Отчет об изучении Spring

Выполнил Загреблеьный Александр

Spring Core

Интерфейс ApplicationContext

Одной из основных особенностей среды Spring является контейнер **IoC (Inversion of Control)**. Контейнер Spring IoC отвечает за управление объектами приложения. Он использует внедрение зависимостей для достижения инверсии управления.

Интерфейсы **BeanFactory** и **ApplicationContext** представляют контейнер Spring IoC.

Интерфейс ApplicationContext

ApplicationContext является подинтерфейсом **BeanFactory**. Таким образом, он предлагает все функции BeanFactory.

Основной задачей **ApplicationContext** является управление bean-компонентами.

Таким образом, приложение должно предоставить конфигурацию компонента контейнеру **ApplicationContext**.

Spring Bean

Spring Bean

В Spring объекты, формирующие основу вашего приложения и управляемые контейнером Spring IoC, называются **bean-компонентами. Bean** — создаваемый Spring-ом объект класса, который можно внедрить в качестве значения поля в другой объект.

```
@Component
public class Shop {

}

@Component
public class Seller {

 @Autowired
 private Shop shop;

}
```

С помощью аннотации **Component** класс помечается как **bean**.

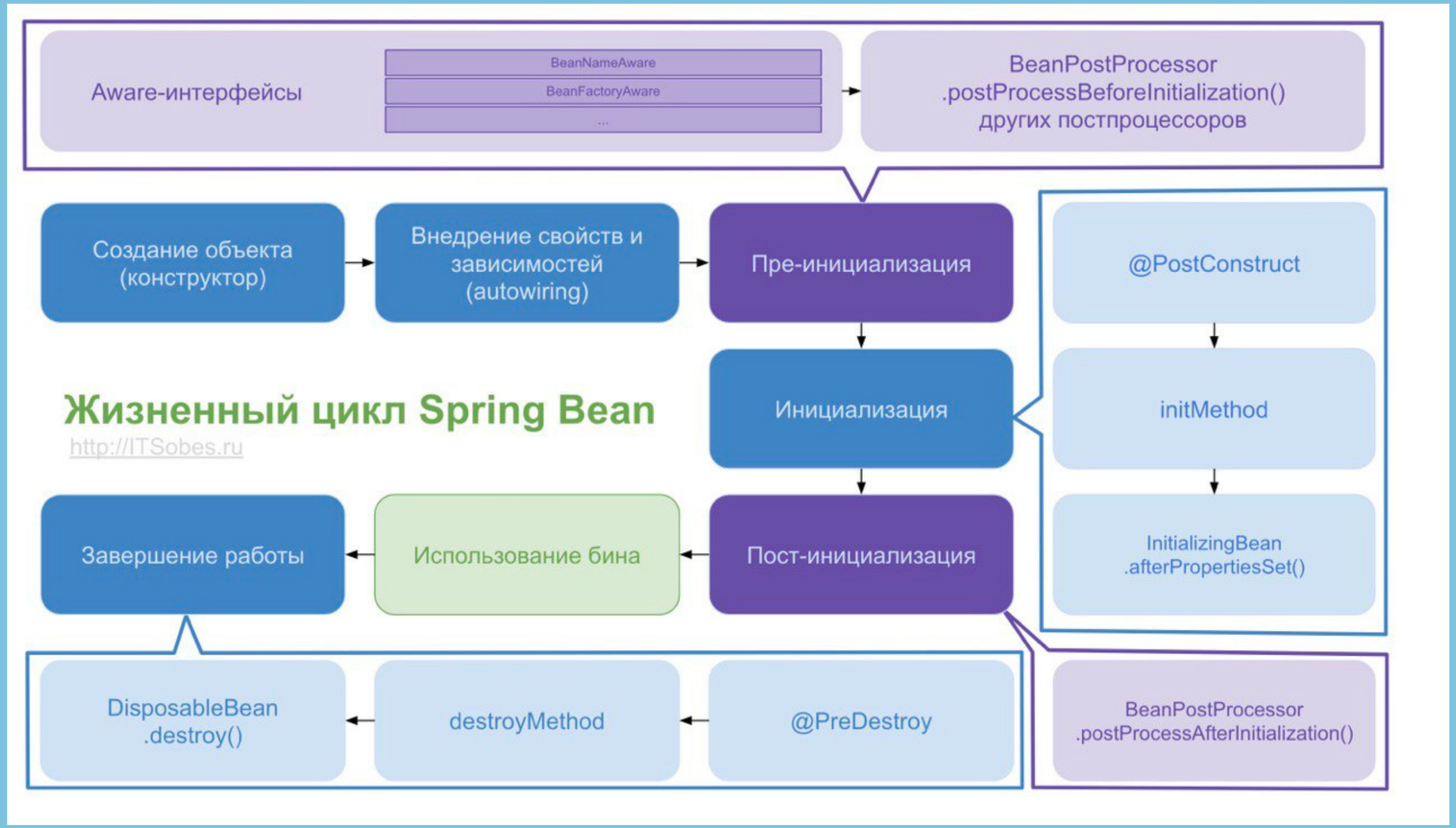
Spring Bean

Жизненный цикл Spring bean - время существования класса. Spring бины инициализируются при инициализации Spring контейнера и происходит внедрение всех зависимостей. Когда контейнер уничтожается, то уничтожается и всё содержимое. Если нам необходимо задать какое-либо действие при инициализации и уничтожении бина, то нужно воспользоваться методами **init()** и **destroy()**. Для этого можно использовать аннотации **@PostConstruct** и **@PreDestroy()**.

```
@PostConstruct
fun postConstruct(): Unit {
    println("init")
}
```

```
@PreDestroy()
fun destroy(): Unit {
    println("destroy")
}
```

Spring Bean


Aware-интерфейсы
BeanNameAware
BeanFactoryAware
...
BeanPostProcessor
.postProcessBeforeInitialization()
других постпроцессоров
Создание объекта
(конструктор)
Внедрение свойств и
зависимостей
(autowiring)
Пре-инициализация
@PostConstruct
Жизненный цикл Spring Bean
http://ITSobes.ru
Инициализация
initMethod
Завершение работы
Использование бина
Пост-инициализация
InitializingBean
.afterPropertiesSet()
DisposableBean
.destroy()
destroyMethod
@PreDestroy
BeanPostProcessor
.postProcessAfterInitialization()

Аннотации

Основные Spring Framework Аннотации

@Required

Применяется к методам-сеттерам и означает, что значение метода должно быть установлено в XML-файле. Если этого не будет сделано, то мы получим BeanInitializationException.

