AXIOM JDK

Фреймворк "Axiom Spring" Руководство администратора

Axiom JDK I 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 5

Фреймворк "Axiom Spring" в производственной среде 5

2. Системные требования и настройка среды 7

Spring Boot 7

Контейнеры сервлетов 7

Инструментарий Нативных Образов Axiom NIK Pro 7

Базы данных 8

3. Установка и развертывание 9

Загрузка и обновление Spring из репозиториев Axiom 9

Доступ к репозиторию 9

Просмотр и скачивание содержимого репозитория в браузере 9

Настройка инструментов сборки для работы с репозиторием 9

Maven 9

Gradle 11

Развертывание приложений Spring 12

Развёртывание через самостоятельный JAR-файл 12

Развертывание файла WAR 14

Контейнерное развертывание 15

Создание образа из файла Docker 15

Создание образа с помощью Cloud-Native Buildpack 16

Развертывание в Kubernetes 17

4. Использование профилей 19

Определение профиля в конфигурации 19

Активация профиля 19

Разделение конфигурации по файлам 20

Использование в коде 20

5. Мониторинг и управление приложением с Actuator 21

Настройка Spring Boot Actuator 21

Примеры полезных точек доступа 23

1. Введение

Настоящий документ содержит сведения по настройке и администрированию программного обеспечения Фреймворк "Axiom Spring". Данный документ предназначен для системных администраторов, разработчиков и команд по внедрению.

Фреймворк "Axiom Spring" в производственной среде

Spring — популярный фреймворк на основе Java, который упрощает и ускоряет разработку корпоративных приложений. Он состоит из множества мини-сервисов, библиотек и инструментов для создания масштабируемых и высокопроизводительных приложений.

Некоторые особенности Spring в производственной среде:

- Поддержка аспектно-ориентированного программирования. Позволяет отделить сквозные задачи, такие как ведение журнала, безопасность и кэширование, от основной бизнес-логики.
- Встроенная поддержка кэширования. Повышает производительность приложений, требующих частого доступа к данным.
- Набор инструментов для мониторинга и управления производительностью приложений. Помогает разработчикам быстро выявлять и устранять проблемы с производительностью.
- Интеграция с системами обмена сообщениями. Поддерживает различные технологии обмена сообщениями, такие как RabbitMQ, Kafka и ActiveMQ, что обеспечивает эффективную связь между службами.
- Интеграция с базами данных NoSQL. Позволяет эффективно хранить и извлекать данные, что важно в приложениях с высокой нагрузкой, где критична скорость доступа к данным.

Модули фреймворка Axiom Spring, которые полезны в производственной среде:

- **Spring Boot** - Обеспечивает упрощённый подход к созданию автономных приложений производственного уровня, способных обрабатывать большие нагрузки трафика.
- **Spring Data** - Предоставляет согласованную модель программирования для взаимодействия с различными источниками данных, включая реляционные базы данных, базы данных NoSQL и хранилища данных в памяти.
- **Spring Cloud** - Полезен для создания высоконагруженных приложений, работающих в облачных средах.

Фреймворк Spring широко используется в промышленности благодаря своей гибкости, модульности и простоте использования.

2. Системные требования и настройка среды

Требования к программному обеспечению для работы с Фреймворком Axiom Spring:

- JDK с помощью которой можно создавать приложения на платформе Spring. 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.
- Среда разработки приложений на языке Java. Например, InelliJ IDEA, NetBeans, Eclipse.
- СУБД для работы с базами данных.
- Драйверы JDBC для выбранной СУБД.
- Редактор HTML-кода.
- Сервер приложений Axiom Libercat для запуска веб-приложений.

Spring Boot

Поддерживаются следующие инструменты сборки:

- Maven
- Gradle

Контейнеры сервлетов

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

версии 22.3 или выше.

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.

