AXIOM JDK

Фреймворк "Axiom Spring" Руководство пользователя

Axiom JDK | Апрель 2025

Программное обеспечение АКСИОМ содержит программное обеспечение с открытым исходным кодом. Дополнительная информация о коде сторонних разработчиков доступна на сайте https://axiomjdk.ru/third_party_licenses. Для дополнительной информации о том, как получить копию исходного кода, можно обратиться по адресу info@axiomjdk.ru.

ДАННАЯ ИНФОРМАЦИЯ МОЖЕТ ИЗМЕНЯТЬСЯ БЕЗ ПРЕДВАРИТЕЛЬНОГО УВЕДОМЛЕНИЯ. АКСИОМ ПРЕДОСТАВЛЯЕТ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ "КАК ЕСТЬ" БЕЗ КАКИХ-ЛИБО ГАРАНТИЙ, АКСИОМ ПРЯМО ОТКАЗЫВАЕТСЯ ОТ ВСЕХ ПОДРАЗУМЕВАЕМЫХ ГАРАНТИЙ, ВКЛЮЧАЯ, НО НЕ ОГРАНИЧИВАЯСЬ ПОДРАЗУМЕВАЕМЫМИ ГАРАНТИЯМИ ТОВАРНОЙ ПРИГОДНОСТИ И ПРИГОДНОСТИ ДЛЯ ОПРЕДЕЛЕННОЙ ЦЕЛИ.

АКСИОМ НИ ПРИ КАКИХ ОБСТОЯТЕЛЬСТВАХ НЕ НЕСЕТ ОТВЕТСТВЕННОСТИ ЗА ЛЮБЫЕ КОСВЕННЫЕ, СЛУЧАЙНЫЕ, СПЕЦИАЛЬНЫЕ, ШТРАФНЫЕ ИЛИ КОСВЕННЫЕ УБЫТКИ, ИЛИ УБЫТКИ ОТ ПОТЕРИ ПРИБЫЛИ, ДОХОДА, ДАННЫХ ИЛИ ИСПОЛЬЗОВАНИЯ ДАННЫХ, ПОНЕСЕННЫЕ ВАМИ ИЛИ ЛЮБОЙ ТРЕТЬЕЙ СТОРОНОЙ, БУДЬ ТО В РЕЗУЛЬТАТЕ ДЕЙСТВИЯ ДОГОВОРА ИЛИ ДЕЛИКТА, ДАЖЕ ЕСЛИ АКСИОМ БЫЛО ПРЕДУПРЕЖДЕНО О ВОЗМОЖНОСТИ ТАКИХ УБЫТКОВ.

Использование любого программного продукта АКСИОМ регулируется соответствующим лицензионным соглашением, которое никоим образом не изменяется условиями данного уведомления. Программные продукты и фирменные наименования: Axiom JDK, Axiom JDK Pro, Axiom Runtime Container Pro, Axiom Linux, Libercat, Libercat Certified и АКСИОМ принадлежат АКСИОМ и их использование допускается только с разрешения правообладателя.

Товарный знак Linux® используется в соответствии с сублицензией от Linux Foundation, эксклюзивного лицензиата Линуса Торвальдса, владельца знака на всемирной основе. Java и OpenJDK являются товарными знаками или зарегистрированными товарными знаками компании Oracle и/или ее аффилированных лиц. Другие торговые марки являются собственностью их соответствующих владельцев и используются только в целях идентификации.

Содержание

1. Введение

Настоящий документ содержит начальные сведения по эксплуатации программного обеспечения Фреймворк "Axiom Spring".

Что такое Фреймворк "Axiom Spring"?

Фреймворк "Axiom Spring" в Java — это фреймворк, который предоставляет набор инструментов и библиотек для упрощения и ускорения процесса разработки, позволяя сосредоточиться на бизнес-логике приложения.

С помощью Spring Java-разработчики создают серверное ПО для работы с современными десктопными, мобильными и веб-приложениями. Фреймворк полезен как для простых сервисов, так и для больших проектов из множества модулей и сотен функций.

Основные возможности Фреймворка

- **Упрощение разработки** - Spring автоматизирует многие аспекты разработки, такие как управление зависимостями и конфигурация компонентов. Это позволяет разработчикам сосредоточиться на бизнес-логике, а не на технических деталях.
- **Модульность и гибкость** - Благодаря своей модульной архитектуре Spring позволяет выбирать и интегрировать только те компоненты, которые необходимы для конкретного приложения.
- **Улучшение тестируемости** - Spring облегчает создание тестов для компонентов приложения, предоставляя механизмы для создания изолированных и легко тестируемых модулей.
- **Поддержка масштабируемости** - Spring позволяет строить масштабируемые приложения благодаря своей гибкой архитектуре и поддержке микросервисов.
- **Широкая экосистема и сообщество** - Spring обладает большим сообществом и обширной экосистемой дополнительных инструментов и библиотек, что обеспечивает поддержку и развитие фреймворка, а также предоставляет множество ресурсов для обучения и решения проблем.
- **Работа с данными** - Spring предоставляет инструменты для работы с базами данных и интеграции с различными технологиями, такими как JDBC, JPA и Hibernate.
- **Обеспечение безопасности** - Spring Security предоставляет комплексные решения для

обеспечения безопасности приложений, включая аутентификацию, авторизацию и защиту от атак.

