**Контрольные вопросы**

**Общий вопрос:**

JDBC— платформенно независимый промышленный стандарт взаимодействия Java-приложений с различными [СУБД](https://ru.wikipedia.org/wiki/%D0%A1%D0%A3%D0%91%D0%94), реализованный в виде пакета java.sql, входящего в состав [Java SE](https://ru.wikipedia.org/wiki/Java_SE). JDBC основан на концепции так называемых драйверов, позволяющих получать соединение с [базой данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) по специально описанному [URL](https://ru.wikipedia.org/wiki/URL). Драйверы могут загружаться динамически (во время работы программы). Загрузившись, [драйвер](https://ru.wikipedia.org/wiki/%D0%94%D1%80%D0%B0%D0%B9%D0%B2%D0%B5%D1%80) сам регистрирует себя и вызывается автоматически, когда программа требует [URL](https://ru.wikipedia.org/wiki/URL), содержащий протокол, за который драйвер отвечает.

Java Persistence API (JPA) — спецификация [API](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9) [Java EE](https://ru.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition), предоставляет возможность сохранять в удобном виде Java-объекты в базе данных. Существует несколько реализаций этого интерфейса, одна из самых популярных использует для этого [Hibernate](https://ru.wikipedia.org/wiki/Hibernate_(%D0%B1%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0)). JPA реализует концепцию [ORM](https://ru.wikipedia.org/wiki/ORM).

Поддержка сохранности данных, предоставляемая JPA, покрывает области:

* непосредственно API, заданный в пакете [javax.persistence](http://docs.oracle.com/javaee/7/api/javax/persistence/package-summary.html);
* платформо-независимый объектно-ориентированный язык запросов [Java Persistence Query Language](https://ru.wikipedia.org/wiki/Java_Persistence_Query_Language);
* [метаинформация](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%B0%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D0%B5), описывающая связи между объектами.
* Генерация [DDL](https://ru.wikipedia.org/wiki/DDL) для сущностей

Hibernate — библиотека для языка программирования [Java](https://ru.wikipedia.org/wiki/Java), предназначенная для решения задач объектно-реляционного отображения ([ORM](https://ru.wikipedia.org/wiki/ORM)), самая популярная реализация спецификации [JPA](https://ru.wikipedia.org/wiki/Java_Persistence_API). 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](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85). Библиотека не только решает задачу связи классов Java с таблицами базы данных (и типов данных Java с типами данных [SQL](https://ru.wikipedia.org/wiki/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](https://ru.wikipedia.org/wiki/SQL)- и [JDBC](https://ru.wikipedia.org/wiki/JDBC)-кода.

Hibernate одна из самых популярных открытых реализаций последней версии спецификации. Даже скорее самая популярная, почти стандарт де-факто. То есть JPA только описывает правила и API, а Hibernate реализует эти описания, впрочем у Hibernate (как и у многих других реализаций JPA) есть дополнительные возможности, не описанные в JPA (и не переносимые на другие реализации JPA).

Разkичия между Hibernate и JDBC

1. Hibernate и JDBC Главное отличие заключается в том, что Hibernate сначала извлекает отображение объектов в кэше (т.е. операции Hibernate - это объекты), а JDBC является прямой операционной базой данных.

2, Hibernate - это легкий объектный пакет JDBC, который является отдельным объектом постоянным слоем рамы. Hibernate можно использовать в любом JDBC

3, Hibernate - это рамка, тесно связанная с JDBC, так что Hibernate совместимость и драйвер совместимости JDBC, и база данных имеет определенные отношения, но используют свою программу Java, нет никаких отношений с сервером App, и нет проблем со совместимости.,

4. Если технология JDBC используется правильно, эффективность его выполнения должна быть лучше, чем Hibernate, потому что Hibernate - это технология на основе JDBC.

5 JDBC использует оператор SQL, Hibernate использует оператор HQL, но оператор HQL в конечном итоге преобразуется в операторы SQL.

