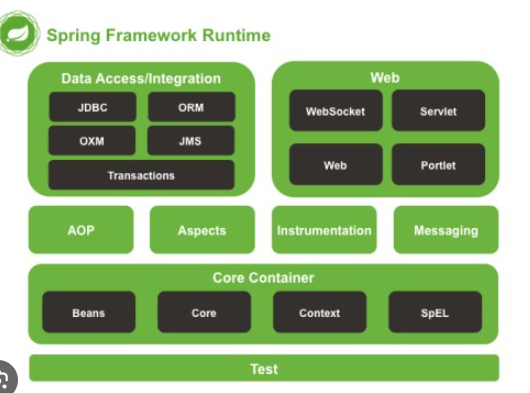
Dependency Lookup, Dependency Injection, IoC, Spring, Application Context

Здравствуйте дорогие слушатели. Меня зовут Василий. Я являюсь java разработчиком в финансовой сфере. И сегодня я подготовил краткую лекцию тему Dependency Lookup, Dependency Injection, IoC, Spring, Application Context



Spring — это мощный фреймворк для разработки Java-приложений, который значительно упрощает создание enterprise-приложений за счёт предоставления готовых решений для работы.

Любое приложение состоит из множества классов, и эти классы как то должны взаимодействовать между собой. Переиспользовать общие свойства. На этапе рассвета програмирования, разработчики писали собственные реализации для межклассового взаимодейсвтия. В дальнейшем они были названы фреймворками.

Что такое фреймворк – автоматическая работа. Начинающий разработчик может не разбиратьтся в деталях но знать основные принципы и уже уметь создавать какие то простые приложения.

IoC – мы отдаем управление над бинами фреймоворку и не тратим время на вмешательство.

Dependency Injection – поиск и внедрение зависимостей тоесть заполнение свойств объектов так же делается фреймвворком, у нас сокращается время на то чтобы сосредоточиться на написании логики приложения.

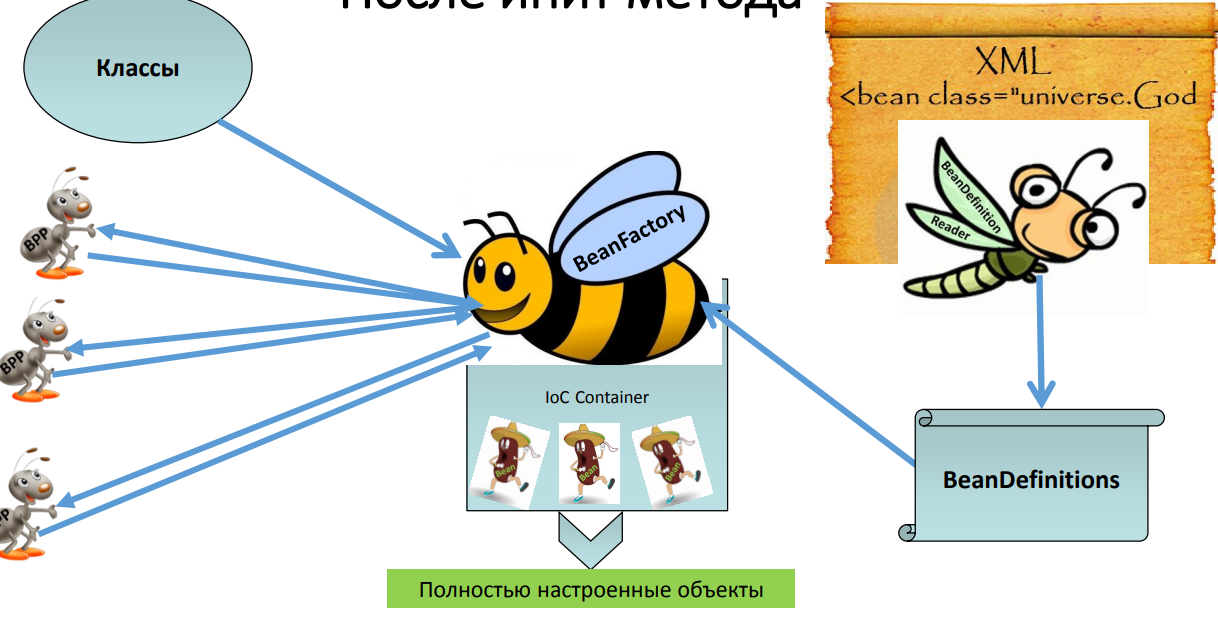
Бином в спаринге называют джава класс, который создан и управляем самим спрингом

XmlBeanDefinitionReader -сканирует xml и переводи в beanDefinition

BeanDefinition- объекты которые хронят информацию про бины. Их нейминг, какие зависимости им нужны, какими свойствами они обладают.