Некоторые преимущества Фреймворка

- Возможность использования инверсии управления (Inversion of Control, IoC), которая позволяет уменьшить связность между компонентами системы;
- Поддержка аспектно-ориентированного программирования (Aspect-Oriented Programming, AOP), что упрощает реализацию функциональности, которая должна быть доступна во многих частях приложения (например, логирование);
- Обширная поддержка работы с базами данных, включая упрощение работы с SQL и поддержку транзакций;
- Встроенная поддержка различных протоколов и форматов данных для создания web-сервисов.
- Поддержка предыдущих версий Spring Framework, включающая исправление уязвимостей.
- Сборки Spring Framework соответствующие необходимым требованиям регуляторных органов.

Обзор экосистемы Spring

- **Spring Boot** — комплексный фреймворк для создания и запуска приложений с минимальными усилиями и настройками. Предоставляет разработчикам готовые шаблоны, что ускоряет время разработки и делает код более читаемым. С помощью Spring Boot можно создавать микросервисы, реактивные системы и веб-приложения.
- **Spring Data** — компонент, предназначенный для работы с разными типами баз данных (SQL и NoSQL), например, MySQL или Apache Cassandra. Содержит несколько внутренних компонентов, совместимых с разными базами данных. Основная цель — создать инструмент, который даст программистам возможность легко взаимодействовать с базами данных, при этом учитывая особенности каждой из них.
- **Spring MVC** — это модуль фреймворка Spring, который позволяет разрабатывать веб-приложения на Java. Он широко используется для создания масштабируемых веб-приложений. Spring MVC состоит из следующих компонентов: Model — View — Controller.
- **Spring Cloud** — используется в микросервисной архитектуре, упрощая взаимодействие микросервисов между собой и автоматизируя развёртывание приложений на облачных

платформах типа AWS, Azure и т. д..

- **Spring Batch** — платформа для разработки пакетных приложений. Подходит как для простых, так и для более сложных проектов — платформа легко масштабируется и может обрабатывать большие объёмы информации.

Spring имеет несколько фундаментальных особенностей, за счёт которых он выделяется на фоне других фреймворков: универсальность, облегчённость и поддержка инфраструктуры.

Фреймворк имеет модульную структуру, так как состоит из нескольких связанных друг с другом компонентов (модулей). Разработчики могут подключать только те модули, которые нужны им для решения задачи.

2. Начало работы с Фреймворком "Axiom Spring"

Системные требования

Ниже приведены только общие требования для Фреймфорка Axiom Spring. Конкретные версии компонентов и продуктов для вашей версии Фреймфорка Axiom Spring, узнавайте на странице Axiom Spring на сайте Axiom JDK.

Для Spring Framework 5.3.x и Spring Boot 2.x требуется наличие установленной JDK 8 или более поздней версии.

Начиная с версии 6.0, для Spring требуется JDK 17 или более поздняя версия.

Поддерживаются следующие инструменты сборки:

- Maven
- Gradle

Spring Boot поддерживает следующие встроенные контейнеры сервлетов:

- Axiom Libercat
- Jetty
- Undertow

Инструментарий Нативных Образов Axiom NIK Pro

Приложения Spring Boot могут быть преобразованы в нативный образ с помощью Axiom NIK Pro. Axiom NIK Pro - это универсальный инструмент на базе GraalVM Open Source для многоязычного программирования и ускорения работы ваших приложений.

Образы могут быть созданы с помощью встроенных средств сборки Gradle/Maven или встроенного средства создания нативных образов, предоставляемого GraalVM. Вы также можете создавать собственные образы, используя Paketo buildpack.

Настройка среды разработки

Установка Spring Boot

Spring Boot можно использовать с “классическими” средствами разработки на языке Java или установить как средство командной строки. Вам понадобится JDK версии 17 или выше. Перед началом работы проверьте текущую версию JDK с помощью следующей команды:

```
$ java -version
```

Вы можете использовать Spring Boot так же, как и любую стандартную библиотеку Java. Для этого включите соответствующие файлы `spring-boot-*.jar` в свою переменную `classpath`. Spring Boot не требует интеграции каких-либо специальных инструментов, поэтому вы можете использовать любую среду IDE или текстовый редактор. Кроме того, вы можете запускать и отлаживать приложение Spring Boot так же, как и любую другую Java-программу.

