**Контрольные вопросы**

**Что такое JDBC, JPA и Hibernate? В чем их разница?**

***JDBC (англ. Java DataBase Connectivity – соединение с базами данных на Java)*** – платформенно независимый промышленный стандарт взаимодействия Java-приложений с различными СУБД, реализованный в виде пакета java.sql, входящего в состав Java SE.

JDBC основан на концепции так называемых драйверов, позволяющих получать соединение с базой данных по специально описанному URL. Драйверы могут загружаться динамически (во время работы программы). Загрузившись, драйвер сам регистрирует себя и вызывается автоматически, когда программа требует URL, содержащий протокол, за который драйвер отвечает.

***Java Persistence API (JPA)*** – спецификация API Java EE, предоставляет возможность сохранять в удобном виде Java-объекты в базе данных.

Существует несколько реализаций этого интерфейса, одна из самых популярных использует для этого Hibernate. JPA реализует концепцию ORM.

Поддержка сохранности данных, предоставляемая JPA, покрывает области:

* непосредственно API, заданный в пакете javax.persistence;
* платформо-независимый объектно-ориентированный язык запросов Java Persistence Query Language;
* метаинформация, описывающая связи между объектами.
* Генерация DDL для сущностей

***Hibernate*** – библиотека для языка программирования Java, предназначенная для решения задач объектно-реляционного отображения (ORM), самая популярная реализация спецификации JPA. Распространяется свободно на условиях GNU Lesser General Public License.

Позволяет сократить объёмы низкоуровневого программирования при работе с реляционными базами данных; может использоваться как в процессе проектирования системы классов и таблиц «с нуля», так и для работы с уже существующей базой.

Библиотека не только решает задачу связи классов Java с таблицами базы данных (и типов данных Java с типами данных SQL), но и также предоставляет средства для автоматической генерации и обновления набора таблиц, построения запросов и обработки полученных данных и может значительно уменьшить время разработки, которое обычно тратится на ручное написание SQL- и JDBC-кода. Hibernate автоматизирует генерацию SQL-запросов и освобождает разработчика от ручной обработки результирующего набора данных и преобразования объектов, максимально облегчая перенос (портирование) приложения на любые базы данных SQL.

***Различия:***

JDBC является гораздо более низкой (и более старой) спецификацией, чем JPA. JDBC – это API-интерфейс для взаимодействия с базой данных с использованием чистого SQL – отправки запросов и получения результатов. Он не имеет понятия об объектах или иерархиях. При использовании JDBC вам необходимо преобразовать набор результатов в объекты Java.

А в JPA (который использует JDBC снизу) вы также указываете эти детали метаданных базы данных, но с использованием аннотаций Java. Таким образом, JPA создает запросы на обновление для вас и управляет объектами, которые вы искали или создали / обновили (это также делает больше).

Hibernate одна из самых популярных открытых реализаций последней версии спецификации. То есть JPA только описывает правила и API, а Hibernate реализует эти описания.

**1. Как развернуть приложение Spring Boot на сервере (tomcat)?**

Чтобы файл WAR был развернут и запущен в Tomcat, нужно выполнить следующие шаги:

1. Загрузить Apache Tomcat и распаковать его в папку tomcat
2. Скопировать файл WAR из target/spring-boot-deployment.war с папкой tomcat /webapps /
3. С терминала перейти в папку tomcat / bin и выполнить запуск catalina.bat (в Windows)
4. Перейти к http://localhost:8080

**2. В чем разница между Spring Boot и Spring MVC?**

Разница между Spring MVC и Spring Boot:

|  |  |  |
| --- | --- | --- |
| № | SPRING MVC | Spring Boot |
| 1 | Spring MVC - это представление модели и веб-платформа на основе контроллера, широко используемая для разработки веб-приложений. | Spring Boot построен поверх обычного spring framework, широко используемого для разработки REST API |
| 2 | Если мы используем Spring MVC, нам нужно создать конфигурацию вручную. | Если мы используем Spring Boot, нет необходимости создавать конфигурацию вручную. |
| 3 | В Spring MVC требуется дескриптор развертывания | В Spring Boot нет необходимости в дескрипторе развертывания |
| 4 | Spring MVC определяет каждую зависимость отдельно | Он объединяет зависимости в единое целое. |
| 5 | Платформа Spring MVC состоит из четырех компонентов : модели, представления, контроллера и внешнего контроллера. | В Spring Boot есть четыре основных уровня: уровень представления, уровень доступа к данным, уровень обслуживания и уровень интеграции. |
| 6 | Это занимает больше времени в разработке | Это сокращает время разработки и повышает производительность |
| 7 | Spring MVC не обеспечивают мощной пакетной обработки. | Мощная пакетная обработка обеспечивается Spring Boot |
| 8 | Готовые к использованию функции предоставляются ит для создания веб-приложений. | Ит-службы предоставляют конфигурации по умолчанию для построения фреймворка с поддержкой Spring |

