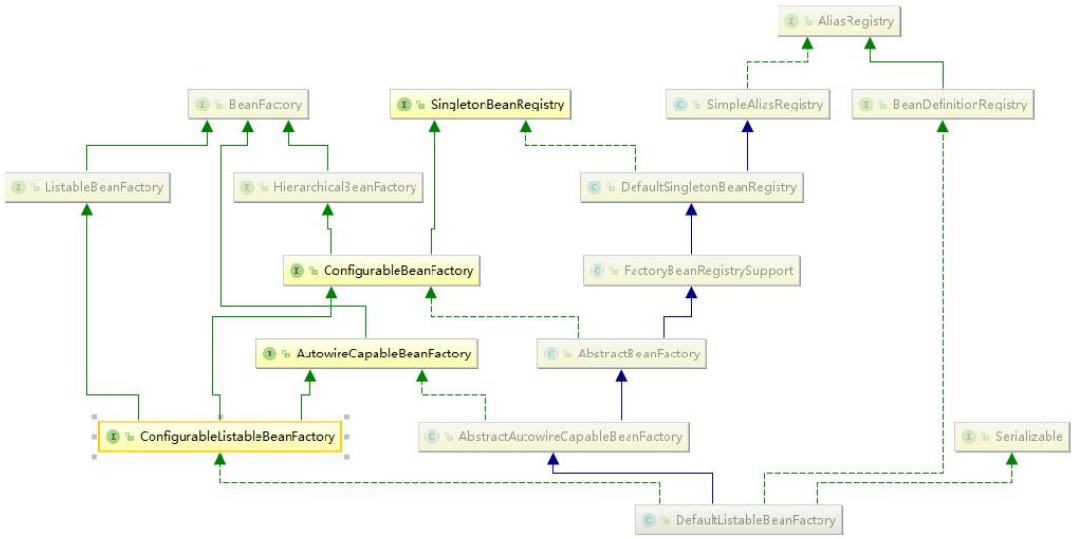
Tests passed: 1 of 1 test – 134 ms

```
  =========|_|==============|___/=/_/_/_/
  :: Spring Boot ::        (v2.1.0.RELEASE)

2018-11-24 11:10:51.014  INFO 4672 --- [           main] c.j.s.SpringdemoApplicationTests         : Starting SpringdemoApplicat
2018-11-24 11:10:51.014  INFO 4672 --- [           main] c.j.s.SpringdemoApplicationTests         : No active profile set, fall
2018-11-24 11:10:52.580  INFO 4672 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorServic
2018-11-24 11:10:52.829  INFO 4672 --- [           main] c.j.s.SpringdemoApplicationTests         : Started SpringdemoApplicati
beanFactory:class org.springframework.beans.factory.support.DefaultListableBeanFactory
2018-11-24 11:10:52.982  INFO 4672 --- [       Thread-2] o.s.s.concurrent.ThreadPoolTaskExecutor  : Shutting down ExecutorServi
```

BeanFactory:

class.org.springframework.beans.factory.support.DefaultListableBeanFactory 工厂，通过查看继承图



个人水平有限，这里只选择几个比较重要的接口简要概述。

AutowireCapableBeanFactory:

AutowireCapableBeanFactory 在 BeanFactory 基础上实现了对存在实例的管理.可以使用这个接口集成其它框架,捆绑并填充并不由 Spring 管理生命周期并已存在的实例.像集成 WebWork 的 Actions 和 Tapestry Page 就很实用.

个人的理解是可以填充不给 Spring 管理的 Bean，同时对 Bean 提供了更为细粒度的控制。

ListableBeanFactory:

可以枚举所有 bean 实例，而不是尝试 bean 查找，按客户要求一一列出。BeanFactory 实现，预加载它们的所有 bean 定义(例如基于 xml 的工厂)都可以实现此接口。

部分 Api

```
//得到 Bean 的数量
int getBeanDefinitionCount();
//检查此 bean 工厂是否包含具有给定名称的 bean 定义。
boolean containsBeanDefinition(String beanName);
//返回在这个工厂中定义的所有 bean 的名称。
String[] getBeanDefinitionNames();
```

ConfigurableBeanFactory :