Базы данных

Spring Data — это мощная платформа доступа к данным в экосистеме Spring, которая упрощает взаимодействие с базами данных для реляционных (SQL) и нереляционных (NoSQL) баз данных. Она избавляет от шаблонного кода и предоставляет простой в использовании уровень абстракции для разработчиков, работающих с JPA, MongoDB, Redis, Cassandra, Elasticsearch и другими.

Фреймворк "Axiom Spring" предоставляет обширную поддержку для работы с базами данных, от прямого доступа JDBC с использованием `JdbcClient` или `JdbcTemplate` до полноценных технологий «объектно-реляционного сопоставления», таких как Hibernate.

Spring Data JDBC в фреймворке Axiom Spring напрямую поддерживает различные версии баз данных, в том числе встроенные:

- DB2
- H2
- HSQLDB
- MariaDB
- Microsoft SQL Server
- MySQL
- Oracle
- Postgres

Spring Data предоставляет дополнительный уровень функциональности: создание реализаций Repository непосредственно на основе интерфейсов и использование соглашений для генерации запросов на основе названий ваших методов.

С появлением больших данных, облачных вычислений, и баз данных NoSQL, Spring Data стал необходим для масштабируемых и высокопроизводительных приложений. Он обеспечивает простую интеграцию с современными базами данных, сохраняя при этом простоту и гибкость Spring.

3. Установка и развертывание

Загрузка и обновление Spring из репозиториев Axiom

Доступ к репозиторию

Авторизация для доступа к репозиторию осуществляется с помощью имени пользователя - это имя проекта и пароля - это токен.

Имя проекта можно увидеть и скопировать в личном кабинете (ЛК) на портале поддержки под заголовком "Поддержка" в строке **Имя проекта**. Токен необходимо создать в том же личном кабинете на портале поддержки.

Информацию по созданию токена смотри в разделе "Аутентификация" главы "Автоматизация загрузок с партнерским API" в документе "Инструкция по работе с порталом поддержки". Там же находится общая информация по работе с порталом поддержки, включая авторизацию, доступ к продуктам и т.д.

Просмотр и скачивание содержимого репозитория в браузере

Содержимое репозитория можно просматривать по ссылке https://spring.axiomjdk.ru/ с помощью браузера (напр. Microsoft Edge, Google Chrome, Firefox и др.).

При сборке проекта нужные артефакты должны быть скачаны из репозитория axiom и использованы для построения вашего проекта.

Настройка инструментов сборки для работы с репозиторием

Maven

Ниже приведены настройки Maven для работы с репозиторием Axiom Spring.

Доверенный репозиторий может быть определен как внутри сборочного файла проекта `pom.xml`, так и внутри общего файла настроек Maven `settings.xml`. Определения в файле проекта имеют приоритет над определениями в файле настроек. При этом определения в файле настроек являются глобальными и доступны во всех проектах.

Для добавления доверенного репозитория в файл проекта `pom.xml` нужно в секции

<repositories> добавить элемент <repository> следующим образом:

```
<project>
...
    <repositories>
        <repository>
            <id>trusted-repo</id>
            <name>Axiom Trusted Repository</name>
            <url>https://spring.axiomjdk.ru/</url>
        </repository>
    ...
    </repositories>
...
</project>
```

Аналогичным образом репозиторий может быть определен в файле настроек settings.xml, находящимся в корне локального репозитория (обычно ~/.m2):

```
<settings>
    ...
    <profiles>
    ...
        <profile>
            <id>myprofile</id>
            <repositories>
                <repository>
                    <id>trusted-repo</id>
                    <name>Axiom Trusted Repository</name>
                    <url>https://spring.axiomjdk.ru/</url>
                </repository>
            </repositories>
        </profile>
    ...
    </profiles>

    <activeProfiles>
    <activeProfile>myprofile</activeProfile>
    </activeProfiles>
    ...
</settings>
```

Поскольку для доступа к доверенному репозиторию требуется пройти процедуру аутентификации, необходимо в файл настроек settings.xml добавить предоставленные имя

пользователя и пароль:

```
<settings>
    <servers>
        <server>
            <id>trusted-repo</id>
            <username>имя пользователя</username>
            <password>ваш токен</password>
        </server>
    </servers>
</settings>
```

Важно:

Обратите внимание на элемент `<id>` внутри элемента `<server>`, он должен совпадать с элементом `<id>` внутри элемента `<repository>`. В противном случае `maven` не сможет использовать предоставленные аутентификационные данные для доверенного репозитория.