1. Традиционный способ развертывания – заставить класс Spring Boot Application @SpringBootApplication расширить класс SpringBootServletInitializer . Файл класса Spring Boot Servlet Initializer позволяет настроить приложение при его запуске с помощью контейнера сервлетов. Нам нужно расширить класс SpringBootServletInitializer для поддержки развертывания WAR-файла. Для Maven добавьте начальный класс в свойствах pom.xml. Мы должны обновить упаковочный JAR-файл в WAR, используя следующие фрагменты кода:

<packaging>war</packaging>

Для Maven добавьте стартовую зависимость Spring Boot в pom.xml, используя код:

<dependency>

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

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

</dependency>

Теперь создайте файл WAR для развертывания на сервере Tomcat. Теперь запустите сервер Tomcat и разверните файл WAR в каталоге webapps. После успешного развертывания, нажмите на URL в вашем веб-браузере http: // localhost: 8080 / …

1. Spring MVC - это фреймворк MVC, основанный на сервлетах. Он решает проблему разработки WEB с помощью Dispatcher Servlet и ModelAndView. Но его конфигурация громоздка, с большим количеством файлов .xml и .properties. И при использовании maven для управления проектами легко возникают конфликты пакетов jar.

Spring Boot - это пакет для быстрой разработки и интеграции на основе Spring, который включает не только Spring mvc, но также Spring JPA и Spring Security.  
реализует автоматическую настройку и снижает сложность построения проекта. А самодостаточный tomcat.pom.xml может быть настроен как стартовый, что упрощает настройку и решает проблему конфликтов пакетов jar.

1. jar и war – расширения платформо-независимых файлов-архивов Java-приложения. Кроме них есть еще третий формат, ear. Все эти форматы – на самом деле просто zip-архивы с классами и другими частями приложения. Вы можете переименовать такой файл в .zip, разархивировать, и посмотреть, что внутри.

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.

1. **Spring Boot** — это полезный проект, целью которого является упрощение создания приложений на основе Spring. Он позволяет наиболее простым способом создать web-приложение, требуя от разработчиков минимум усилий по его настройке и написанию кода.

**Spring Boot** обладает большим функционалом, но его наиболее значимыми особенностями являются: управление зависимостями, автоматическая конфигурация и встроенные контейнеры сервлетов.

1. Создайте Maven-проект в используемой вами IDE, назвав его SpringBootRestService.Обязательно используйте версию Java 8+, поскольку Spring Boot не работает с более ранними версиями.Вторым шагом необходимо настроить Spring Boot в файле pom.xml.

Все приложения Spring Boot конфигурируются от spring-boot-starter-parent, поэтому перед дальнейшим определением зависимостей, добавьте starter-parent следующим образом:

<parent>

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

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

<version>2.1.1.RELEASE</version>

</parent>

Т.к. мы создаем REST API, то необходимо в качестве зависимости использовать spring-boot-starter-web, которая неявно определяет все остальные зависимости, такие как spring-core, spring-web, spring-webmvc, servlet api, и библиотеку jackson-databind, поэтому просто добавьте в pom. xml следующее:

<dependency>

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

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

</dependency>

Следующий шаг — добавляем Spring Boot плагин:

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

Последний шаг — сделать так, чтобы Maven генерировал исполняемый jar-файл при сборке:

<packaging>jar</packaging>

Теперь мы собираемся создать контроллер платежей вместе с POJO-классами для запросов и ответов. А также класс, обрабатывающий базовый ответ, возвращаемый нашим сервисом. А теперь создадим контроллер.

Этот последний шаг заключается в создании класса конфигурации и запуска приложения. Spring Boot поддерживает новую аннотацию @SpringBootApplication, которая эквивалентна использованию @Configuration, @EnableAutoConfiguration и @ComponentScan с их атрибутами по умолчанию.