Создание проекта в IDE

В большинстве современных IDE (например, IntelliJ IDEA, Eclipse) установка Classpath происходит автоматически при добавлении зависимостей.

> **Примечание:**
>
> Вместо копирования файлов jar из Spring Boot, мы рекомендуем вам использовать инструмент сборки, поддерживающий управление зависимостями (например, Maven или Gradle).

Перед тем как создавать проект в своей IDE, выполните следующие шаги:

1. Создайте проект Spring Boot на странице Spring Initializr. Нужно заполнить следующие сведения:
    - **Project**: выбрать проект Maven или Gradle (в зависимости от предпочтений).
    - **Language**: Java.
    - **Spring Boot Version**: выбрать последнюю стабильную версию.
    - **Project Metadata**: указать группу, артефакт, имя и название пакета.
    - **Зависимости**: добавить, например, Spring Web, Spring Data JPA и Spring Boot DevTools (в

зависимости от требований проекта).

2. Нажмите на кнопку **Generate**, чтобы загрузить проект в виде ZIP-файла.
3. Распакуйте ZIP-файл созданного проекта Spring Boot в нужное место на локальном диске.

IntelliJ IDEA

> **Примечание:**
>
> Проекты Spring Boot можно создавать в IntelliJ IDEA версии Ultimate.

Импорт проекта Spring Boot в IntelliJ IDEA включает несколько шагов:

1. Откройте IntelliJ IDEA.
2. Перейдите в меню **Файл** > **Открыть**.
3. Выберите папку проекта Spring Boot и нажмите **OK**.

   IntelliJ IDEA автоматически определит тип проекта (Maven или Gradle) и начнёт импортировать проект. Нужно подождать, пока IntelliJ IDEA загрузит все необходимые зависимости и настроит проект.

Eclipse

> **Примечание:**
>
> При установке Eclipse IDE рекомендуется использовать пакет «Eclipse IDE для разработчиков Java» и Axiom JDK Pro.

Для импорта созданного проекта Spring Boot в Eclipse IDE, выполните следующие шаги:

1. Откройте Eclipse IDE.
2. Перейдите в меню **File** > **Import**.
3. Введите “maven” или "gradle" в строке поиска, выберите существующий проект и нажмите на кнопку **Next**.
4. Нажмите на кнопку **Browse** и выберите папку, в которую вы извлекли проект Spring Boot.
5. Нажмите **Finish** для завершения создания проекта Spring Boot.

Установка JDK

Мы рекомендуем использовать Axiom JDK Pro вместе с проектами использующими Фреймворк "Axiom Spring".

Вы можете установить Axiom JDK Pro на компьютеры под управлением Microsoft Windows, Linux и macOS. Тип установки, который вы выберете, зависит от ваших требований и платформы, которую вы используете.

Вы можете загрузить Axiom JDK Pro либо из Центра загрузок Axiom JDK Pro, либо по ссылке на портале поддержки. Доступ к этому порталу предоставляется клиентам с активным договором поддержки.

Подробную информацию по установке Axiom JDK Pro вы можете получить из Руководства по установке соответствующей версии на странице центра загрузок.

Инструменты сборки

Перед установкой инструментов сборки необходимо убедиться, что на компьютере установлена JDK. Чтобы убедиться в этом, выполните команду `Java -version` в терминале или командной строке.

> **Примечание:**
>
> Для автоматической установки инструментов сборки с помощью пакетных менеджеров или магазина приложений, обратитесь к документации вашей ОС или среды разработки.

Maven

Чтобы автоматизировать процесс скачивания необходимых компонентов Maven, выполните следующее:

1. Создайте проект Spring Boot на странице Spring Initializr и укажите Maven в разделе Project. Укажите другие настройки проекта, если необходимо.
2. Нажмите **Generate**, чтобы загрузить проект в виде ZIP-файла.
3. Распакуйте ZIP-файл созданного проекта Spring Boot в нужное место на локальном диске.
4. В распакованном проекте найдите файл `mvnw.sh` или `mvnw.cmd` в зависимости от используемой ОС. Запустите файл и все необходимые компоненты и зависимости для Maven будут закачаны и установлены автоматически.

Если вам не подходит автоматическая закачка и установка компонентов, установите Maven вручную, как показано ниже.

Чтобы установить Maven вручную, выполните следующие шаги:

1. Скачайте архив со сборщиком для своей операционной системы с официального сайта Apache Maven.
2. Распакуйте архив. Лучше создать для этого отдельную папку, так как путь к ней понадобится позже.
3. Отредактируйте переменные среды. На разных операционных системах этот процесс различается.