**3. Что такое War-файл и отличие от Jar-файла?**

***JAR*** – Java Archive. Содержит файлы классов, ресурсы, зависимые библиотеки, и другие необходимые для приложения файлы. Может содержать точку входа, и использоваться как цель для исполнения команды java.

***WAR*** – Web Archive. Технически имеет ту же структуру, но другую роль – архив JavaEE web-компонента. Обычно содержит jar-ы с реализацией, JSP, статические файлы фронт-энда, и мета-информацию для сервлет-контейнера (web.xml). В основном используется как деплоймент web-приложения в сервлет-контейнер. С приходом Servlet API 3.0 и embedded-контейнеров, всё больше становится принято запаковывать и web-компоненты в самодостаточные jar.

**4. Зачем использовать Spring Boot?**

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

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 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.
* 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.

**5. Как можно запустить Spring Boot?**

1. Запуск из IDE

Вы можете запустить приложение Spring Boot из IDE как простое Java приложение. Однако сначала вам нужно импортировать ваш проект. Этапы импорта зависят от вашей IDE и системы сборки. Большинство IDE могут импортировать проекты Maven напрямую. Например, пользователи Eclipse могут выбрать «Импорт… → Существующие проекты Maven» из меню «Файл».

Если вы не можете напрямую импортировать свой проект в IDE, вы можете создать метаданные IDE с помощью подключаемого модуля сборки. Maven включает плагины для Eclipse и IDEA. Gradle предлагает плагины для различных IDE.

2. Запуск в виде упакованного приложения

Если вы используете Spring Boot Maven или Gradle плагины для создания исполняемого файла jar, вы можете запустить свое приложение, используя java -jar, как показано в следующем примере:

*$ java -jar target/myapplication-0.0.1-SNAPSHOT.jar*

Также возможно запустить упакованное приложение с включенной поддержкой удаленной отладки. Это позволяет подключить отладчик к упакованному приложению, как показано в следующем примере:

*$ java -Xdebug -Xrunjdwp:server=y,transport=dt\_socket,address=8000, suspend=n \-jar target/myapplication-0.0.1-SNAPSHOT.jar*

3. Использование Maven плагина

Spring Boot Maven плагин включает цель run, которую можно использовать для быстрой компиляции и запуска приложения. Приложения запускаются в разобранном виде, как в вашей IDE. В следующем примере показана типичная команда Maven для запуска приложения Spring Boot:

$ mvn spring-boot:run

Вы также можете использовать переменную среды операционной системы MAVEN\_OPTS, как показано в следующем примере:

$ export MAVEN\_OPTS=-Xmx1024m

4. Использование Gradle плагина

Spring Boot Gradle плагин также включает в себя задачу bootRun, которую можно использовать для запуска приложения в разобранном виде. Задача bootRun добавляется всякий раз, когда вы применяете плагины org.springframework.boot и java, и отображается в следующем примере:

$ gradle bootRun

Вы также можете использовать переменную среды операционной системы JAVA\_OPTS, как показано в следующем примере:

$ export JAVA\_OPTS=-Xmx1024m

**6. Требуется ли для запуска Spring Boot отдельный контейнер?**

Каждое Spring Boot web-приложение включает встроенный web-сервер. Разработчикам теперь не надо беспокоиться о настройке контейнера сервлетов и развертывании приложения на нем. Теперь приложение может запускаться само, как исполняемый jar-файл с использованием встроенного сервера

Если вам нужно использовать отдельный HTTP-сервер, для этого достаточно исключить зависимости по умолчанию. Spring Boot предоставляет отдельные starter-пакеты для разных HTTP-серверов

Создание автономных web-приложений со встроенными серверами не только удобно для разработки, но и является допустимым решением для приложений корпоративного уровня и становится все более полезно в мире микросервисов. Возможность быстро упаковать весь сервис (например, аутентификацию пользователя) в автономном и полностью развертываемом артефакте, который также предоставляет API – делает установку и развертывание приложения значительно проще.

