75 вопросов и ответов на собеседование по Spring

Read & Write Code

1). Что такое Spring Framework?

Spring Framework (или просто **Spring**) - это популярный и мощный фреймворк для разработки приложений на языке Java. Он предоставляет комплексный набор инструментов и функций для упрощения разработки Java-приложений, особенно в области создания **веб-приложений** и **веб-сервисов**. Spring был создан для решения проблем, связанных с разработкой сложных и масштабируемых приложений, и он стал одним из самых популярных фреймворков для разработки на Java.

Spring Framework является открытым исходным кодом и имеет большое сообщество разработчиков, что делает его надежным и активно поддерживаемым фреймворком для создания разнообразных Java-приложений.

2). Из каких частей состоит Spring Framework?

Вот основные модули:

- **Основной контейнер** – предоставляет основной функционал Spring. Главным компонентом контейнера является BeanFactory – реализация паттерна Фабрика. BeanFactory позволяет разделить конфигурацию приложения и информацию о зависимостях от кода.
- **Spring context** – конфигурационный файл, который предоставляет информация об окружающей среде для Spring. Сюда входят такие **enterprise**-сервисы, как **JNDI**, **EJB**, интернационализация, валиадция и т.п.
- **Spring AOP** – отвечает за интеграцию аспектно-ориентированного программирования во фреймворк. Spring AOP обеспечивает сервис управления транзакциями для Spring-приложения.
- **Spring DAO** – абстрактный уровень Spring JDBC DAO предоставляет иерархию исключений и множество сообщений об ошибках для разных БД. Эта иерархия упрощает обработку исключений и значительно уменьшает количество кода, которое вам нужно было бы написать для таких операций, как,например,открытие и закрытие соединения.
- **Spring ORM** – отвечает за интеграцию Spring и таких популярных ORM-фреймворков, как Hibernate, iBatis и JDO.
- **Spring Web module** – отвечает за context web-приложений.
- **Spring MVC framework** – реализация паттерна MVC для построения Web-приложений.

Read & Write Code

3). В чем преимущества Spring?

Вот некоторые из ключевых особенностей Spring Framework:

1.**IoC** (Inversion of Control): Spring предлагает механизм инверсии управления, который позволяет управлять жизненным циклом и зависимостями объектов в приложении. **Контейнер** Spring заботится о связывании зависимостей различных объектов вместо создания или поиска зависимых объектов.Это делает код более гибким и управляемым.

2.**DI** (Dependency Injection): Spring поддерживает внедрение зависимостей, что упрощает создание и управление зависимостями между компонентами приложения.

3.Аспектно-ориентированное программирование (**AOP**): Spring предоставляет поддержку AOP для реализации перехватчиков (aspects), таких как логирование, безопасность и другие аспекты приложения.

4.**Модульность**: Spring разделен на модули, что позволяет вам выбирать только необходимые части фреймворка для вашего проекта.

5.**Поддержка транзакций**: Spring предоставляет механизм управления транзакциями в приложении, что делает его отличным выбором для работы с базами данных.

6.**Поддержка веб-приложений**: Spring **MVC** - часть Spring Framework, предоставляющая мощные средства для разработки веб-приложений и API.

7.**Тестирование**: Spring обеспечивает хорошую поддержку для тестирования приложений с помощью модулей, таких как Spring Test и JUnit.

4). Объясните суть паттерна DI или IoC

Dependency injection (DI) – паттерн проектирования и архитектурная модель, так же известная как **Inversion of Control (IoC)**.
В полном соответствии с принципом единой обязанности объект отдаёт заботу о построении требуемых ему зависимостей внешнему, специально предназначенному для этого общему механизму.
Суть DI заключается в том, что объекты не создают свои зависимости сами, а вместо этого получают их из внешних источников. DI описывает ситуацию, когда один объект реализует свой функционал через другой объект. Например, соединение с базой данных передается конструктору объекта через аргумент, вместо того чтобы конструктор сам устанавливал соединение. Термин “dependency injection” немного неправильный, т.к. внедряются не зависимости, а функциональность или ресурсы. Существуют три формы внедрения (но не типа) зависимостей: **сэттер, конструктор** и **внедрение путем интерфейса**.

DI – это способ достижения **слабой связанности**. IoC предоставляет возможность объекту получать ссылки на свои зависимости. Обычно это реализуется через **lookup**-метод. Преимущество IoC в том, что эта модель позволяет отделить объекты от реализации механизмов, которые он использует.
В результате мы получаем **большую гибкость** как при разработке приложений, так и при их тестировании.

5). Как реализуется DI в Spring Framework?

Реализация **DI** в Spring основана на **двух** ключевых концепциях Java — компонентах **JavaBean** и **интерфейсах**.

При использовании Spring в качестве поставщика DI вы получаете гибкость определения конфигурации зависимостей внутри своих приложений разнообразными путями (т.е. с помощью XML-файлов, с помощью конфигурационных Java классов Spring или посредством аннотаций Java в коде).

Компоненты JavaBean (также называемые **POJO** (Plain Old Java Object — простой старый объект Java)) предоставляют стандартный механизм для создания ресурсов Java, которые являются конфигурируемыми множеством способов.
За счет применения DI объем кода, который необходим при проектировании приложения на основе интерфейсов, снижается почти до нуля.
Кроме того, с помощью интерфейсов можно получить **максимальную** отдачу от DI, потому что бины могут использовать любую реализацию интерфейса для удовлетворения их зависимости.

Read & Write Code

5). Продолжение

К типам реализации внедрения зависимостей в Spring относят:

Constructor Dependency Injection — это тип внедрения зависимостей, при котором зависимости компонента предоставляются ему в его конструкторе (или конструкторах).

```
public class ConstructorInjection {

private Dependency dependency;

  public ConstructorInjection(Dependency dependency) {
  this.dependency = dependency;
  }
}
```

Setter Dependency Injection — контейнер IoC внедряет зависимости компонента в компонент через методы установки в стиле JavaBean.

```
public class SetterInjection {
private Dependency dependency;
  public void setDependency(Dependency dependency) {
  this.dependency = dependency;
  }
}
```

6). Объясните работу BeanFactory в Spring

BeanFactory – это реализация паттерна Фабрика, является основным интерфейсом, отвечающим за создание и управление бинами в Spring-приложениях. Он служит как контейнер для бинов. BeanFactory обычно настраивается с помощью конфигурационных файлов, таких как XML-файлы, аннотации или Java-конфигурации. Он понимает, какие бины должны быть созданы и как они связаны друг с другом.
Один из ключевых моментов - **ленивая инициализация**. BeanFactory создает бины только в тот момент, когда они действительно нужны, что способствует оптимизации использования ресурсов.
Существует несколько реализаций BeanFactory, самая используемся – “**org.springframework.beans.factory.xml.XmlBeanFactory**”. Она загружает бины на основе конфигурационного XML-файла. Чтобы создать XmlBeanFactory передайте конструктору InputStream, например:

```
BeanFactory factory = new XmlBeanFactory(new FileInputStream("myBean.xml"));
```

После этой строки фабрика знает о бинах, но их экземпляры еще не созданы. Чтобы инстанцировать бин нужно вызвать метод getBean().