   Для **Windows**:

   a. Нажмите правой кнопкой мыши на «Этот компьютер» и выберите пункт меню «Свойства».

   b. Выберите «Дополнительные параметры системы» → «Дополнительно» → «Переменные среды».

   c. В окне «Переменные среды» найдите переменную Path и нажмите на кнопку «Изменить».

   d. В открывшемся окне нажмите кнопку «Создать» и укажите полный путь до папки bin из распакованного архива Maven.

   e. Проверьте настройку переменных среды. Для этого откройте командную строку и выполните команду `mvn -v`. Если Maven установлен правильно, то появится информация о его версии.

   Для Linux и macOS:

   a. Откройте в текстовом редакторе файл `~/.bashrc` или `~/.bash_profile` в Linux или `.zshrc` в macOS.

      Если файла нет, создайте его и впишите: `export PATH="<path_to_maven>:$PATH"`. Вместо `path_to_maven` укажите путь к файлу.

   b. Проверьте настройку переменных среды. Для этого запустите терминал и выполните команду `mvn -n`. Должно появиться сообщение с версией Maven.

Gradle

Чтобы автоматизировать процесс скачивания необходимых компонентов Gradle, выполните следующее:

1. Создайте проект Spring Boot на странице Spring Initializr и укажите Gradle в разделе Project.

Укажите другие настройки проекта, если необходимо.

2. Нажмите **Generate**, чтобы загрузить проект в виде ZIP-файла.
3. Распакуйте ZIP-файл созданного проекта Spring Boot в нужное место на локальном диске.
4. В распакованном проекте найдите файл `gradlew.sh` или `gradlew.bat` в зависимости от используемой ОС. Запустите файл и все необходимые компоненты и зависимости для Maven будут закачаны и установлены автоматически.

Если вам не подходит автоматическая закачка и установка компонентов, установите Gradle вручную, как показано ниже.

Чтобы установить Gradle, выполните следующие шаги:

1. Скачайте последнюю версию Gradle. Дистрибутив можно загрузить на официальном сайте по адресу gradle.org.
2. Распакуйте дистрибутив. Для Linux и macOS нужно распаковать ZIP-файл в выбранную директорию. В Windows нужно создать новую директорию `C:\Gradle` и распаковать папку `gradle-8.13` из ZIP-архива в созданную директорию `C:\Gradle`.
3. Настройте системную среду. Для Linux и macOS нужно настроить переменную среды PATH, чтобы включить директорию bin распакованного дистрибутива. В Windows в File Explorer нажмите правой кнопкой мыши на значок «Этот ПК» (или «Компьютер»), затем выберите «Свойства» — «Дополнительные системные настройки» — «Переменные среды». В разделе «Системные переменные» выберите Path, затем нажмите «Редактировать». Добавьте путь к каталогу bin в папке Gradle и сохраните изменения.
4. Проверьте правильность установки Gradle. Для этого откройте консоль или командную строку Windows и выполните команду `gradle -v`, чтобы запустить Gradle и вывести версию.

   Если установка прошла успешно, в выводе будет указана версия Gradle и другая информация.

Чтобы использовать Gradle в IntelliJ IDEA, нужно выполнить следующие шаги:

1. Запустите IntelliJ IDEA и создайте новый проект Java. Затем выберите папку, в которой будет храниться проект, и укажите его имя.
2. После создания проекта IntelliJ IDEA предложит настроить Gradle. Выберите опцию «Use auto-import» и укажите путь к установленному Gradle.
3. В корне проекта создайте файл с именем «build.gradle». В нём будут определяться настройки сборки.
4. Отредактируйте файл сборки.

5. После внесения изменений в файл сборки выполните синхронизацию Gradle, чтобы IntelliJ IDEA обновила зависимости и настройки проекта. Для этого щёлкните правой кнопкой мыши на «build.gradle» и выберите опцию «Sync Gradle».

   После синхронизации Gradle можно собрать Java-проект, выбрав опцию «Build» в меню IntelliJ IDEA

Чтобы использовать Gradle в Eclipse, нужно установить плагин Gradle Integration Plugin. Используйте следующие шаги:

1. Запустите Eclipse.
2. Нажмите «Help» > «Eclipse Marketplace».
3. Найдите Gradle Integration Plugin.
4. Установите плагин.

После установки плагина нужно настроить расположение Gradle в Eclipse:

1. Запустите Eclipse и перейдите в меню «Windows».
2. Нажмите «Preferences».
3. В окне «Preferences» найдите «Gradle» и выберите его.
4. В разделе «Gradle» найдите опцию «Локальный каталог установки».
5. Введите путь к папке Gradle в системе.

   На Windows это обычно место, где распаковывается папка Gradle.