@Bean

Используется для указания того, что метод создает, настраивает и инициализирует новый объект, управляемый Spring IoC контейнером. Такие методы можно использовать как в классах с аннотацией @Configuration, так и в классах с аннотацией @Component(или её наследниках).

Основные Spring Framework Аннотации

@Autowired

Помечает конструктор, поле, метод установки или метод конфигурации как автоматически подключаемые средствами внедрения зависимостей Spring.

@Configuration

Аннотация на уровне класса, указывающая, что объект является источником определений **bean**-компонентов. Классы **@Configuration** объявляют компоненты с помощью общедоступных аннотированных методов **@Bean**.

Основные Spring Framework Аннотации

@ComponentScan

Эта аннотация используется вместе с **@Configuration** аннотацией, чтобы Spring мог узнать, какие пакеты сканируются для аннотированных компонентов

@Qualifier

Применяется для устранения случаев, когда мы хотели бы автоматически подключить более одного bean-компонента одного типа. Аннотация **@Qualifier** позволяет уточнить имя бина, который надо внедрить. Используется прямо перед аргументом. Работает в паре с аннотацией @Configuration

Properties

Properties

Файлы свойств используются для того, чтобы хранить количество N свойств в одном файле для запуска приложения в другой среде. В Spring Boot свойства хранятся в файле **application.properties**.

Файл **application.properties** находится в каталоге **src/main/resources**/.

Аналог application.properties — **application.yml**. Можно задавать свойства в любом формате, без разницы.

Зададим в application.properties пару свойств:

```
app.n=1
app.m=2
```

Properties

Теперь рассмотрим, как получить доступ к этим свойствам из кода.

Можно внедрить отдельное свойство с помощью аннотации **@Value**.

```
@Value("${app.n}")
var n: Int;

@Value("${app.m}")
var m: Int;
```

Properties

Можно также выделить серию свойств из файла **application.properties** и создать для нее отдельный класс, который можно внедрять. На первый взгляд такой вариант кажется сложнее, но он подходит, если свойств много.

```
@Configuration
@ConfigurationProperties(prefix = "app")
class AppProperties {
    var n: Int
    var m: Int
}
```

Spring Boot

Spring Boot

Spring Boot — это фреймворк на основе Java с открытым исходным кодом.
Благодаря быстродействию и простоте работы он стал популярным решением для создания развертываний в виде архива веб-приложений (WAR) и автономных Java-приложений.