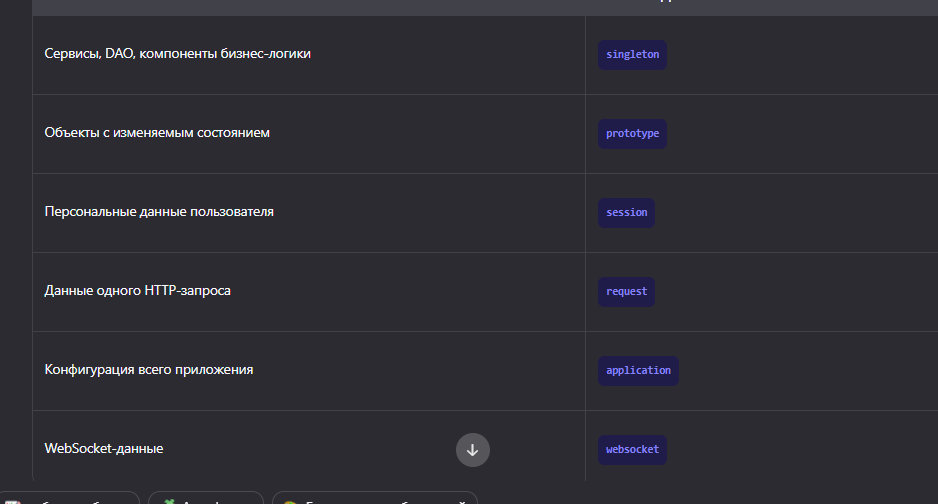
На начале создания был придуман механизм декларации классов, тоесть спринг должен был знать какие бины ему создавать. Под это подошел xml файл. Был создан XmlBeanDefinitionReader который читал файл, создавала beanDefinitions – шаблоны с инофрмацией о бинах, его свойствах и его имени. Все передавалось в BeanFactory в Spring — Базовый контейнер. И создавались бины. В дальнейшем были созданы механизмы донастройки бинов, перед или после этапов иництиализации чтобы обеспечить гибкость для разработчиков.







ПЕРЕЙТИ НА ПРИМЕР С XML

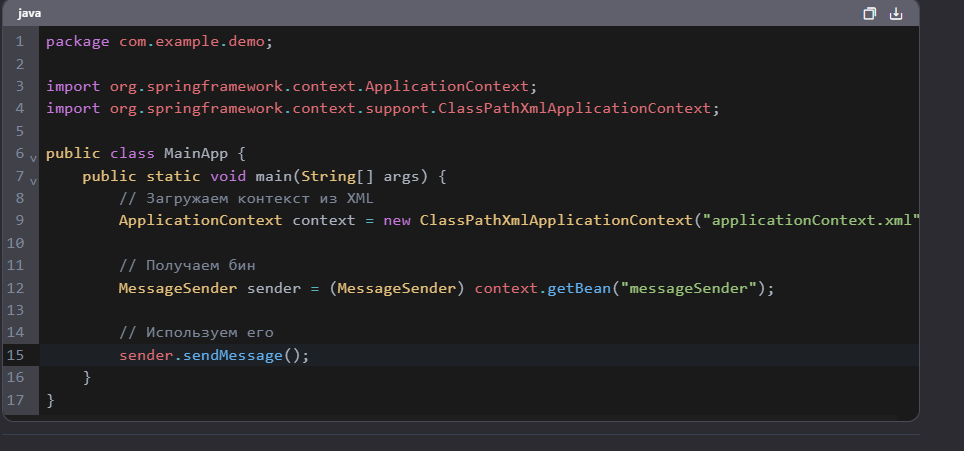


Хороший вопрос! 👌

Давай разберём, кладёт ли Spring prototype-бин в контейнер (ApplicationContext) и как он с ним работает.

🧠 Ответ коротко:

✅ Да , Spring создаёт prototype-бин через контейнер и может внедрять его в другие бины , но ❌ не управляет его временем жизни после создания .



BeanFactory в Spring — это...

Базовый контейнер (IoC-контейнер) в Spring Framework, который отвечает за:

-Создание и управление бинами

-Внедрение зависимостей (DI)

-Управление жизненным циклом бинов

Это самый базовый интерфейс контейнера , на котором строятся более мощные реализации, такие как ApplicationContext.