Таким образом, вам просто нужно создать класс, аннотированный с помощью @SpringBootApplication, а Spring Boot включит автоматическую настройку и отсканирует ваши ресурсы в текущем пакете:

package springboot.topjava.ru;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class Application {

public static void main(String[] args) {

SpringApplication.run(Application.class, args);

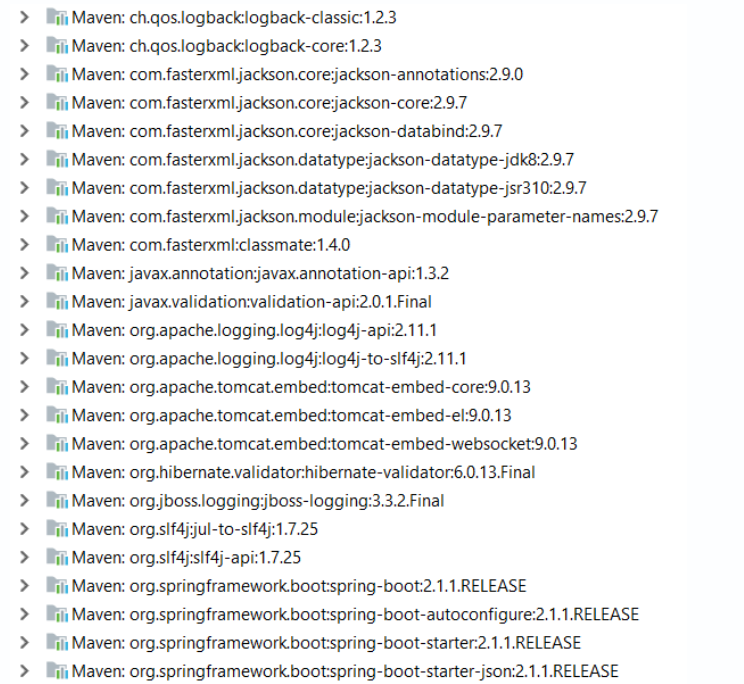
}

}

1. Да, требуется отдельный контейнер. Проект Spring Boot будет иметь встроенный tomcat, проект можно запустить только с помощью простой команды, но в производственной среде мы по-прежнему используем публикацию проекта в третьем контейнере сервлета. Изменение способа упаковки проекта на war:

<packaging>jar</packaging>

**7)** Создайте Maven-проект в используемой вами IDE, назвав его SpringBootRestService.Обязательно используйте версию Java 8+, поскольку Spring Boot не работает с более ранними версиями. Теперь следующие jar-библиотеки автоматически импортируются в ваш проект:



Следующий шаг — добавляем Spring Boot плагин:

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

Последний шаг — сделать так, чтобы Maven генерировал исполняемый jar-файл при сборке:

<packaging>jar</packaging>

**8)** Spring Boot включает поддержку следующих встроенных реактивных веб-серверов: Reactor Netty, 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 автоматической настройке сервера Reactor Netty или Jetty Spring Boot создаст определенные bean-компоненты, которые будут предоставлять HTTP-ресурсы экземпляру сервера: ReactorResourceFactory или JettyResourceFactory.

По умолчанию эти ресурсы также будут совместно использоваться клиентами Reactor Netty и Jetty для оптимальной производительности, учитывая:

* та же технология используется для сервера и клиента
* экземпляр клиента создается с помощью bean-компонента WebClient.Builder, автоматически настраиваемого Spring Boot

Разработчики могут переопределить конфигурацию ресурсов для Jetty и Reactor Netty, предоставив пользовательский компонент ReactorResourceFactory или JettyResourceFactory - это будет применяться как к клиентам, так и к серверам.

**9)** 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. Интерфейс RestController аннотируется @Controller и @ResponseBody вместо того, чтобы напрямую аннотировать его с помощью @Component .

Если мы заменим аннотацию нашего контроллера на @RestController , нам не нужно будет изменять домен и уровень сохраняемости, поскольку они все равно будут совместимы с этой аннотацией.

**10)** T**hymeleaf** — 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.

Thymeleaf также был разработан с самого начала с учетом стандартов Web, особенно **HTML5**, что позволяет вам создавать полностью соответствующие стандарту шаблоны.

Из коробки, Thymeleaf позволяет работать с шестью типами шаблонов, каждый из которых называется «Формат шаблона/Template Mode»:

* HTML
* XML
* TEXT
* JAVASCRIPT
* CSS
* RAW

**Thymeleaf** — чрезвычайно расширяемый механизм (на самом деле его можно назвать платформой шаблонов), который позволяет вам определять и настраивать способ обработки ваших шаблонов до тонкого уровня детализации.