Например:

```
myBean bean1 = (myBean)factory.getBean("myBean");
```

Read & Write Code

7). В чем роль ApplicationContext в Spring?

В то время, как BeanFactory используется в простых приложениях, **Application Context** – это более сложный контейнер. Как и BeanFactory он может быть использован для загрузки и связывания бинов, но еще он предоставляет:

1. возможность получения текстовых сообщений, в том числе поддержку интернационализации;
2. общий механизм работы с ресурсами;
3. события для бинов, которые зарегестрированы как слушатели.

Из-за большей функциональности **рекомендуется** использование Application Context вместо BeanFactory. Последний используется только в случаях нехватки ресурсов, например при разработке для мобильных устройств.

Существует 3 основных реализации Application context:

1. **ClassPathXmlApplicationContext** – получает информацию из xml-файла, находящегося в classpath. Пример: ApplicationContext context = new ClassPathXmlApplicationContext(“bean.xml”);
2. **FileSystemXmlApplicationContext** – получает информацию из xml-файла. Например: ApplicationContext context = new FileSystemXmlApplicationContext(“bean.xml”);
3. **XmlWebApplicationContext** – получает информацию из xml-файла за пределами web-приложения.

Read & Write Code

8). Что такое AOP? Как это относиться к IoC?

AOP (Aspect-Oriented Programming) - это методология программирования, которая позволяет выделять и абстрагировать "аспекты" поведения приложения, такие как логирование, безопасность, транзакции и другие поперечно-распределенные задачи, и внедрять их в приложение независимо от основной логики программы.
AOP решает проблему размещения кода, который не является частью бизнес-логики, внутри классов, что может сделать их менее читаемыми и сложными для поддержки. Вместо этого, вы можете создать аспекты, которые содержат этот поперечно-распределенный код, и затем применять эти аспекты к классам или методам, где он должен выполняться.
Как это связано с IoC (Inversion of Control)? Внедрение зависимостей (DI) и управление контейнером (IoC) являются фундаментальными концепциями Spring, которые позволяют разделить создание объектов и их зависимостей от основной логики приложения. AOP в Spring интегрируется с DI и IoC для обеспечения более гибкого управления поведением вашего приложения.
Как это работает:

1. **Аспекты**: В Spring AOP вы можете создать аспекты, которые описывают поведение, которое вы хотите выделить. Например, аспект для логирования.
2. Цели (**Targets**): Цели - это компоненты вашего приложения, к которым вы хотите применить аспект. Это могут быть классы или методы.
3. Советы (**Advices**): Аспекты содержат советы, которые представляют собой фрагменты кода, выполняющиеся в определенные моменты времени, например, до выполнения метода, после выполнения метода и так далее.
4. Точки среза (**Pointcuts**): Точки среза определяют, когда и где должны выполняться советы. Они определяют условия для применения аспекта.

Read & Write Code

9). Что такое DAO?

Spring предоставляет различные возможности и механизмы для поддержки **DAO (Data Access Object)** в приложениях. DAO - это шаблон проектирования, который используется для абстрагирования доступа к данным и обеспечения уровня абстракции между приложением и базой данных. Вот как Spring поддерживает DAO:

1). JDBC Template: Spring предоставляет класс JdbcTemplate, который упрощает работу с JDBC (Java Database Connectivity). Он обрабатывает открытие и закрытие соединений с базой данных, управление транзакциями и обработку исключений. Вы можете использовать JdbcTemplate для выполнения SQL-запросов и обновления данных в базе данных.
Пример:

```
@Autowired
private JdbcTemplate jdbcTemplate;

public void insertData(String data) {
 jdbcTemplate.update("INSERT INTO my_table (data) VALUES (?)", data);
 }
```

2). Интеграция с **ORM** (Object-Relational Mapping): Spring обеспечивает интеграцию с различными ORM-фреймворками, такими как Hibernate, JPA (Java Persistence API), и JDO (Java Data Objects). Это позволяет вам создавать DAO, которые работают с объектами Java, а не с SQL-запросами напрямую. Spring управляет жизненным циклом сессии или Entity Manager'a и обеспечивает транзакционность.
Пример с Hibernate:

```
@Repository
public class MyEntityDao {
 @Autowired
 private SessionFactory sessionFactory;

 public void save(MyEntity entity) {
 Session session = sessionFactory.getCurrentSession();
session.save(entity);
   }
}
```

9). Продолжение

3). Декларативная транзакционность: Spring позволяет определять транзакции для методов DAO с помощью аннотаций или XML-конфигурации. Это обеспечивает управление транзакциями на уровне методов без необходимости явно управлять транзакциями.
Пример с аннотациями:

```
@Transactional
public class MyEntityService {
 @Autowired
 private MyEntityDao entityDao;

  public void saveEntity(MyEntity entity) {
entityDao.save(entity);
   }
}
```

4). **Поддержка исключений**: Spring предоставляет возможность определять, какие исключения должны считаться checked или unchecked (что влияет на поведение транзакции) при работе с DAO.

5). Поддержка JNDI и DataSource: Spring упрощает настройку и использование источников данных (DataSource), что делает его более гибким при переносе приложения между разными средами.

10). Что такое Been?

Spring Beans — это объекты Java, которые инициализируются контейнером Spring IoC. Spring контейнер обеспечивает удобный способ управления их жизненным циклом и конфигурацией.

Read & Write Code

11). Какие вы знаете различные scope у Spring Bean?

Какие вы знаете различные scope у Spring Bean?

В Spring предусмотрены различные области времени действия бинов:

1. **singleton** — может быть создан только один экземпляр бина. Этот тип используется спрингом по умолчанию, если не указано другое. Следует осторожно использовать публичные свойства класса, т.к. они не будут потокобезопасными.
2. **prototype** — создается новый экземпляр при каждом запросе.
3. **request** — аналогичен prototype, но название служит пояснением к использованию бина в веб приложении. Создается новый экземпляр при каждом HTTP request.
4. **session** — новый бин создается в контейнере при каждой новой HTTP сессии.
5. **global-session**: используется для создания глобальных бинов на уровне сессии для Portlet приложений.

По умолчанию Spring Bean инициализируется как **singleton**.

12) Что такое жизненный цикл Bean?

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

```
 @PostConstruct
 public void init(){
 System.out.println("Bean init method called");
}
```

```
@PreDestroy
 public void destroy(){
 System.out.println("Bean destroy method called");
}
```

Или через xml конфигурацию:

```
<bean name="myBean" class="ru.readandwritecode.spring.MyBean"
 init-method="init" destroy-method="destroy">
 <property name="someProp" ref="someProp"></property>
</bean>
```

13). Как получить объекты ServletContext и ServletConfig внутри Spring Bean?

Доступны два способа для получения основных объектов контейнера внутри бина:

- Реализовать один из Spring*Aware (ApplicationContextAware, ServletContextAware, ServletConfigAware и др.) интерфейсов.
- Использовать автоматическое связывание @Autowired в спринг. Способ работает внутри контейнера спринг.

```
@Autowired
ServletContext servletContext;
```

14). Являются ли Singleton Beans потокобезопасными?

Нет, singleton bean-компоненты **не** потокобезопасны, поскольку потокобезопасность связана с выполнением, тогда как singleton — это шаблон проектирования, ориентированный на создание.

Безопасность потоков зависит только от самой реализации компонента.