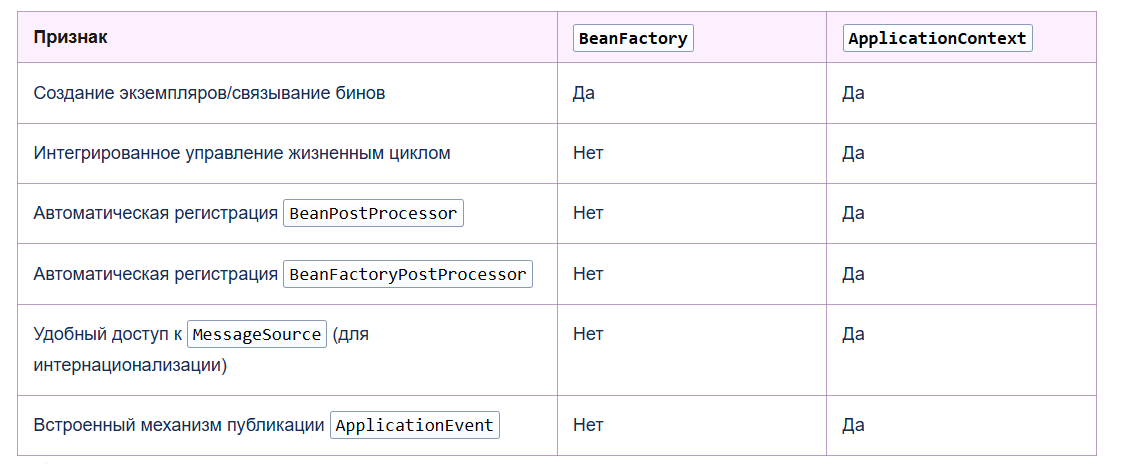
Когда стоит использовать BeanFactory?

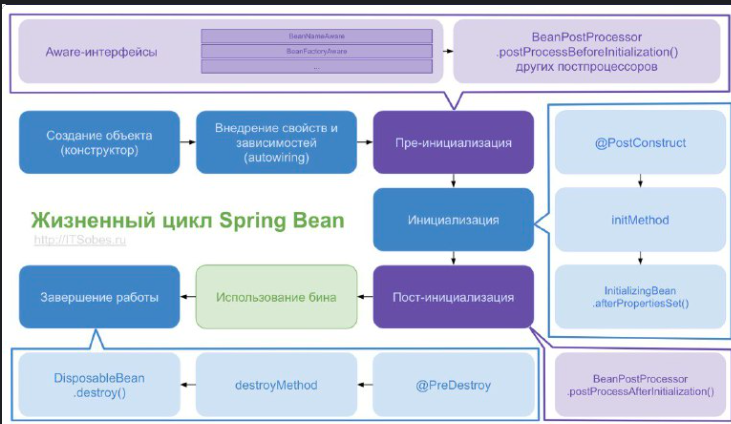
Если нужен минимальный контейнер без лишних функций