Spring Boot

Spring Boot выделяется среди других фреймворков, поскольку он предоставляет разработчикам программного обеспечения гибкую настройку, надежную пакетную обработку, эффективный рабочий процесс и большое количество инструментов, помогая разрабатывать надежные и масштабируемые приложения на базе **Spring**.

Spring Boot

Когда дело доходит до **Spring Boot**, для начала следует упомянуть, что **Spring Boot** и **Spring Framework** — это разные технологии. **Spring** — это целая экосистема для разработки Java, включающая огромное количество готовых модулей, таких как **Spring MVC**, **Spring JDBC**, **Spring Security** и другие.

Spring Boot, напротив, является расширением **Spring**, используемым для создания приложений на основе микросервисов. Благодаря наличию целого ряда особенностей он облегчает и ускоряет процесс разработки, делая его более продуктивным.

Преимущества Spring Boot

Spring Boot создан, чтобы помочь программистам ускорить процесс разработки. Он позволяет избавиться от трудоемкой первоначальной установки и настройки среды развертывания. Основные преимущества **Spring Boot**:

- Быстрая и легкая разработка приложений на основе **Spring**.
- Автоконфигурация всех компонентов для приложения **Spring** производственного уровня.
- Готовые встроенные серверы (**Tomcat**, **Jetty** и **Undertow**), обеспечивающие ускоренное и более продуктивное развертывание приложений.

Преимущества Spring Boot

- **HTTP end-points**, позволяющие вводить внутренние функции приложения, такие как показатели, состояние здоровья и другие.
- Отсутствие конфигурации **XML**.
- Огромный выбор плагинов, облегчающих работу разработчиков со встроенными базами данных и базами данных в памяти.
- Легкий доступ к базам данных и службам очередей, таким как **MySQL**, **Oracle**, **MongoDB**, **Redis**, **ActiveMQ** и другим.
- Плавная интеграция с экосистемой **Spring**.
- Большое сообщество и множество обучающих программ, облегчающих ознакомительный период

Spring Data

Spring Data

Spring Data – это фреймворк, задача которого облегчить и упростить разработчику приложения работу с различными базами данных.

Миссией **Spring Data** является предоставление единой модели программирования с использованием **Spring** для доступа к данным, сохраняя при этом специальные черты базового хранилища.

Фреймворк позволяет облегчить использование технологий доступа к данным, реляционных и не реляционных баз данных, облачных баз данных.

Spring Data. Основные модули

- **Spring Data Commons** — основные концепции **Spring**, лежащие в основе каждого модуля **Spring Data**.
- **Spring Data JDBC** — поддержка репозитория **Spring Data** для **JDBC**.
- **Spring Data JDBC Ext** — поддержка специфичных для базы данных расширений стандартного **JDBC**, включая поддержку быстрого переключения соединений **Oracle RAC** при отказе, поддержку AQ JMS и поддержку использования расширенных типов данных.

Spring Data. Основные модули

- **Spring Data JPA** — поддержка репозитория Spring Data для JPA.
- Репозитории на основе **Spring Data KeyValue Map** и **SPI** для простого создания модуля Spring Data для хранилищ ключ-значение.
- **Spring Data LDAP** — поддержка репозитория Spring Data для **Spring LDAP** .
- **Spring Data MongoDB** — поддержка объектных документов и репозиториев для **MongoDB** на основе **Spring**.
- **Spring Data Redis** — простая настройка и доступ к **Redis** из приложений **Spring**.

Spring Data. Основные модули

- **Spring Data Redis** — простая настройка и доступ к **Redis** из приложений **Spring**.
- **Spring Data REST** — экспорт репозиториев **Spring Data** в виде ресурсов **RESTful**, управляемых гипермедиа.
- **Spring Data** для **Apache Cassandra** — простая настройка и доступ к **Apache Cassandra** или крупномасштабным, высокодоступным, ориентированным на данные приложениям Spring.

Spring Data. JDBCTemplate

JdbcTemplate — это центральный класс в основном пакете **JDBC**, который упрощает использование **JDBC** и помогает избежать распространенных ошибок. Он внутренне использует **JDBC API** и устраняет множество проблем с **JDBC API**. Он выполняет **SQL**-запросы или обновления, инициируя итерацию по наборам результатов, перехватывая исключения **JDBC** и переводя их в общий вид.

Спасибо за внимание!