15). Перечислите некоторые важные аннотации

Важные аннотации:

@Required;
@Autowired;
@Qualifier;
@Resource;
@PostConstruct;
@PreDestroy;
@Component;
@Controller;
@Repository;
@Service;
@Transactional.

16). Как выглядит типичная реализция метода используя Spring?

Для типичного Spring-приложения нам необходимы следующие файлы:

- Интерфейс, описывающий функционал приложения
- Реализация интерфейса, содержащая свойства, сэттеры-гэттеры, функции и т.п.
- Конфигурационный XML-файл Spring'a.
- Клиентское приложение, которое использует функцию.

17). Что такое связывание в Spring и расскажите об аннотации @Autowired?

Процесс внедрения зависимостей в бины при инициализации называется Spring Bean Wiring.
Считается хорошей практикой задавать явные связи между зависимостями, но в Spring предусмотрен дополнительный механизм связывания **@Autowired**. Аннотация может использоваться над **полем** или **методом** для связывания по типу. Чтобы аннотация заработала, необходимо указать небольшие настройки в конфигурационном файле спринг с помощью элемента **context:annotation-config.**

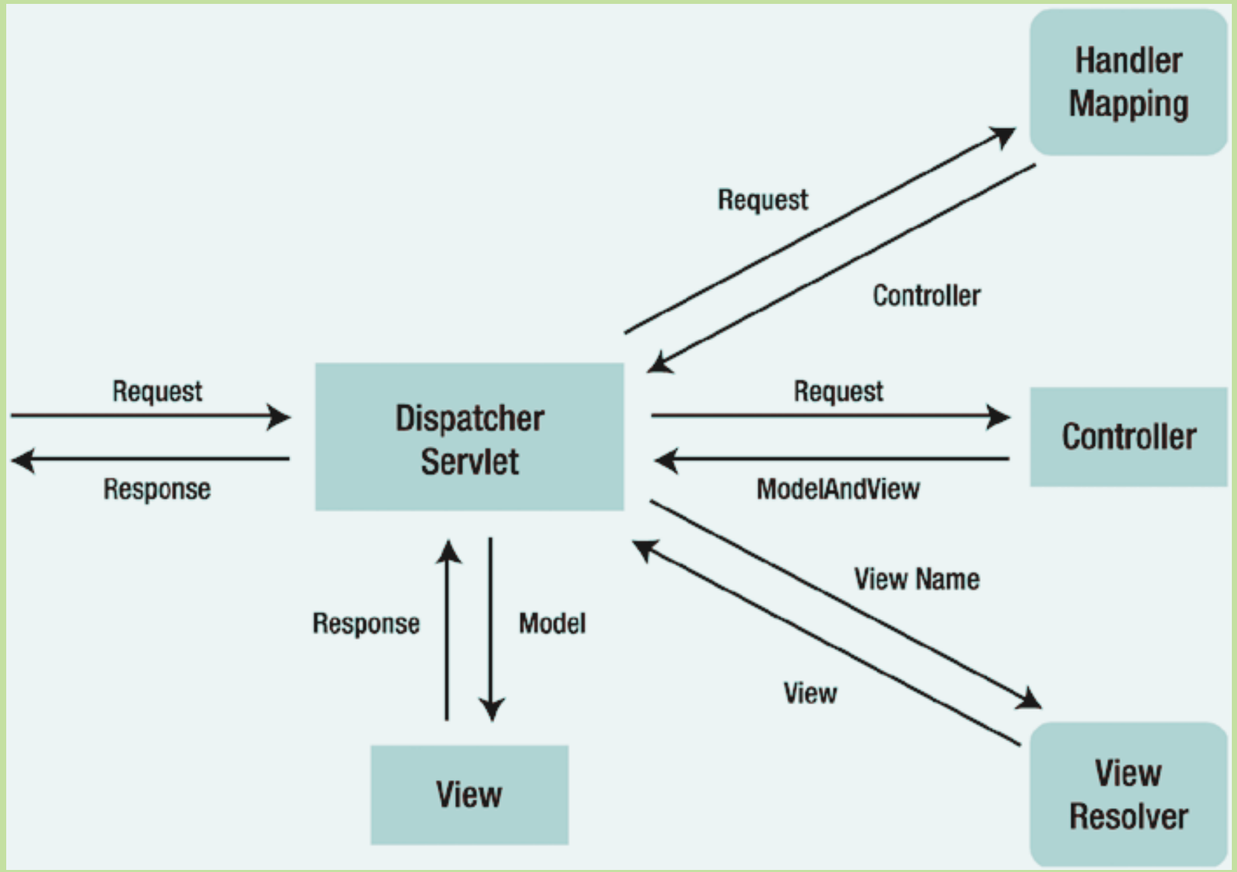
18). Какие различные типы автоматического связывания в Spring?

Существует четыре вида связывания в спринг:

- autowire byName,
- autowire byType,
- autowire by constructor,
- autowiring by @Autowired and @Qualifier annotations

19). Что такое контроллер в Spring MVC?

Ключевым интерфейсом в Spring MVC является **Controller**. Контроллер обрабатывает запросы к действиям, осуществляемые пользователями в пользовательском интерфейсе, взаимодействуя с уровнем обслуживания, обновляя модель и направляя пользователей на соответствующие представления в зависимости от результатов выполнения. Controller — **управление, связь между моделью и видом.**



Проще говоря, все запросы, обрабатываемые **DispatcherServlet** , направляются к классам, аннотированным **@Controller** . Каждый класс контроллера сопоставляет один или несколько запросов с методами, которые обрабатывают и выполняют запросы с предоставленными входными данными.

20). Какая разница между аннотациями @Component, @Repository и @Service в Spring?

@Component используется для указания класса в качестве компонента спринг. При использовании поиска аннотаций, такой класс будет сконфигурирован как spring bean.

@Controller специальный тип класса, применяемый в MVC приложениях. Обрабатывает запросы и часто используется с аннотацией @RequestMapping.

@Repository указывает, что класс используется для работы с поиском, получением и хранением данных. Аннотация может использоваться для реализации шаблона DAO.

@Service указывает, что класс является сервисом для реализации бизнес логики (на самом деле не отличается от Component, но просто помогает разработчику указать смысловую нагрузку класса).

Для указания контейнеру на класс-бин можно использовать любую из этих аннотаций. Но различные имена позволяют различать назначение того или иного класса.

21). Расскажите, что вы знаете о DispatcherServlet и ContextLoaderListener

DispatcherServlet — сервлет диспетчера. Этот сервлет анализирует запросы и направляет их соответствующему контроллеру для обработки. В Spring MVC класс DispatcherServlet является центральным **сервлетом**, который получает запросы и направляет их соответствующим **контроллерам**. В приложении Spring MVC может существовать произвольное количество экземпляров DispatcherServlet, предназначенных для разных целей (например, для обработки запросов пользовательского интерфейса, запросов веб-служб REST и т.д.). Каждый экземпляр DispatcherServlet имеет собственную конфигурацию **WebApplicationContext**, которая определяет характеристики уровня сервлета, такие как контроллеры, поддерживающие сервлет, отображение обработчиков, распознавание представлений, интернационализация, оформление темами, проверка достоверности, преобразование типов и форматирование и т.п.
ContextLoaderListener — слушатель при старте и завершении корневого класса Spring WebApplicationContext. Основным назначением является связывание жизненного цикла ApplicationContext и ServletContext, а так же автоматического создания ApplicationContext. Можно использовать этот класс для доступа к бинам из различных контекстов спринг. Настраивается в **web.xml**:

```
<context-param>
   <param-name>contextConfigLocation</param-name>
   <param-value>/WEB-INF/spring/root-context.xml</param-value>
</context-param>

<listener>
   <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
```

22). Что такое ViewResolver в Spring?