В целях безопасности, мы рекомендуем использовать шифрование пароля, для более детальных инструкций ознакомьтесь с официальной документацией.

Gradle

Доверенный репозиторий может быть добавлен в секции `repositories` внутри сборочного файла проекта `build.gradle` (или иным способом, плагин, скрипт инициализации и т.п.).

```
repositories {
    ...
    maven {
        name "Axiom Trusted Repository"
        url "https://spring.axiomjdk.ru/"
        credentials {
            username "имя пользователя"
            password "ваш токен"
        }
    }
}
```

В целях безопасности, мы рекомендуем использовать специальные переменные для хранения пароля (имени пользователя) и передавать их в качестве параметров командной строки.

Для использования артефактов Axiom Spring в проектах необходимо к имени группы добавлять `ru.axiomjdk`, например, для Maven:

```
<dependency>
    <groupId>ru.axiomjdk.org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>6.2.5</version>
</dependency>
```

для Gradle:

```
implementation("ru.axiomjdk.org.springframework:spring-core:6.2.5")
```

Развертывание приложений Spring

Гибкие возможности упаковки Spring Boot предоставляют широкий выбор при развертывании вашего приложения. Вы можете развернуть приложения Spring Boot на различных облачных платформах, на виртуальных/реальных машинах или сделать их полностью исполняемыми.

В этом разделе рассматриваются некоторые из наиболее распространенных сценариев развертывания.

Развёртывание через самостоятельный JAR-файл

Вы можете упаковать ваше приложение в исполняемый JAR-файл, который содержит все зависимости и сервер приложения. Такой JAR запускается командой `java -jar <имя_файла>.jar`. Это простой и гибкий вариант развёртывания для небольших сервисов.

Если вы используете Maven, то для создания исполняемого файла JAR необходимо выполнить следующие шаги:

1. Создайте Maven-проект в используемой IDE.
2. Укажите правильную зависимость для создания JAR в файле `pom.xml`.

   ```
   <packaging>jar</packaging>
   ```

3. Добавьте необходимые зависимости. Например:

   ```
   <dependency>
   ```

```
    <groupId>ru.axiomjdk.org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>6.2.5</version>
</dependency>
```

4. Запустите командную строку или терминал, перейти в папку с `pom.xml` и введите команду:

   ```
   mvn clean package
   ```

   или

   ```
   ./mvnw clean package
   ```

Spring Boot автоматически встроит сервер приложений (например, Tomcat, Axiom Libercat, Jetty или Undertow) в приложение, так что вам не нужно настраивать внешний сервер.

Если вы используете Gradle, то для создания исполняемого файла JAR необходимо выполнить следующие шаги:

1. Создайте Gradle-проект в используемой IDE.
2. Настройте файл `build.gradle` для приложения Spring Boot.

   ```
   mainClassName = "com.company.application.Main"

   jar {
     manifest {
       attributes "Main-Class": "$mainClassName"
     }
   ```

3. Добавьте необходимые зависимости. Например:

   ```
   implementation("ru.axiomjdk.org.springframework:spring-core:6.2.5")
   ```

4. Откройте терминал или командную строку в корневом каталоге вашего проекта (где находится файл `build.gradle`) и выполните следующую команду:

   ```
   ./gradlew bootJar
   ```

   Задача `bootJar` автоматически упаковывает ваше приложение и все его зависимости, а также встроенный контейнер сервлетов (например, Tomcat, Axiom Libercat, Jetty или Undertow) в один исполняемый файл JAR.

После завершения сборки исполняемый JAR-файл будет размещен в каталоге `build/libs` вашего проекта.