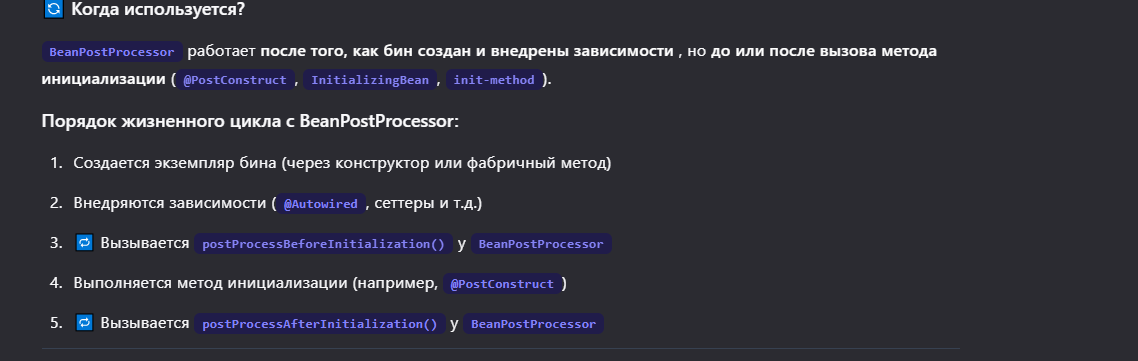
При работе с ограниченными ресурсами

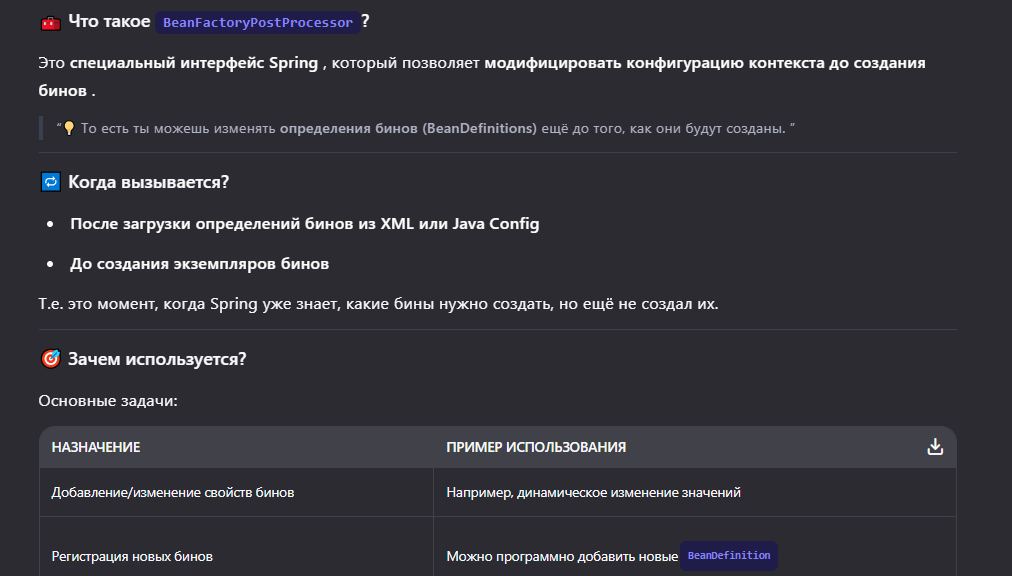
В случаях, когда тебе не нужны расширенные возможности Spring , например, события, сообщения, AOP

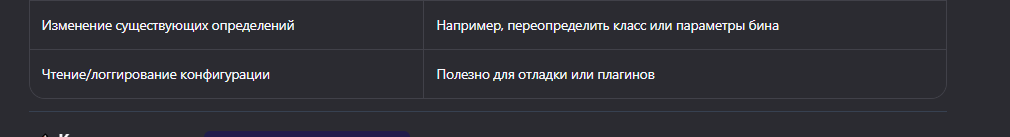
Но в реальных проектах чаще используют ApplicationContext, потому что он намного богаче по возможностям.











У интерфейса ApplicationContext есть большое количество реализаций:

\* ClassPathXmlApplicationContext;

\* FileSystemXmlApplicationContext;

\* GenericGroovyApplicationContext;

\* AnnotationConfigApplicationContext;

\* XmlWebApplicationContext

\* и даже StaticApplicationContext;

Они отличаются друг от друга именно тем, каким способом задаются мета-данные и где хранится эта конфигурация. Например:

— ClassPathXmlApplicationContext — метаданные конфигурируются XML-файлом(-ами) и они лежат в classpath, т. е. в ресурсах

модуля;

— FileSystemXmlApplicationContext — метаданные тоже конфигурируются XML-файлом(-ами), но они находятся где-то в файловой

системе, например, /etc/yourapp/spring-context.xml;

— AnnotationConfigApplicationContext — метаданные конфигурируются с помощью аннотаций прямо на классах.

ClassPathXmlApplicationContext будет читать файлы из вашего classpath. Они должны быть в classesпапке вашего

веб-приложения или в jarвашей libпапке.

FileSystemXmlApplicationContext может получить доступ ко всей вашей файловой системе, например c:

/config/applicationContext.xml.

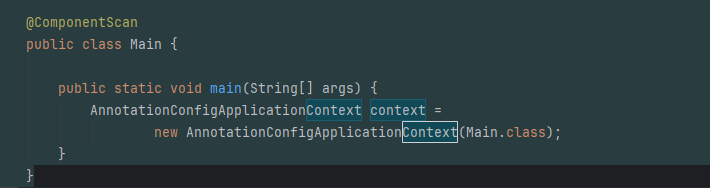
XmlWebApplicationContext конечно, может получить доступ к файлам, содержащимся в вашем веб-приложении, но это не самое

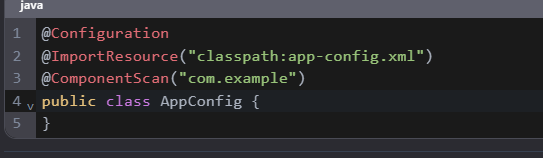
главное. Он реализует WebApplicationContext , а это означает, что он будет обнаруживать бины ServletContextAware ,