ViewResolver — распознаватель представлений. Интерфейс ViewResolver в Spring MVC (из пакета **org.springframework.web.servlet**) поддерживает распознавание представлений на основе логического имени, возвращаемого контроллером. Для поддержки различных механизмов распознавания представлений предусмотрено множество классов реализации. Например, класс **UrlBasedViewResolver** поддерживает прямое преобразование логических имен в URL.
Класс **ContentNegotiatingViewResolver** поддерживает динамическое распознавание представлений в зависимости от типа медиа, поддерживаемого клиентом (XML, PDF, JSON и т.д.). Существует также несколько реализаций для интеграции с различными технологиями представлений, такими как FreeMarker (**FreeMarkerViewResolver**), Velocity (**VelocityViewResolver**) и JasperReports (**JasperReportsViewResolver**).

```
<!-- Resolves views selected for rendering by @Controllers to .jsp resources
        in the /WEB-INF/views directory -->
    <bean
          class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/views/" />
        <property name="suffix" value=".jsp" />
    </bean>
```

InternalResourceViewResolver — реализация **ViewResolver**, которая позволяет находить представления, которые возвращает контроллер для последующего перехода к нему. Ищет по заданному пути, префиксу, суффиксу и имени.

23). Что такое MultipartResolver и когда его использовать?

Интерфейс **MultipartResolver** используется для загрузки файлов. Существуют две реализации: **CommonsMultipartResolver** и **StandardServletMultipartResolver**, которые позволяют фреймворку загружать файлы. По умолчанию этот интерфейс не включается в приложении и необходимо указывать его в файле конфигурации. После настройки любой запрос о загрузке будет отправляться этому интерфейсу.

```
<beans:bean id="multipartResolver"
      class="org.springframework.web.multipart.commons.CommonsMultipartResolver">

      <!-- setting maximum upload size -->
      <beans:property name="maxUploadSize" value="100000" />

 </beans:bean>
```

24). Как обрабатывать исключения в Spring MVC Framework?

В Spring MVC интерфейс **HandlerExceptionResolver** (из пакета **org.springframework.web.servlet**) предназначен для работы с непредвиденными исключениями, возникающими во время выполнения обработчиков.

По умолчанию **DispatcherServlet** регистрирует класс **DefaultHandlerExceptionResolver** (из пакета **org.springframework.web.servlet.mvc.support**).
Этот распознаватель обрабатывает определенные стандартные исключения Spring MVC, устанавливая специальный код состояния ответа. Можно также реализовать собственный обработчик исключений, аннотировав метод контроллера с помощью аннотации **@ExceptionHandler** и передав ей в качестве атрибута тип исключения. В общем случае обработку исключений можно описать таким образом:

Controller Based – указать методы для обработки исключения в классе контроллера. Для этого нужно пометить такие методы аннотацией **@ExceptionHandler**.

Global Exception Handler – для обработки глобальных исключений спринг предоставляет аннотацию **@ControllerAdvice**.

HandlerExceptionResolver implementation – Spring Framework предоставляет интерфейс **HandlerExceptionResolver**, который позволяет задать глобального обработчика исключений. Реализацию этого интерфейса можно использовать для создания собственных глобальных обработчиков исключений в приложении.

25). Можем ли мы иметь несколько файлов конфигурации Spring?

С помощью указания contextConfigLocation можно задать несколько файлов конфигурации Spring. Параметры указываются через **запятую** или **пробел**:

```
<servlet>
   <servlet-name>appServlet</servlet-name>
   <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
   <init-param>
     <param-name>contextConfigLocation</param-name>
     <param-value>/WEB-INF/spring/appServlet/servlet-context.xml,/WEB-INF/spring/appServlet/servlet-jdbc.xml</param-value>
   </init-param>
   <load-on-startup>1</load-on-startup>
</servlet>
```

Поддерживается возможность указания нескольких корневых файлов конфигурации Spring:

```
<context-param>
   <param-name>contextConfigLocation</param-name>
   <param-value>/WEB-INF/spring/root-context.xml /WEB-INF/spring/root-security.xml</param-value>
</context-param>
```

Файл конфигурации можно импортировать:

```
<beans:import resource="spring-jdbc.xml"/>
```

26). Какие минимальные настройки нужны, чтобы создать приложение Spring MVC?

Для создания простого Spring MVC приложения необходимо пройти следующие шаги:

- Добавить зависимости spring-context и spring-webmvc в проект.
- Указать DispatcherServlet в web.xml для обработки запросов внутри приложения.
- Задать определение spring bean (аннотацией или в xml). Добавить определение view resolver для представлений.
- Настроить класс контроллер для обработки клиентских запросов.

27). Как бы вы связали Spring MVC Framework и архитектуру MVC?

- Моделью (**Model**) выступает любой Java bean в Spring. Внутри класса могут быть заданы различные атрибуты и свойства для использования в представлении.
- Преставление (**View**) — JSP страница, HTML файл и т.п. служат для отображения необходимой информации пользователю. Представление передает обработку запросов к диспетчеру сервлетов (контроллеру).
- DispatcherServlet (**Controller**) — это главный контроллер в приложении Spring MVC, который обрабатывает все входящие запросы и передает их для обработки в различные методы в контроллеры.

28). Как добиться локализации в приложениях Spring MVC?

Spring MVC предоставляет очень простую и удобную возможность локализации приложения. Для этого необходимо сделать следующее:

- Создать файл resource bundle, в котором будут заданы различные варианты локализированной информации.
- Определить messageSource в конфигурации Spring используя классы ResourceBundleMessageSource или ReloadableResourceBundleMessageSource.
- Определить localeResolver класса CookieLocaleResolver для включения возможности переключения локали.
- С помощью элемента spring:message DispatcherServlet будет определять в каком месте необходимо подставлять локализированное сообщение в ответе.

```
<beans:bean id="messageSource"
   class="org.springframework.context.support.ReloadableResourceBundleMessageSource">
   <beans:property name="basename" value="classpath:messages" />
   <beans:property name="defaultEncoding" value="UTF-8" />
</beans:bean>

<beans:bean id="localeResolver"
   class="org.springframework.web.servlet.i18n.CookieLocaleResolver">
   <beans:property name="defaultLocale" value="en" />
   <beans:property name="cookieName" value="myAppLocaleCookie"></beans:property>
   <beans:property name="cookieMaxAge" value="3600"></beans:property>
</beans:bean>

<interceptors>
   <beans:bean
      class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">
      <beans:property name="paramName" value="locale" />
   </beans:bean>
</interceptors>
```

29). Как мы можем использовать Spring для создания веб-службы RESTful, возвращающей JSON?