   На Mac найти расположение Gradle можно с помощью команды «Brew info gradle» в терминале.
6. Нажмите «Применить» и «Закрыть», чтобы сохранить изменения.

Редактирование и запуск приложения Spring

Откройте проект в своей IDE и найдите файл `DemoApplication.java`, который находится в папке `src/main/java/com/example/demo`. Теперь измените содержимое файла, добавив дополнительный метод и аннотации, показанные в приведенном ниже коде. Вы можете скопировать и вставить код или просто ввести его.

Это весь код, необходимый для создания простого веб-сервиса “Hello World” в Spring Boot.

```
package com.example.demo;
```

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
@RestController
public class DemoApplication {
    public static void main(String[] args) {
      SpringApplication.run(DemoApplication.class, args);
    }
    @GetMapping("/hello")
    public String hello(@RequestParam(value = "name", defaultValue = "World")
String name) {
      return String.format("Hello %s!", name);
    }
}
```

Добавленный вами метод `hello()` предназначен для того, чтобы принимать строковый параметр name, а затем комбинировать этот параметр со словом "Hello" в коде. Это означает, что если вы укажете в запросе свое имя "Анна", ответом будет “Привет, Анна”.

Аннотация `@RestController` сообщает Spring, что этот код описывает конечную точку, которая должна быть доступна через Интернет. `@GetMapping(“/hello”)` указывает Spring использовать наш метод `hello()` для ответа на запросы, которые отправляются на адрес `http://localhost:8080/hello`. Наконец, `@RequestParam` сообщает Spring, что в запросе должно содержаться значение `name`, но если его там нет, по умолчанию будет использоваться слово "World".

Теперь вы можете создать и запустить программу. Откройте командную строку (или терминал) и перейдите в папку, где находятся файлы проекта. Вы можете создать и запустить приложение, выполнив следующую команду:

MacOS/Linux:

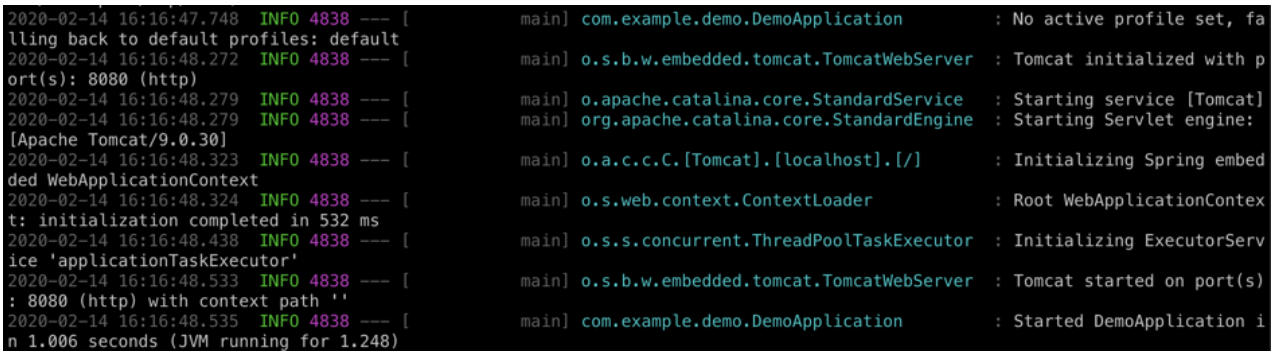
```
./gradlew bootRun
```

Windows:

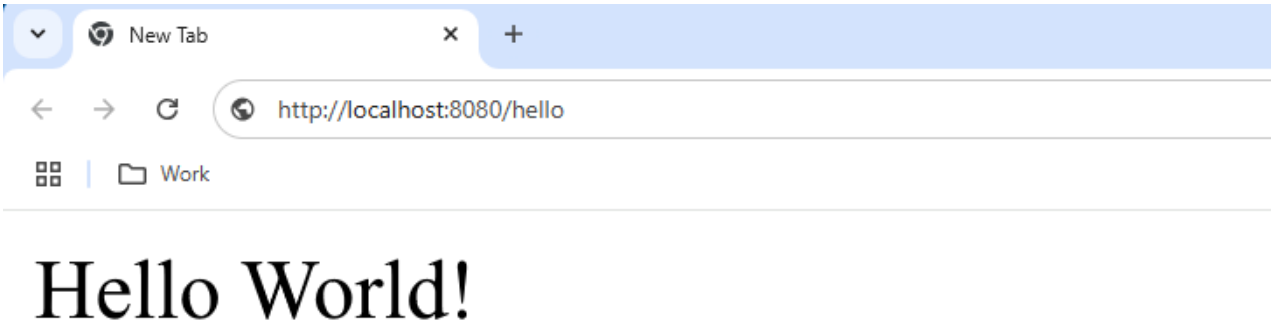
```
.\gradlew.bat bootRun
```

Вы должны увидеть некоторые выходные данные, которые выглядят как на следующей

иллюстрации:



Последние две строки говорят нам о том, что Spring запущен. Встроенный сервер Apache Tomcat в Spring Boot работает как веб-сервер и прослушивает запросы, поступающие на локальный порт 8080. Откройте браузер и в адресной строке вверху введите `http://localhost:8080/hello`. Вы должны получить ответ, подобный этому:



3. Основные концепции Spring

Инверсия управления (IoC)

Инверсия управления (IoC) в Spring — это принцип в разработке программного обеспечения, при котором управление объектами или частями программы передаётся контейнеру или фреймворку.

Другими словами, вы должны объяснить Spring (с помощью аннотаций либо в конфигурационном файле XML), какие именно объекты он должен создать самостоятельно. Spring управляет жизненным циклом объектов, и потому его контейнер называется IoC-контейнер.