Развертывание файла WAR

Это традиционный способ развертывания приложений на основе Spring, когда приложение собирается в файл WAR (Web Application Archive). Развёртывание приложений Spring с помощью WAR позволяет размещать их на внешних серверах приложений, таких как Tomcat, Axiom Libercat или Jetty. Сервер приложений будет обрабатывать запросы, и ваше приложение будет доступно по определенному URL.

Чтобы развернуть приложение на внешнем сервере, выполните следующие шаги.

1. Измените конфигурацию сборки таким образом, чтобы ваш проект создавал файл `war`, а не `jar`. Если вы используете Maven и `spring-boot-starter-parent` (который настраивает для вас плагин Maven war), всё, что вам нужно сделать, — это изменить `pom.xml` следующим образом:

```
<packaging>war</packaging>
```

2. Если вы используете Gradle, вам нужно изменить `build.gradle` для применения плагина `war` к проекту следующим образом:

```
apply plugin: 'war'
```

3. Убедитесь, что встроенный контейнер сервлетов не конфликтует с контейнером сервлетов, в котором развёрнут WAR-файл. Для этого необходимо пометить встроенную зависимость контейнера сервлетов как provided.

```
<packaging>war</packaging>

<dependencies>
    <!-- ... -->
    <dependency>
        <!-- Добавить зависимость от Tomcat -->
        <groupId>ru.axiomjdk.org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-tomcat</artifactId>
        <scope>provided</scope>
    </dependency>
    <!-- ... -->
</dependencies>
```

Если вы используете Gradle, в следующем примере контейнер сервлетов (в данном случае Tomcat) указан как provided:

```
dependencies {
    // ...
    providedRuntime 'ru.axiomjdk.org.springframework.boot:spring-boot-
starter-tomcat'
    // ...
}
```

Если вы используете плагины для сборки Spring Boot, пометка встроенной зависимости контейнера сервлетов как provided приводит к созданию исполняемого WAR-файла с включенными зависимостями, упакованными в каталог `lib-provided`. Это означает, что помимо развертывания в контейнере сервлетов вы также можете запустить свое приложение с помощью `java -jar` в командной строке.

4. Далее разворачивайте WAR файл на сервере приложений (например, Axiom Libercat).
5. Конфигурируйте сервер приложений для работы с приложением.

Контейнерное развертывание

Контейнеры становятся предпочтительным средством упаковки приложения со всеми зависимостями программного обеспечения и операционной системы, а затем доставки их в различные среды.

В этой части рассматривается:

- Создание образа Docker с помощью файла Docker
- Создание образа OCI из исходного кода с помощью Cloud-Native Buildpack
- Особенности развертывания в Kubernetes

Создание образа из файла Docker

В общем случае вам необходимо выполнить следующие шаги для создания контейнера с помощью файла Docker:

1. Создайте исполняемый файл JAR из вашего приложения Spring со всеми необходимыми зависимостями с помощью Maven или Gradle. Здесь мы используем Maven:

```
mvn clean package
```

Эта команда создает исполняемый JAR-файл приложения. Далее нужно преобразовать этот исполняемый JAR в образ Docker для работы в контейнере.

2. Добавьте ваше приложение в Dockerfile. Например:

```
FROM <URL адрес репозитория>/axiomjdk-pro:jre-17
WORKDIR /home/myapp
ARG JAR_FILE=target/*.jar
COPY target/myapp.jar application.jar
ENTRYPOINT ["java","-jar","/application.jar"]
```

Данный файл Docker содержит базовый образ, из репозитория Axiom, поверх которого копируется файл JAR. Также здесь указан каталог, в котором образ будет выполняться внутри Docker.

3. Создайте образ контейнера с помощью следующей команды:

```
docker build -t mycontainerapp .
```

4. Вы можете увидеть созданный образ в списке с помощью команды `docker images`.

5. Теперь можно запустить контейнер следующей командой:

```
docker run -p 8080:8080 myapp
```

Создание образа с помощью Cloud-Native Buildpack

Cloud-Native Buildpacks обеспечивают стандартизацию между платформами, поддерживая формат образа OCI, который гарантирует, что образ может запускаться движком Docker.