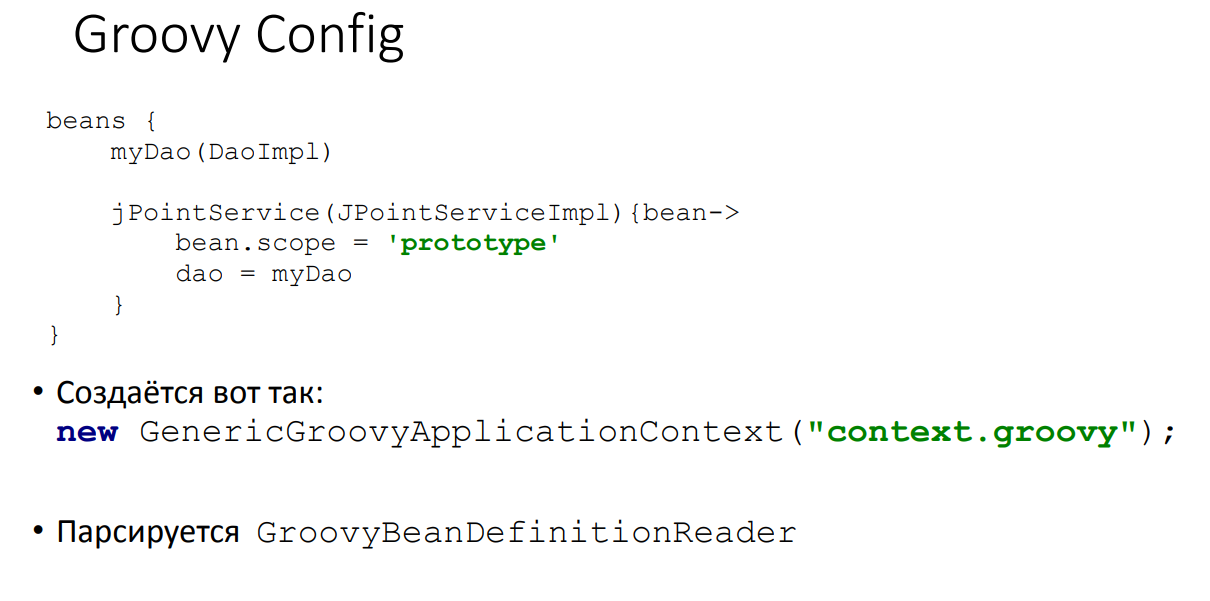
регистрировать пользовательские области (запрос, сеанс, ...) среди прочего

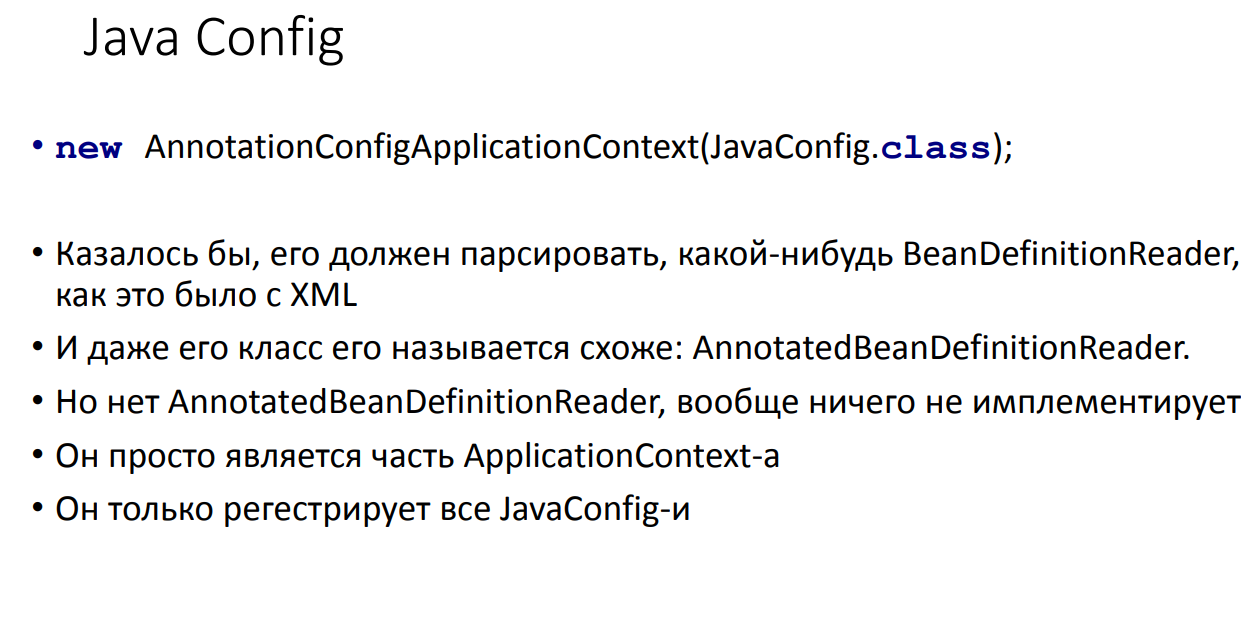
Современным способом конфигурирования считаются аннотации (AnnotationConfigApplicationContext), дальше будем создавать