Spring Framework позволяет создавать Restful веб сервисы и возвращать данные в формате JSON. Spring обеспечивает интеграцию с Jackson JSON API для возможности отправки JSON ответов в restful web сервисе. Для отправки ответа в формате JSON из Spring MVC приложения необходимо произвести следующие настройки:

- Добавить зависимости Jackson JSON. С помощью maven это делается так:

```
<!-- Jackson -->
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>${jackson.databind-version}</version>
</dependency>
```

29). Продолжение

- Настроить бин RequestMappingHandlerAdapter в файле конфигурации Spring и задать свойство messageConverters на использование бина MappingJackson2HttpMessageConverter.

```
<!-- Configure to plugin JSON as request and response in method handler -->
<beans:bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter">
    <beans:property name="messageConverters">
        <beans:list>
            <beans:ref bean="jsonMessageConverter"/>
        </beans:list>
    </beans:property>
</beans:bean>

<!-- Configure bean to convert JSON to POJO and vice versa -->
<beans:bean id="jsonMessageConverter"
class="org.springframework.http.converter.json.MappingJackson2HttpMessageConverter">
</beans:bean>
```

- В контроллере указать с помощью аннотации @ResponseBody возвращение Object:

```
@RequestMapping(value = EmpRestURIConstants.GET_EMP, method = RequestMethod.GET)
public @ResponseBody Employee getEmployee(@PathVariable("id") int empId) {
    logger.info("Start getEmployee. ID="+empId);

    return empData.get(empId);
}
```

30). Можем ли мы послать объект как ответ метода обработчика контроллера?

Да, это возможно. Для этого используется аннотация **@ResponseBody**. Так можно отправлять ответы в виде JSON, XML в restful веб сервисах.

31). Как проверить (валидировать) данные формы в Spring Web MVC Framework?

Spring поддерживает аннотации валидации из JSR-303, а так же возможность создания своих реализаций классов валидаторов. Пример использования аннотаций:

```
@Size(min=2, max=30)
private String name;

@NotEmpty @Email
private String email;

@NotNull @Min(18) @Max(100)
private Integer age;

@NotNull
private Gender gender;

@DateTimeFormat(pattern="MM/dd/yyyy")
@NotNull @Past
private Date birthday;
```

32). Что вы знаете о Spring MVC Interceptor и как он используется?

Перехватчики в Spring (**Spring Interceptor**) являются аналогом **Servlet Filter** и позволяют перехватывать запросы клиента и обрабатывать их. Перехватить запрос клиента можно в трех местах: preHandle, postHandle и afterCompletion.

- **preHandle** — метод используется для обработки запросов, которые еще не были переданы в метода обработчик контроллера. Должен вернуть **true** для передачи следующему перехватчику или в handler method. **False** укажет на обработку запроса самим обработчиком и отсутствию необходимости передавать его дальше. Метод имеет возможность выкидывать исключения и пересылать ошибки к представлению.
- **postHandle** — вызывается после handler method, но до обработки **DispatcherServlet** для передачи представлению. Может использоваться для добавления параметров в объект **ModelAndView**.
- **afterCompletion** — вызывается после отрисовки представления.

Для создания обработчика необходимо расширить абстрактный класс HandlerInterceptorAdapter или реализовать интерфейс HandlerInterceptor. Так же нужно указать перехватчики в конфигурационном файле Spring.

```
<!-- Configuring interceptors based on URI -->
   <interceptors>
      <interceptor>
         <mapping path="/home" />
         <beans:bean class="ru.readandwritecode.spring.RequestProcessingTimeInterceptor"></beans:bean>
      </interceptor>
   </interceptors>
```

33). Spring JdbcTemplate класс и его применение

Spring предоставляет отличную поддержку JDBC API и предлагает класс утилиту JdbcTemplate, с помощью которого можно избавиться от многократного повторения похожего кода в приложении (вроде операций **open \ closing connection**; **ResultSet**, **PreparedStatement** и др.). Для подключения необходимо настроить файл конфигурации spring и получить объект **JdbcTemplate**. Например.
spring.xml:

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

   <bean id="employeeDAO" class="com.readandwritecode.spring.jdbc.dao.EmployeeDAOImpl">
      <property name="dataSource" ref="dataSource" />
   </bean>

   <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">

      <property name="driverClassName" value="com.mysql.jdbc.Driver" />
      <property name="url" value="jdbc:mysql://localhost:3306/TestDB" />
      <property name="username" value="rwc" />
      <property name="password" value="rwc123" />
   </bean>

</beans>
```

33). Продолжение

Пример использования **JdbcTemplate**:

```
@Override
  public void save(Employee employee) {
     String query = "insert into Employee (id, name, role) values (?,?,?)";

     JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);

     Object[] args = new Object[] {employee.getId(), employee.getName(),employee.getRole()};

     int out = jdbcTemplate.update(query, args);

     if(out !=0){
        System.out.println("Employee saved with id="+employee.getId());
     }else System.out.println("Employee save failed with id="+employee.getId());
  }
```

34). Как использовать Tomcat JNDI DataSource в веб-приложении Spring?

Для использования контейнера сервлетов настроенного на использование **JNDI DataSource**, необходимо задать соответствующее свойство в файле конфигурации и затем **внедрять его как зависимость**. Далее мы можем использовать объект **JdbcTemplate** для выполнения операций с базами данных.

```
<beans:bean id="dbDataSource" class="org.springframework.jndi.JndiObjectFactoryBean">
    <beans:property name="jndiName" value="java:comp/env/jdbc/MyLocalDB"/>
</beans:bean>
```

35). Каким образом можно управлять транзакциями в Spring?

Транзакциями в Spring управляют с помощью Declarative Transaction Management (программное управление). Используется аннотация **@Transactional** для описания необходимости управления транзакцией. В файле конфигурации нужно добавить настройку **transactionManager** для DataSource.

```
<bean id="transactionManager"
    class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource" />
</bean>
```

36). Расскажите о Spring DAO

Spring DAO предоставляет возможность работы с доступом к данным с помощью технологий вроде JDBC, Hibernate в удобном виде. Существуют специальные классы: **JdbcDaoSupport**, **HibernateDaoSupport**, **JdoDaoSupport**, **JpaDaoSupport**.
В Spring DAO поддерживается иерархия исключений, что помогает не обрабатывать некоторые исключения.

```
import java.util.List;
import org.springframework.jdbc.core.support.JdbcDaoSupport;

public class PersonDao extends JdbcDaoSupport{

public void insert(Person person){
String insertSql ="INSERT INTO PERSON (NAME, EMAIL) VALUES(?,?);";
String name = person.getName();
String email = person.getEmail();

getJdbcTemplate().update(insertSql,new Object[]{name,email});
}

public List<Person> selectAll(){
String selectAllSql = "SELECT * FROM PERSON;";

return getJdbcTemplate().query(selectAllSql, new PersonRowMapper());
}

}
```

37). Как внедрить java.util.Properties в Spring Bean?

Для возможности использования Spring для внедрения свойств (properties) в различные бины необходимо определить propertyConfigure bean, который будет загружать файл свойств.