С помощью сборочных пакетов вы можете создавать образы, совместимые с Docker, которые можно запускать где угодно. Spring Boot уже содержит плагины Maven / Gradle, отвечающие за реализацию пакетов сборки.

Maven:

```
<plugin>
    <groupId>ru.axiomjdk.org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
```

Gradle:

```
plugins {
  java
  id("ru.axiomjdk.org.springframework.boot") version "3.4.4"
  id("io.spring.dependency-management") version "1.1.7"
}
```

Spring Boot создает образы OCI из исходного кода с помощью Buildpack. Образы создаются с использованием `bootBuildImage` задачи (Gradle) или `spring-boot:build-image` цели (Maven) и локальной установки Docker.

- Команда Maven для выполнения build-image цели

  ```
  mvn spring-boot:build-image
  ```

- Команда Gradle для выполнения задачи bootBuildImage

  ```
  gradle bootBuildImage
  ```

Пока идет создание образа вы можете изучить выходные данные терминала или командной строки.

Развертывание в Kubernetes

Kubernetes — это система с открытым исходным кодом для автоматизации развёртывания, масштабирования и управления контейнерными приложениями. Она группирует контейнеры, из которых состоит приложение, в логические единицы для упрощения управления и обнаружения.

Перед тем как развертывать приложение в Kubernetes, убедитесь, что вы можете использовать командную строку Linux или Linux-подобной системы. Это может быть терминал Linux, терминал MacOS или оболочка WSL в Windows. Вам также понадобится кластер Kubernetes и инструмент командной строки Kubectl. Вы можете создать кластер локально с помощью Kind (на Docker) или Minikube. В качестве альтернативы вы можете использовать облачного провайдера.

Чтобы развернуть ваше приложение Spring в Kubernetes, выполните следующие шаги:

1. Постройте файл `.jar` из приложения Spring Boot с помощью следующей команды либо воспользуйтесь готовым приложением:

   ```
   $ mvnw install
   ```

   Если сборка прошла успешно, вы увидите файл JAR.

2. Контейнеризируйте приложение как показано в разделе Создание образа с помощью Cloud-Native Buildpack.

   Вы можете запустить контейнер локально, чтобы проверить работу приложения, например:

   ```
   $ docker run -p <имя репозитория>:<тег образа>
   ```

3. Для запуска приложения в Kubernetes необходимо создать файл YAML в котором указываются параметры необходимые для определения и настройки ресурсов, таких как модули, службы и развёртывания. Вы также можете использовать kubectl, чтобы сгенерировать файл YAML.

   ```
   $ kubectl create deployment demo --image=<имя образа> --dry-run -o=yaml >
   deployment.yaml
   $ echo --- >> deployment.yaml
   ```

   Замените ‹имя образа› на существующее имя.

4. Выполните следующую команду для запуска приложения:

   ```
   $ kubectl apply -f deployment.yaml
   deployment.apps/demo created
   service/demo created
   ```

5. Убедитесь, что приложение запущено с помощью команды `kubectl get all`.

Spring Boot автоматически определяет среду развёртывания Kubernetes, проверяя окружение на наличие переменных `*_SERVICE_HOST` и `*_SERVICE_PORT`. Вы можете отменить это определение с помощью свойства `spring.main.cloud-platform`.

Spring Boot помогает вам управлять состоянием вашего приложения и экспортировать его с помощью HTTP Kubernetes Probes с помощью Actuator.

4. Использование профилей

В Spring (включая Spring Boot) механизмы профилей (`@Profile`) позволяют гибко управлять конфигурацией в зависимости от окружения — например, `dev`, `test`, `prod` и т.д.