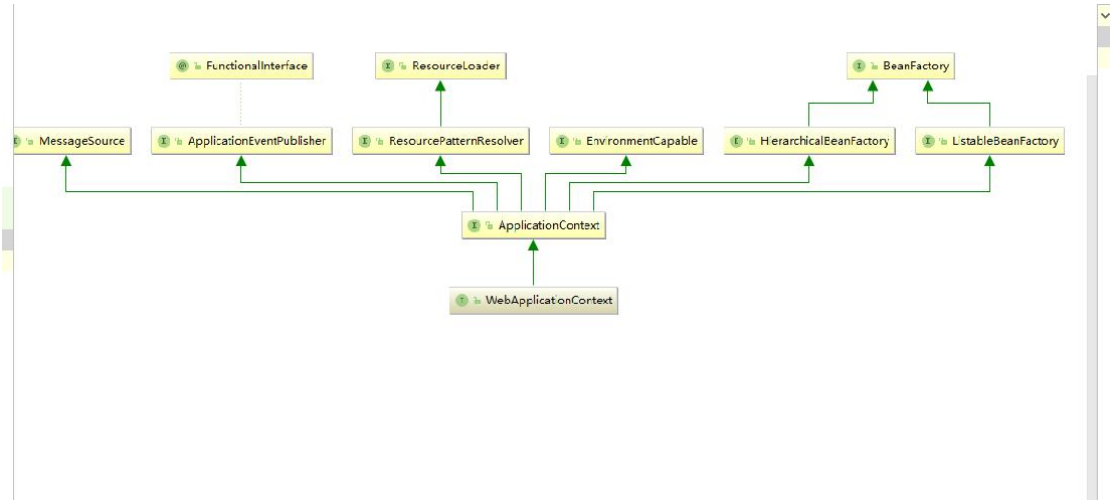
定义 Bean 的配置

SingletonBeanRegistry :

定义 Bean 的注册，实现类为 DefaultSingletonBeanRegistry

ApplicationContext 容器:

通过查看继承关系图，发现 ApplicationContext 最终也是实现了 Beanfactory 接口。



接下来重点看 ApplcationContext 相对于 Beanfactory 扩展的那一部分

ResourceLoader 接口：

Spring 提供了 ResourceLoader 接口用于实现不同的 Resource 加载策略，即将不同 Resource 实例的创建交给 ResourceLoader 来计算。

ApplicationEventPublisher 接口：
封装事件发布功能的接口。

MessageSource 接口：
提供国际化信息支持

同时，Spring 也为 ApplcationContext 接口提供了三个具体的实现供开发者调用。

FileSystemXmlApplicationContext：

在默认情况下，从文件系统加载 bean 定义以及相关资源的 ApplicationContext 实现

ClassPathXmlApplicationContext：

在默认情况下，从 Classpath 加载 bean 定义以及相关资源的 ApplicationContext 实现

XmlWebApplicationContext:

Spring 提供的用于 Web 应用程序的 ApplicationContext 实现。

同时在查看源码的过程中，我发现呢，Application 在 Spring 中和 SpringBoot 中略有不同，其中 Springboot，主要为 Web 应用额外提供了以下三个具体的实现类。当然这些实现类都实现了 ApplicationContext 接口。

23.6 Web Environment

A `SpringApplication` attempts to create the right type of `ApplicationContext` on your behalf. The algorithm used to determine a `WebApplicationType` is fairly simple:

- If Spring MVC is present, an `AnnotationConfigServletWebServerApplicationContext` is used
- If Spring MVC is not present and Spring WebFlux is present, an `AnnotationConfigReactiveWebServerApplicationContext` is used
- Otherwise, `AnnotationConfigApplicationContext` is used

This means that if you are using Spring MVC and the new `WebClient` from Spring WebFlux in the same application, Spring MVC will be used by default. You can override that easily by calling `setWebApplicationType(WebApplicationType)`.

It is also possible to take complete control of the `ApplicationContext` type that is used by calling `setApplicationContextClass(…)`.

如果使用 SpringMvc 则使用

AnnotationConfigServletWebServerApplicationContext

如果 Spring MVC 不存在，而 Spring WebFlux 存在则使用 AnnotationConfigReactiveWebServerApplicationContext

否则，将使用 AnnotationConfigApplicationContext

附一个 AnnotationConfigReactiveWebServerApplicationContext 实战小实例：

编写一个配置类 UserConfiguration，配置一个 Bean

```java
@Configuration
public class UserConfiguration {

    @Bean(name ="user")
    public User user(){
        User user = new User();
        user.setName("张三");
        return user;
    }
}
```

```java
public static void main(String[] args) {
    //构建一个应用上下文
    AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext();

    //注册一个Bean配置
    context.register(UserConfiguration.class);
    //启动容器
    context.refresh();
    //得到Bean
    User user = (User) context.getBean( name: "user");
    //输出
    System.out.println(user);


}
```

运行结果：

```
12:19:21.976 [main] DEBUG org.springframework.context.annotation.AnnotationConfigApplicationContext - Refreshing org.springframework.context.annotation.AnnotationConfigApplicati
12:19:22.005 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'org.springframework.context.annotati
12:19:22.198 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'org.springframework.context.event.in
12:19:22.205 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'org.springframework.context.event.in
12:19:22.215 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'org.springframework.context.annotati
12:19:22.215 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'org.springframework.context.annotati
12:19:22.249 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'userConfiguration'
12:19:22.269 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'user'
User{name='张三'}

Process finished with exit code 0
```

最后：参考资料

博客园：https://www.cnblogs.com/leftthen/p/5261837.html

Spring 官方文档：

https://docs.spring.io/spring/docs/5.0.11.BUILD-SNAPSHOT/spring-framework-reference/core.html#beans

Sprngboot 官方文档：

https://docs.spring.io/spring-boot/docs/2.0.6.RELEASE/reference/htmlsingle/

写在最后，个人水平有限，如果出现什么错误的地方，还请大家谅解。

作者：大川

时间：2018-11-24