```
<bean id="propertyConfigurer"
  class="org.springframework.context.support.PropertySourcesPlaceholderConfigurer">
    <property name="location" value="/WEB-INF/application.properties" />
</bean>

<bean class="com.readandwritecode.spring.EmployeeDaoImpl">
    <property name="maxReadResults" value="${results.read.max}"/>
</bean>
```

Или через аннотации:

```
@Value("${maxReadResults}")
private int maxReadResults;
```

38). Как задаются файлы маппинга Hibernate в Spring?

Через applicationContext.xml в web/WEB-INF. Например:

```
<property name="mappingResources">
 <list>
  <value>com/example/domain/Entity1.hbm.xml</value>
 list>
property>
```

39). Как добавить поддержку Spring в web-приложение

Достаточно просто указать ContextLoaderListener в web.xml файле приложения:

```
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListenerlistener-class>
listener>
```

40). Можно ли использовать xyz.xml вместо applicationContext.xml?

ContextLoaderListener – это ServletContextListener, который инициализируется когда ваше web-приложение стартует. По-умолчанию оно загружает файл WEB-INF/applicationContext.xml. Вы можете изменить значение по-умолчанию, указав параметр **contextConfigLocation**.

Пример:

```
<listener>
   <listener-class>org.springframework.web.context.ContextLoaderListener
      <context-param>
         <param-name>contextConfigLocation</param-name>
         <param-value>/WEB-INF/xyz.xml</param-value>
      context-param>
   listener-class>
listener>
```

41). Как настраивается соединение с БД в Spring?

Используя datasource “org.springframework.jdbc.datasource.DriverManagerDataSource”.

Пример:

```
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
   <property name="driverClassName" value="com.mysql.cj.jdbc.Driver" />
   <property name="url" value="jdbc:mysql://localhost/mydb" />
   <property name="username" value="myuser" />
   <property name="password" value="mypassword" />
</bean>
```

42). Назовите некоторые из шаблонов проектирования, используемых в Spring Framework?

Spring Framework использует множество шаблонов проектирования, например:

1. Singleton Pattern
2. Factory Pattern
3. Prototype Pattern
4. Adapter Pattern
5. Proxy Pattern
6. Template Method Pattern
7. Front Controller
8. Data Access Object
9. Dependency Injection и Aspect Oriented Programming

43). Для чего нужен @ComponentScan?

Если вы понимаете как работает Component Scan, то вы понимаете Spring

Первый шаг для описания Spring Beans это добавление аннотации — @Component, или @Service, или @Repository.

Однако, Spring ничего не знает об этих бинах, если он не знает где искать их. То, что скажет Spring где искать эти бины и называется Component Scan. В **@ComponentScan** вы указываете пакеты, которые должны сканироваться.

Spring будет искать бины не только в пакетах для сканирования, но и в их подпакетах.

Read & Write Code

44). Как вы добавите Component Scan в Spring Boot?

```
@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

@SpringBootApplication определяет автоматическое сканирование пакета, где находится класс Application

Всё будет в порядке, ваш код целиком находится в указанном пакете или его подпакетах.

Однако, если необходимый вам компонент находится в другом пакете, вы должны использовать дополнительно аннотацию **@ComponentScan**, где перечислите все дополнительные пакеты для сканирования.

45). **В чём отличие между** @Component и @ComponentScan?

@Component и @ComponentScan предназначены для разных целей.

@Component помечает класс в качестве кандидата для создания Spring бина.
@ComponentScan указывает где Spring искать классы, помеченные аннотацией @Component или его производной.

46). Для чего используется аннотация @Bean?

В классах конфигурации Spring, @Bean используется для определения компонентов с кастомной логикой.

47). В чём разница между @Bean и @Component?

@Bean используется в конфигурационных классах Spring. Он используется для непосредственного создания бина.

@Component используется со всеми классами, которыми должен управлять Spring. Когда Spring видит класс с @Component, Spring определяет этот класс как кандидата для создания bean.

48). Можем ли мы использовать @Component вместо @Service для бизнес логики?

Да. конечно.

Если **@Component** является универсальным стереотипом для любого Spring компонента, то **@Service** в настоящее время является его псевдонимом. Однако, в официальной документации Spring рекомендуется использовать именно @Service для бизнес логики.
Вполне возможно, что в будущих версиях фреймворка, для данного стереотипа добавится дополнительная семантика, и его бины станут обладать дополнительной логикой.

49). В чем различие между web.xml и servlet.xml?

web.xml — Метаданные и конфигурация любого веб-приложения, совместимого с Java EE. Java EE стандарт для веб-приложений.

servlet.xml — файл конфигурации, специфичный для Spring Framework.

50). Что предпочитаете использовать для конфигурации Spring – xml или аннотирование?

Предпочитаю аннотации, если кодовая база хорошо описывается такими элементами, как @Service, @Component, @Autowired.

Однако когда дело доходит до конфигурации, у меня нет каких-либо предпочтений. Я бы оставил этот вопрос команде с которой я буду работать.

51). Можем ли мы применить @Autowired с не сеттерами и не конструкторами методами?

Да, конечно.

@Autowired может использоваться вместе с конструкторами, сеттерами или любыми другими методами. Когда Spring находит @Autowired на методе, Spring автоматически вызовет этот метод, после создания экземпляра бина. В качестве аргументов, будут подобраны подходящие объекты из контекста Spring.

52). В чем разница между Сквозной Функциональностью (Cross Cutting Concerns) и АОП (аспектно оринтированное программирование)?

Сквозная Функциональность — функциональность, которая может потребоваться вам на нескольких различных уровнях — логирование, управление производительностью, безопасность и т.д.
АОП — один из подходов к реализации данной проблемы.

53). В чем разница между IOC (Inversion of Control) и Application Context?

IOC — инверсия управления. Вместо ручного внедрения зависимостей, фреймворк забирает ответственность за это.
ApplicationContext — реализация IOC спрингом.

Bean Factory — это базовая версия IOC контейнера

Application Context также включает дополнительные функции, которые обычно нужны для разработки корпоративных приложений.

54). В чем разница между classPathXmlApplicationContext и annotationConfigApplicationContext?

classPathXmlApplicationContext — если вы хотите инициализировать контекст Spring при помощи xml
annotationConfigApplicationContext — если вы хотите инициализировать контекст Spring при помощи конфигурационного класса java.

55). Почему возвращаемое значение при применении аспекта @Around может потеряться? Назовите причины.

Метод, помеченный аннотацией @Around, должен возвращать значение, которое он (метод) получил из joinpoint.proceed()

```
@Around("trackTimeAnnotation()")
public Object around(ProceedingJoinPoint joinPoint) throwsThrowable{
    long startTime = System.currentTimeMillis();
    Object retVal = joinPoint.proceed();
    long timeTaken = System.currentTimeMillis() - startTime;
    logger.info("Time taken by {} is equal to {}",joinPoint, timeTaken);
    return retVal;
}
```

56). Как вы решаете какой бин инжектить, если у вас несколько подходящих бинов. Расскажите о @Primary и @Qualifier?

Если есть бин, который вы предпочитаете большую часть времени по сравнению с другими, то используйте **@Primary**, и используйте **@Qualifier** для нестандартных сценариев. Когда Spring обнаруживает неоднозначность при инжекте бина, он выбирает тот бин, который помечен аннотацией @Primary.

Если все бины имеют одинаковый приоритет, мы всегда будем использовать @Qualifier.
@Qualifier - это аннотация, которую вы можете использовать для явного указания имени или значения квалификатора для бина, который должен быть инжектирован.