Пакеты `org.springframework.beans` и `org.springframework.context` являются основой для IoC-контейнера в фреймворке Axiom Spring. Интерфейс BeanFactory предоставляет расширенный механизм настройки, позволяющий управлять объектами любого типа. ApplicationContext - это подинтерфейс BeanFactory. Он добавляет следующие возможности:

- Упрощенную интеграцию с функциями AOP Spring
- Обработка ресурсов сообщений (для использования в интернационализации)
- Публикация событий
- Контексты, специфичные для прикладного уровня, такие как WebApplicationContext, для использования в веб-приложениях.

BeanFactory предоставляет платформу конфигурации и базовые функциональные возможности, а ApplicationContext добавляет дополнительные функции, специфичные для конкретной реализации. ApplicationContext является полным дополнением к BeanFactory.

В Spring объекты, которые составляют основу вашего приложения и управляются контейнером Spring IoC, называются beans. Bean - это объект, который создается, собирается и управляется контейнером Spring IoC. В противном случае bean-компонент - это просто один из многих объектов в вашем приложении. Bean-компоненты и зависимости между ними отражены в метаданных конфигурации, используемых контейнером.

Внедрение зависимостей (DI)

Внедрение зависимостей (DI) - это способ предоставления объекту его зависимостей извне, вместо того чтобы объект создавал их самостоятельно. Затем контейнер IoC вводит эти зависимости при создании bean-компонента. Этот процесс, по сути, является обратным (отсюда и

название - инверсия управления) самому bean-компоненту, управляющему созданием экземпляров или расположением своих зависимостей с помощью прямого создания классов или такого механизма, как шаблон Service Locator.

Этот подход упрощает управление зависимостями и делает код более модульным, тестируемым и легко расширяемым.

Некоторые преимущества внедрения зависимостей в Spring:

- Сокращение объёма связующего кода. Уменьшается количество кода для связывания между собой разных компонентов приложения.
- Упрощение конфигурации приложения. Для конфигурирования классов, которые можно внедрять в другие классы, используют аннотации либо файлы XML.
- Возможность управлять общими зависимостями в одном репозитории. Вся информация об общих зависимостях находится в одном репозитории, что упрощает процесс управления зависимостями и уменьшает число возможных ошибок.

Внедрение конструктора

Внедрение зависимостей на основе конструктора выполняется с помощью контейнера, вызывающего конструктор с несколькими аргументами, каждый из которых представляет собой зависимость. Вызов фабричного метода `static` с определёнными аргументами для создания bean-компонента практически эквивалентен. В следующем примере показан класс, в который можно внедрить зависимости только с помощью внедрения через конструктор:

```
public class SimpleBookLister {

    // SimpleBookLister зависит от BookFinder
    private final BookFinder bookFinder;

    // конструктор, позволяющий контейнеру Spring внедрять программу
BookFinder
    public SimpleBookLister(BookFinder bookFinder) {
        this.bookFinder = bookFinder;
    }
}
```

Обратите внимание, что это простой Java-объект, который не зависит от конкретных интерфейсов, базовых классов или аннотаций контейнера.

Внедрение сеттера

Внедрение зависимостей на основе сеттера выполняется контейнером, который вызывает методы сеттеров для ваших компонентов после вызова конструктора без аргументов или метода `static` без аргументов для создания экземпляра вашего компонента.

В следующем примере показан класс, в который можно внедрить зависимости только с помощью чистой инъекции сеттеров. Этот класс является обычным классом Java. Это простой Java-объект, который не зависит от конкретных интерфейсов, базовых классов или аннотаций контейнера.

```
public class SimpleBookLister {

    // SimpleBookLister зависит от BookFinder
    private BookFinder bookFinder;

    // метод сеттера, чтобы контейнер Spring мог внедрить BookFinder
    public void setBookFinder(BookFinder bookFinder) {
        this.bookFinder = bookFinder;
    }
}
```

`ApplicationContext` поддерживает внедрение зависимостей на основе конструктора и сеттера для управляемых им компонентов. Он также поддерживает внедрение зависимостей на основе сеттера после того, как некоторые зависимости уже были внедрены с помощью конструктора.

> **Примечание:**
>
> Поскольку вы можете комбинировать внедрение зависимостей на основе конструкторов и сеттеров, рекомендуется использовать конструкторы для обязательных зависимостей, а методы сеттеров или методы конфигурации — для необязательных зависимостей. Обратите внимание, что для того, чтобы свойство стало обязательной зависимостью, можно использовать аннотацию `@Autowired` в методе сеттера; однако предпочтительнее внедрять зависимости через конструктор с программной проверкой аргументов.

Жизненный цикл bean-компонентов Spring

Жизненный цикл компонента Spring — последовательность событий, которые происходят с момента создания экземпляра компонента до его уничтожения.

Основные этапы жизненного цикла:

1. **Инициализация контейнера** - При запуске приложения Spring контейнер создаёт экземпляры всех bean-компонентов, определённых в конфигурации.
2. **Создание bean-компонента** - Контейнер вызывает конструктор или фабричный метод для создания экземпляра bean-компонента.
3. **Внедрение зависимостей** - После создания bean-компонента, контейнер внедряет зависимости, указанные в конфигурации. Это может быть сделано с помощью конструктора, сеттеров или аннотаций.
4. **Настройка bean-компонента** - После внедрения зависимостей, контейнер вызывает методы инициализации bean-компонента, которые могут быть определены в коде bean-компонента или с помощью аннотаций, таких как `@PostConstruct`.
5. **Использование bean-компонента** - После настройки bean-компонента, он готов к использованию в приложении. Клиентский код может получить доступ к bean-компоненту через контейнер и вызывать его методы.
6. **Уничтожение bean-компонента** - Когда контекст приложения закрывается или bean-компонент больше не нужен, контейнер вызывает методы его уничтожения, которые могут быть определены в коде bean-компонента или с помощью аннотаций, таких как `@PreDestroy`.

Области применения bean-компонентов

Фреймворк "Axiom Spring" предоставляет пять областей для bean-компонента. Вы можете использовать три из них только в контексте веб-приложения Spring ApplicationContext, а остальные две доступны как для контейнера IoC, так и для контейнера Spring-MVC. Ниже приведены различные области применения, предоставляемые для bean-компонента:

- **Singleton** - Для одного определения bean-компонента будет создан только один экземпляр для каждого контейнера Spring IoC, и один и тот же объект будет использоваться совместно для каждого запроса, выполненного для этого компонента.
- **Prototype** - Новый экземпляр будет создаваться для одного определения bean-компонента каждый раз, когда будет сделан запрос для этого компонента.
- **Request** - Новый экземпляр будет создаваться для отдельного определения bean-компонента каждый раз, когда для этого компонента выполняется HTTP-запрос. Но он действителен только в контексте веб-приложения Spring ApplicationContext.
- **Session** - Привязывает одно определение bean-компонента к жизненному циклу HTTP-сессии. Но действует только в контексте Spring ApplicationContext, поддерживающего веб-интерфейс.

- **Global-Session** - Привязывает одно определение bean-компонента к жизненному циклу глобальной HTTP-сессии. Это также действует только в контексте Spring ApplicationContext, поддерживающего веб-интерфейс.

Методы настройки

В данном разделе мы рассмотрим три основных способа конфигурации Spring:

- XML-конфигурация (`ClassPathXmlApplicationContext("context.xml")`).
- Конфигурация через аннотации.
- JavaConfig — конфигурация через аннотации с указанием класса (или массива классов), помеченного аннотацией `@Configuration`.

Конфигурация с помощью Xml файла

XML-конфигурация в Spring — это традиционный способ конфигурирования приложений Spring, при котором компоненты объявляются в файлах XML.

Для конфигурирования bean-компонента Spring с помощью xml файла необходимо создать конфигурационный файл в папке resources, как показано ниже:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">
</beans>
```

Для того чтобы Spring понимал какой bean-компонент необходимо создать, нам необходимо добавить информацию о классе в этот конфигурационный файл:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
```

```
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">

    <bean id="hello" class="springbean.xml.DemoApp"/>
</beans>
```

В данной конфигурации присваивается id, по которому мы будем получать данный bean-компонент и месторасположение данного класса.

Для тестирования приложения можно создать класс `Test` в котором вызвать метод `action` из bean-компонента:

```
public class Test {
    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext(
                "demoapp-context.xml"
        );
        Action hello = context.getBean("helloXML", HelloWorld.class);
        hello.action();
    }
}
```

При создании `ApplicationContext` указывается объект `ClassPathXmlApplicationContext`, которому передается название конфигурационного файла.

Далее можно получить bean-компонент по его id и вызвать метод `action()`, чтобы запустить приложение:

```
        Action hello = context.getBean("helloXML", HelloWorld.class);
        hello.action();
```

При запуске данного приложения вы должны увидеть информацию в консоли.

Конфигурация на основе аннотаций

Начиная с версии Spring 2.5, стала возможна конфигурация с помощью аннотаций. Конфигурация Spring на основе аннотаций позволяет связывать между собой бины, вставляя аннотации непосредственно в Java-класс. Доступны аннотации к классам, методам и полям.

Использование аннотаций в конфигурации Spring имеет ряд преимуществ: упрощённая конфигурация, простота обслуживания, автоматическое обнаружение.

Некоторые виды распространённых аннотаций:

- **@Required** - Применяется к методам установки свойств компонента.
- **@Autowired** - Может применяться к методам установки свойств компонента, методам, не являющимся установщиками, конструктору и свойствам.
- **@Qualifier** - Используется вместе с аннотациями Autowired, когда возможна путаница при связывании (непонятно, с каким бином необходимо связать) и определяет конкретный бин.
- **Аннотации на основе JSR-250** - Фреймворк "Axiom Spring" поддерживает аннотации, основанные на JSR-250 (@PostConstruct, @PreDestroy и т. д.).

По умолчанию, связывание с помощью аннотации не включено в контейнер Spring. Поэтому, прежде чем использовать аннотации, нужно разрешить их использование в конфигурационном файле Spring. Для этого нужно вставить `<context:annotation-config/>` в ваш конфигурационный файл:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">

<context:annotation-config/>
<!-- Далее будут определения bean-компонентов. -->

</beans>
```

Конфигурация на основе Java

Конфигурация Spring на основе Java — это альтернативный подход к конфигурации на основе XML для настройки и конфигурирования приложений Spring. С его помощью вы можете использовать простой код Java для определения конфигурации бинов, зависимостей и других компонентов приложения, а не полагаться на XML-файлы.

Конфигурация на основе Java, представленная в Spring 3.0, обладает рядом преимуществ по сравнению с конфигурацией на основе XML. Она обеспечивает более лаконичный и выразительный способ определения конфигурации приложения, позволяет обеспечить безопасность типов и поддержку рефакторинга, а также хорошо интегрируется с современными методами разработки на Java.

Чтобы использовать конфигурацию на основе Java, используйте аннотацию Spring `@Configuration`, чтобы пометить класс как класс конфигурации. Внутри этого класса можно определять компоненты и их зависимости, используя аннотацию `@Bean`, которая используется для объявления метода как производителя бинов. Возвращаемый тип метода представляет тип определяемого бина, а тело метода определяет логику создания и настройки бина.

Следующие шаги помогут вам создать конфигурацию на основе Java.

1. Создайте новый файл под названием `HelloWorld.java` в каталоге `src/`. Добавьте в файл следующий код:

   ```
   public class HelloWorld
   {
       private String msg;

       public HelloWorld(String msg)
       {this.msg = msg;}

       public void print()
       {System.out.println(msg);}
   }
   ```

   Это простой Java-объект, в котором есть поле msg. В нём также есть функция для вывода поля msg.

2. Создайте новый файл с именем `HelloWorldConfig.java` в каталоге `src/` Добавьте в файл следующий код:

   ```
   import org.springframework.context.annotation.Bean;
   import org.springframework.context.annotation.Configuration;

   @Configuration
   public class HelloWorldConfig
   {
       @Bean
       public HelloWorld helloWorld()
       {return new HelloWorld("Hello World!");}
   }
   ```

   Аннотация `@Configuration` используется, чтобы пометить класс как класс конфигурации на основе Java. Кроме того, мы использовали аннотацию `@Bean`, чтобы превратить функцию в bean-компонент.

3. В каталоге `src/` должен быть файл с именем `App.java`. Как правило, этот файл создается

автоматически при создании проекта. Этот файл содержит функцию `main()`. Измените файл следующим образом:

```
import org.springframework.context.ApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigApplicationContext;

public class App {
    public static void main(String[] args)
    {
        ApplicationContext context = new
AnnotationConfigApplicationContext(HelloWorldConfig.class);

        HelloWorld hw = context.getBean(HelloWorld.class);
        hw.print();
    }
}
```

Функция `main()` считывает файл класса конфигурации `HelloWorld` и получает объект `HelloWorld`. Наконец, объект используется для печати сообщения.

4. Теперь вы можете скомпилировать и запустить приложение.

AXIOM JDK

Фреймворк "Axiom Spring"
Руководство пользователя