Определение профиля в конфигурации

Вы можете аннотировать `@Component`, `@Service`, `@Configuration`, `@Bean` и т. д., чтобы они загружались только при активном определённом профиле, например:

```
@Configuration
@Profile("dev")
public class DevConfig {
    @Bean
    public DataSource dataSource() {
        // Настройка DataSource для разработки
        return new HikariDataSource();
    }
}
```

Активация профиля

Активация профиля в `application.properties`:

```
spring.profiles.active=dev
```

Активация профиля в файле YAML:

```
spring:
  profiles:
    active: dev
```

Активация через переменную окружения или флаг при запуске:

```
java -jar your-app.jar --spring.profiles.active=prod
```

В IDE, например IntelliJ необходимо добавить VM options в конфигурацию **Run** › **Debug**:

```
-Dspring.profiles.active=dev
```

> **Совет:**
>
> Вы можете активировать одновременно несколько профилей: `spring.profiles.active=dev,debug`.

Разделение конфигурации по файлам

Можно создать разные файлы с разными настройками и Spring автоматически подгрузит нужный файл в зависимости от активного профиля, например:

- `application-dev.properties`
- `application-prod.properties`

Использование в коде

Если вы хотите программно узнать активный профиль, то можете это сделать следующим образом:

```
@Autowired
private Environment env;

public void checkProfiles() {
    System.out.println(Arrays.toString(env.getActiveProfiles()));
}
```

5. Мониторинг и управление приложением с Actuator

Spring Boot Actuator — это подпроект Spring Boot, который предоставляет готовые к работе функции для мониторинга и управления приложением.

Он предоставляет набор встроенных точек доступа (endpoints), которые позволяют получать информацию о состоянии приложения, метриках и среде, а также динамически управлять им.

Некоторые возможности Spring Boot Actuator:

- Проверка состояния приложения и его зависимостей.
- Сбор метрик, например, использование памяти, сбор мусора, детали веб-запроса и т. д.
- Информация о среде. Доступ к свойствам среды приложения.
- Информация о сборке приложения. Получение информации о сборке приложения, такой как версия и имя.
- Динамические уровни логирования. Изменение уровней логов без перезапуска приложения.
- Трассировка HTTP-запросов.

Настройка Spring Boot Actuator

1. Добавьте зависимости.

   Если используется Maven:

   ```
   <dependency>
       <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-actuator</artifactId>
   </dependency>
   ```

   Если используется Gradle:

   ```
   implementation 'org.springframework.boot:spring-boot-starter-actuator'
   ```

2. Укажите точки доступа (endpoints) в `application.properties` или `application.yml`.

Пример для `application.properties`:

```
management.endpoints.web.exposure.include=health,info,metrics
```

Можно также открыть все точки:

```
management.endpoints.web.exposure.include=*
```

Пример для YAML:

```
management:
  endpoints:
    web:
      exposure:
        include: "*"
```

3. Измените базовый путь (необязательно).

```
management.endpoints.web.base-path=/actuator
```

В этом случае по умолчанию все эндпоинты будут доступны по пути `/actuator`, например:

- `/actuator/health`
- `/actuator/info`
- `/actuator/metrics`

4. Добавьте информацию в `/actuator/info` (опционально)

```
info.app.name=MyApp
info.app.version=1.0.0
```

И добавьте аннотацию, если нужно:

```
@RefreshScope
@Configuration
public class AppInfoConfig {
}
```

5. Настройте безопасность.

Если вы используете Spring Security, доступ к точкам доступа actuator может быть ограничен.

Пример настройки в `application.properties`:

```
management.endpoints.web.exposure.include=*
management.endpoint.health.show-details=always
```

Можно отдельно настроить доступ в SecurityConfig, например:

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http
        .authorizeHttpRequests()
        .requestMatchers("/actuator/**").permitAll()
        .anyRequest().authenticated();
}
```

Примеры полезных точек доступа

- `/actuator/health` — состояние приложения
- `/actuator/info` — информация по настройкам из info.*
- `/actuator/metrics` — метрики JVM, CPU, HTTP и др.
- `/actuator/beans` — все созданные бины
- `/actuator/env` — переменные окружения и свойства

AXIOM JDK

Фреймворк "Axiom Spring"
Руководство администратора