Если бин надо выбрать во время исполнения программы, то эти аннотации вам не подойдут. Вам надо в конфигурационном классе создать метод, пометить его аннотацией **@Bean**, и вернуть им требуемый бин.

57). В чём разница между @Controller и @RestController?

@RestController = @Controller + @ResponseBody

@RestController превращает помеченный класс в Spring-бин. Этот бин для конвертации входящих/исходящих данных использует **Jackson message converter**. Как правило целевые данные представлены в json или xml.

58). Почему иногда мы используем @ResponseBody, а иногда ResponseEntity?

ResponseEntity необходим, только если мы хотим кастомизировать ответ, добавив к нему **статус ответа**. Во всех остальных случаях будем использовать @ResponseBody.

```
@GetMapping(value=”/resource”)
@ResponseBody
public Resource sayHello() { return resource; }

@PostMapping(value=”/resource”)
public ResponseEntity createResource() {
    ....
return ResponseEntity.created(resource).build();
}
```

Стандартные HTTP коды статусов ответов, которые можно использовать.

200 — SUCCESS
201 — CREATED
404 — RESOURCE NOT FOUND
400 — BAD REQUEST
401 — UNAUTHORIZED
500 — SERVER ERROR

Для @ResponseBody единственные состояния статуса это SUCCESS(200), если всё ок и SERVER ERROR(500), если произошла какая-либо ошибка.

Допустим мы что-то создали и хотим отправить статус CREATED(201). В этом случае мы используем ResponseEntity.

59). В чем разница между Filters, Listeners и Interceptors?

Концептуально всё просто, фильтры сервлетов могут перехватывать только **HTTPServlets**. Listeners могут перехватывать специфические **события**. Как перехватить события которые относятся ни к тем не другим?

Фильтры и перехватчики делают по сути одно и тоже: они перехватывают какое-то событие, и делают что-то до или после.

Java EE использует термин Filter, Spring называет их Interceptors.

Именно здесь AOP используется в **полную силу,** благодаря чему возможно перехватывание вызовов любых объектов.

60). Как работает аннотация @RequestMapping ?

Аннотация @**RequestMapping** используется для сопоставления веб-запросов с методами Spring Controller. Помимо простых вариантов использования, мы можем использовать его для сопоставления заголовков HTTP, привязки частей URI с помощью @**PathVariable** и работы с параметрами URI и аннотацией @**RequestParam** .

61). В чем разница между ModelMap и ModelAndView?

ModelMap и ModelAndView - это два разных способа передачи модели (данных) из контроллера в представление в Spring MVC.

ModelMap:
ModelMap - это простая структура данных, используемая для хранения модели (данных), которые должны быть переданы в представление. ModelMap представляет собой обычный словарь, где ключи - это имена атрибутов, а значения - это сами данные. Вы можете добавлять и удалять атрибуты в ModelMap и передавать её в представление.
Пример использования ModelMap:

```
@GetMapping("/example")
public String example(ModelMap model) {
    model.addAttribute("message", "Hello, World!");
    return "exampleView";
}
```

Здесь мы добавляем атрибут "message" в ModelMap, который будет доступен в представлении "exampleView".

61). Продолжение

ModelAndView:
ModelAndView - это объект, который сочетает в себе модель (данные) и информацию о представлении (название представления). В отличие от ModelMap, ModelAndView позволяет явно указать, какое представление должно быть использовано для отображения данных.
Пример использования ModelAndView:

```
@GetMapping("/example")
public ModelAndView example() {
    ModelAndView modelAndView = new ModelAndView("exampleView");
    modelAndView.addObject("message", "Hello, World!");
    return modelAndView;
}
```

Здесь мы создаем ModelAndView, указываем название представления "exampleView" и добавляем атрибут "message" к модели. В результате, Spring MVC знает, какое представление использовать для отображения данных.

Разница между ними заключается в том, что ModelMap предоставляет только данные (**модель**), но не указывает, как их отображать, в то время как ModelAndView позволяет объединить модель и представление в один объект, что может быть удобно в некоторых случаях. Выбор между ними зависит от ваших предпочтений и требований для конкретного контроллера и представления.

62). В чем разница между model.put() и model.addAttribute()?

model.**put**() - это метод, который предоставляется Java Map, и его можно использовать для добавления атрибутов в модель.

model.**addAttribute**() - это специфичный метод, предоставляемый Spring MVC для добавления атрибутов в модель.

Метод addAttribute отделяет нас от работы с базовой структурой hashmap. По сути addAttribute это обертка над put, где делается дополнительная проверка на null.
Метод addAttribute в отличии от put возвращает modelmap.

model.addAttribute(“attribute1”,”value1”).addAttribute(“attribute2”,”value2”);

63). Что можете рассказать про Form Binding?

Нам это может понадобиться, если мы, например, захотим взять некоторое значение с HTML страницы и сохранить его в БД. Для этого нам надо это значение переместить в контроллер Спринга.

Если мы будем использовать Spring MVC form tags, Spring автоматически свяжет переменные на HTML странице с Бином Спринга.

64). Почему мы используем Hibernate Validator?

Hibernate Validator никак не связан с БД. Это просто библиотека для валидации.

Hibernate Validator версии 5.x является эталонной реализацией Bean Validation 1.1

Hibernate Validator является сертифицированным.

65). Где должны располагаться статические (css, js, html) ресурсы в Spring MVC приложении?

Расположение статических ресурсов можно настроить. В документации Spring Boot рекомендуется использовать /static, или /public, или /resources, или /META-INF/resources.

66). Почему для конфиденциальных данных рекомендуется использовать POST, а не GET запросы?

В случае **GET** запроса передаваемые параметры являются частью **url**, и все маршрутизаторы, через которые пройдет наш GET запрос, смогут их прочитать.

В случае **POST** запроса передаваемые параметры являются частью **тела** запроса. При использовании HTTPs, тело запроса **шифруется**. Следовательно, использование POST запросов является более безопасным.

67). Можно ли передать в запросе один и тот же параметр несколько раз?

Пример:
http://localhost:8080/login?name=Ranga&name=John&name=Smith

Да, можно принять все значения, используя массив в методе контроллера.

```
public String method(@RequestParam(value="name") String[] names){
}
```

68). В чем разница между Spring Boot и Spring MVC? Или между Spring Boot и Spring Framework? Можете ли вы использовать их вместе в одном проекте?

Spring Boot построен поверх Spring Framework.
Пример: Spring Framework предлагает вам возможность читать файлы свойств .properties из различных мест, например, с помощью аннотаций @PropertySource. Он также предлагает вам возможность писать JSON REST контроллеры с помощью инфраструктуры Web MVC.
Проблема в том, что вы должны указать Spring откуда читать свойства приложения и правильно настроить веб-фреймворк, например, для поддержки JSON. Spring Boot, с другой стороны, берет эти отдельные части и предварительно настраивает их для вас.
Например:

- Он всегда автоматически ищет файлы application.properties в различных заранее определенных местах и считывает их.
- Он всегда загружает **встроенный** Tomcat, поэтому вы можете сразу увидеть результаты написания ваших @RestControllers и начать писать веб-приложения.
- Он автоматически настраивает все для отправки / получения JSON, не беспокоясь о конкретных зависимостях Maven / Gradle. Всё, путем запуска основного метода в классе Java, который аннотируется аннотацией @SpringBootApplication.