**7. Как настроить приложение Spring Boot с помощью maven?**

С помощью File -> New -> Project… создаем новый проект.

В открывшемся окне, в левом боковом меню, выбираем Spring Initializr, выбираем Project SDK, опцию Initializr Service URL оставляем дефолтной.

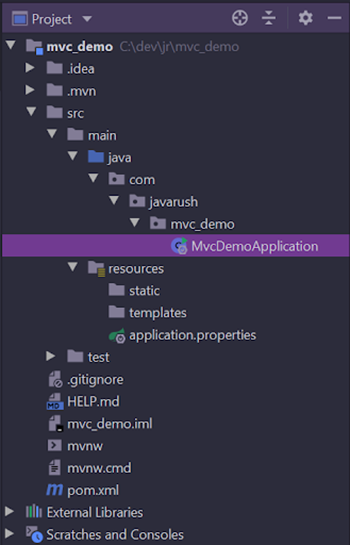
В следующем окне нам нужно выбрать параметры проекта. У нас будет Maven проект. Выбираем Type - Maven Project, заполняем Group и Artifact.

В следующем окне необходимо выбрать компоненты Spring Framework, который мы будем использовать. Нам достаточно двух:

Spring Web – компонента, которая позволит создавать Web приложения. Данная компонента включает в себя Spring MVC.

В следующем окне выбираем имя и расположение проекта в файловой системе:

После этого проект создан. Получается следующая структура проекта:

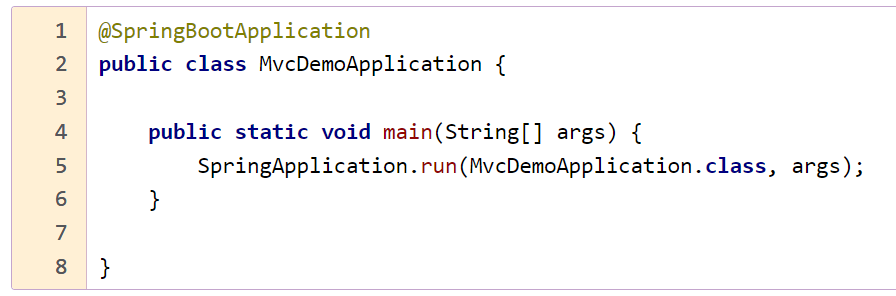


pom.xml – дескриптор развертывания, который позволяет быстро и просто импортировать библиотеки из разных фреймвроков в проект. Приложение собирается с помощью Maven, pom.xml – конфигурационный файл этой системы сборки.

Java класс – MvcDemoApplication. Это главный класс приложения, с которого будет запускаться spring-boot проект. Для запуска достаточно запустить метод main в данном классе.

Приведем код данного класса, а также файла pom.xml:

MvcDemoApplication:



pom.xml:

*<?xml version="1.0" encoding="UTF-8"?>*

*<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"*

*xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">*

*<modelVersion>4.0.0</modelVersion>*

*<parent>*

*<groupId>org.springframework.boot</groupId>*

*<artifactId>spring-boot-starter-parent</artifactId>*

*<version>2.2.2.RELEASE</version>*

*<relativePath/> <!-- lookup parent from repository -->*

*</parent>*

*<groupId>com.javarush</groupId>*

*<artifactId>mvc\_demo</artifactId>*

*<version>0.0.1-SNAPSHOT</version>*

*<name>mvc\_demo</name>*

*<description>Spring MVC Demo</description>*

*<properties>*

*<java.version>1.8</java.version>*

*</properties>*

*<dependencies>*

*<dependency>*

*<groupId>org.springframework.boot</groupId>*

*<artifactId>spring-boot-starter-thymeleaf</artifactId>*

*</dependency>*

*<dependency>*

*<groupId>org.springframework.boot</groupId>*

*<artifactId>spring-boot-starter-web</artifactId>*

*</dependency>*

*<dependency>*

*<groupId>org.springframework.boot</groupId>*

*<artifactId>spring-boot-starter-test</artifactId>*

*<scope>test</scope>*

*<exclusions>*

*<exclusion>*

*<groupId>org.junit.vintage</groupId>*

*<artifactId>junit-vintage-engine</artifactId>*

*</exclusion>*

*</exclusions>*

*</dependency>*

*</dependencies>*

*<build>*

*<plugins>*

*<plugin>*

*<groupId>org.springframework.boot</groupId>*

*<artifactId>spring-boot-maven-plugin</artifactId>*

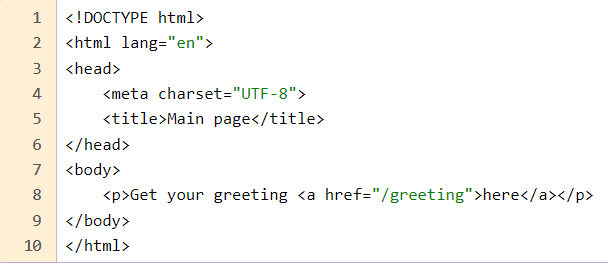
*</plugin>*

*</plugins>*

*</build>*

*</project>*

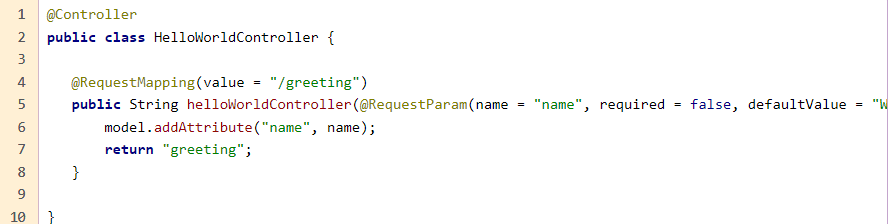
Далее создадим главную страницу приложения - index.html:



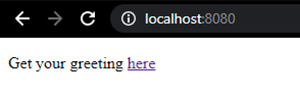
Затем создадим страницу greeting.html:



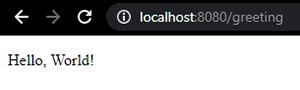
Внутри пакета mvc\_demo создадим пакет contoller, внутри которого создадим контроллер, HelloWorldController:



Для запуска, нужно запустить метод main в классе MvcDemoApplication. В логах запуска, можно увидеть, что web приложение стартовало на порту 8080. Здесь отобразиться страница index.html.



Можно перейти по ссылке на greetin. При этом переходе сработал контроллер.



**8. Какие встроенные серверы поддерживает Spring Boot?**

Spring Boot позволяет напрямую внедрять серверы, такие как Tomcat, Jetty и Undertow. Большинство разработчиков используют соответствующий «Starter» для получения полностью настроенного экземпляра. По умолчанию встроенный сервер прослушивает HTTP-запросы через порт 8080.

При использовании встроенного контейнера сервлетов вы можете зарегистрировать сервлеты, фильтры и всех прослушивателей (таких как HttpSessionListener) из спецификации Servlet либо с помощью bean-компонентов Spring, либо путем сканирования компонентов Servlet. Любой экземпляр Servlet, Filter или servlet \*Listener, являющийся компонентом Spring, регистрируется во встроенном контейнере.

**9. Для чего используются аннотации @RestController и @Controller?**

В Spring Boot класс контроллера отвечает за обработку входящих запросов REST API, подготовку модели и возврат представления для отображения в качестве ответа.

Классы контроллеров в Spring аннотируются аннотацией @Controller или @RestController. Они помечают классы контроллеров как обработчики запросов, чтобы Spring мог распознать их как службу RESTful во время выполнения.

***@Controller*** – это специализация универсальной стереотипной @Component, которая позволяет распознавать класс как управляемый Spring компонент.

@Controller расширяет возможности использования @Component и отмечает аннотированный класс как бизнес-уровень или уровень представления. Когда запрос сделан, это проинформирует DispatcherServlet о включении класса контроллера в сканирование методов, отображаемых аннотацией @RequestMapping.

***@RestController*** в Spring по сути представляют собой просто комбинацию @Controller и @ResponseBody . Эта аннотация была добавлена в Spring 4.0, чтобы удалить избыточность объявления @ResponseBody в вашем контроллере.

**10. Что такое Thymeleaf**

***Thymeleaf*** – современный серверный механизм Java-шаблонов для веб- и автономных сред, способный обрабатывать HTML, XML, JavaScript, CSS и даже простой текст.

Основной целью Thymeleaf является создание элегантного и удобного способа шаблонизации. Чтобы достичь этого, Thymeleaf основывается на концепции Natural Templates, чтобы внедрить свою логику в файлы шаблонов таким образом, чтобы этот шаблон не влиял на отображение прототипа дизайна. Это улучшает коммуникацию в команде и уменьшает разрыв между дизайнерско-программистскими группами.