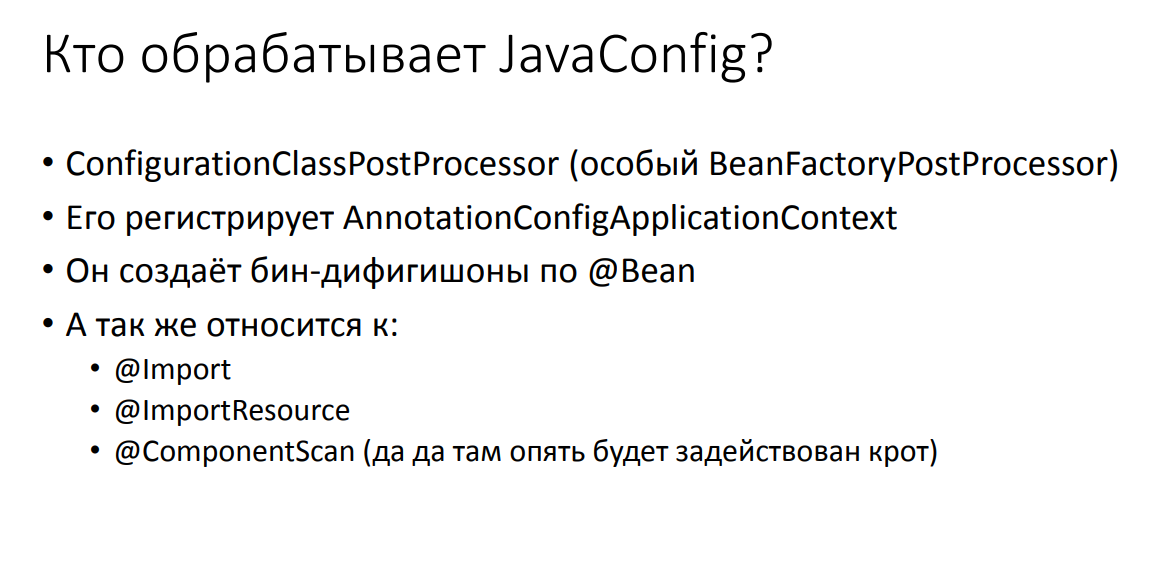
именно их.











**LOOKUP, INJECTION**

@ComponentScan: активирует сканирование аннотации @Component для пакета, в котором находится приложение

Сейчас я продемонстрирую пример Dependency Lookup — когда объект сам "достаёт" зависимость из контейнера при инициадизации. Мы в ручную говорим классу. «Возьми пожалуйста нужный нам сервис»

Для активации этих трех функций можно использовать единую аннотацию @SpringBootApplication:

@EnableAutoConfiguration: активирует механизм автоконфигурации Spring Boot

@ComponentScan: активирует сканирование аннотации @Component для пакета, в котором находится приложение

@SpringBootConfiguration: позволяет регистрировать дополнительные бины в контексте или импортировать дополнительные конфигурационные классы. Альтернатива стандартной аннотации @Configuration из Spring, которая помогает обнаруживать конфигурации в ваших интеграционных тестах.

В данном примере мы создаем класс обертку который нам напрямую дает доступ к контексту

И Dependency Injection – когда мы спрингу доверяем внедрение. Это проще и удобнее.

В enterprise разработке Dependency Lookup считается плахой практикой, так как мы лазим в контекст напрямую. Тем самым нарушая данный контракт спринга по автоматическому внедрению. Внутренние механизмы автовнедрения удобны и осуществляются 3 способами. Тоесть разработчики спринга создали гибкие механизмы для использования.