69). Почему вам не нужно указывать версии зависимостей в файле pom.xml при включении сторонних библиотек? Верно ли это для всех сторонних библиотек или только для некоторых?

Это потому, что Spring Boot выполняет за вас некоторое управление зависимостями.
На верхнем уровне стартеры Spring Boot закачивают родительский файл pom.xml (или файл build.gradle), в котором определены все зависимости и соответствующие версии, которые поддерживает конкретная версия Spring Boot - так называемый **BOM** (**Bill Of Materials**). Затем вы можете просто использовать эти предопределенные версии или переопределить номера версий в своих собственных сценариях сборки.

70). Расскажите о Spring Security.

Проект Spring Security предоставляет широкие возможности для защиты приложения. Кроме стандартных настроек для аутентификации, авторизации и распределения ролей и маппинга доступных страниц, ссылок и т.п., предоставляет защиту от различных вариантов атак (например CSRF). Имеет множество различных настроек, но остается легким в использовании.

Чтобы использовать Spring Security в веб-приложениях, мы можем начать с простой аннотации **@EnableWebSecurity** .

71). Как работает прототип Scope?

Prototype Scope означает, что каждый раз, когда мы вызываем экземпляр Bean, Spring будет создавать новый экземпляр и возвращать его. Это отличается от **singleton** по умолчанию, где один экземпляр объекта создается один раз для контейнера Spring **IoC**.

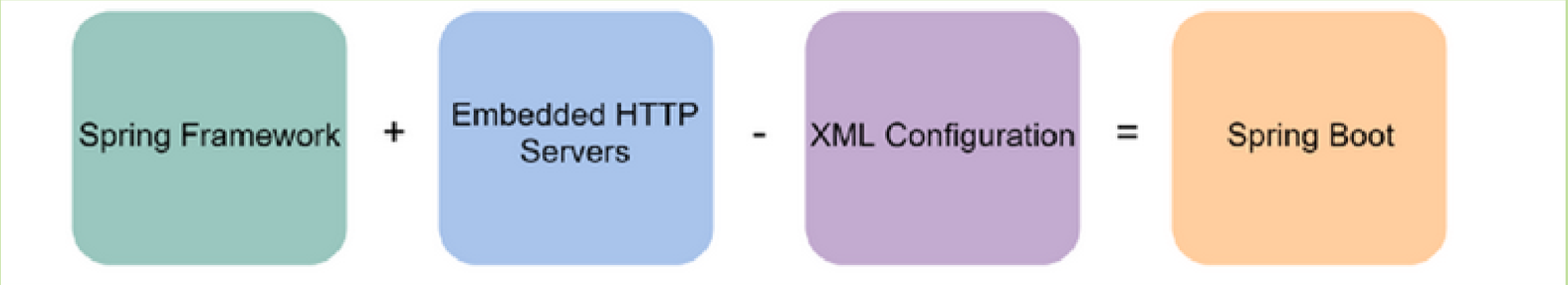
72). Что такое Spring Boot?

Spring Boot – это Java-фреймворк с открытым исходным кодом, используемый для создания микросервисов. Это проект, построенный на основе Spring, чтобы упростить задачу развертывания приложений Java. Его двумя основными компонентами являются Spring Framework и встроенные HTTP-серверы.

Spring Boot используется для:

- Упрощение процесса разработки готовых к производству пружинных приложений
- Избегания конфигурации XML Spring
- Сокращения времени разработки за счет уменьшения количества необходимых инструкций по импорту
- Обеспечения взвешенного подхода к развитию

Часто используются для быстрого запуска приложений Spring.



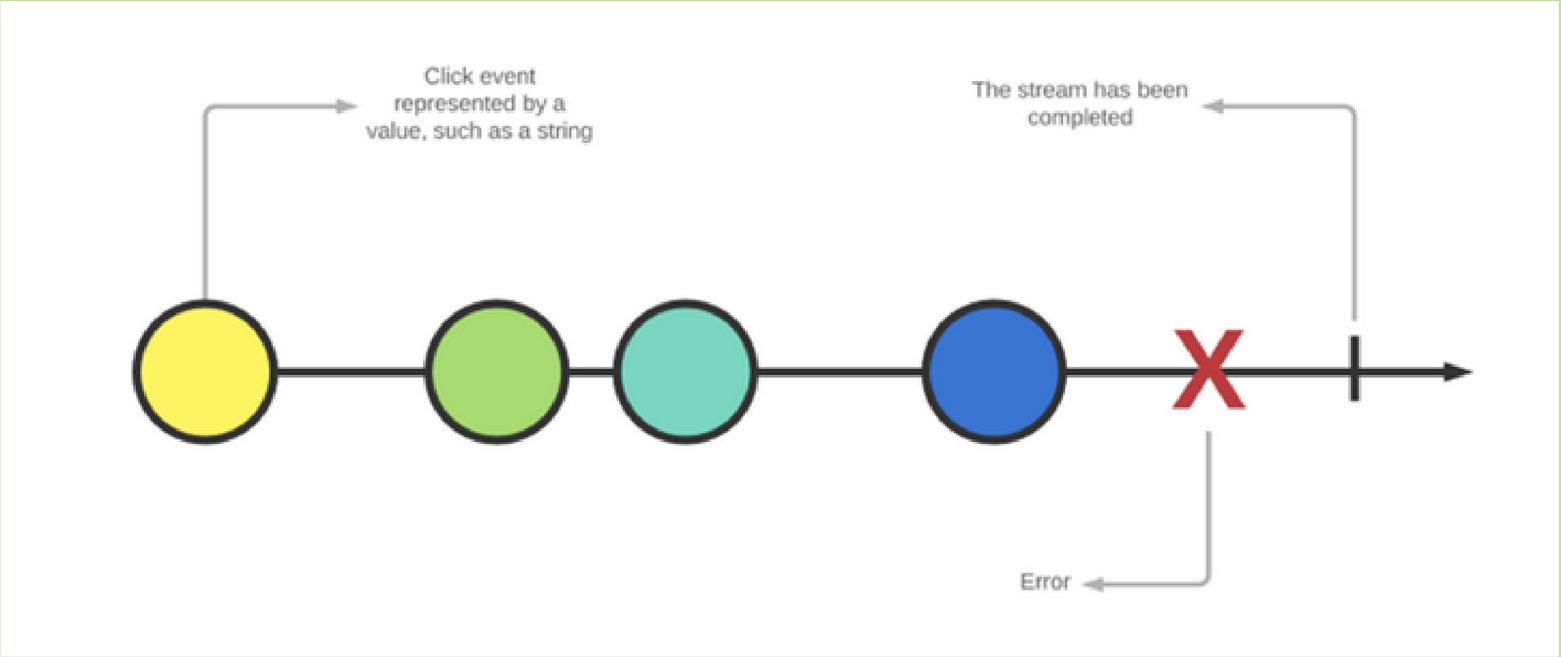
73). Как проще всего развернуть приложение Spring Boot в рабочей среде? Какие еще есть варианты?

Самый простой способ развернуть приложение Spring Boot — это поместить **.jar** файл со встроенным контейнером сервлетов на любой сервер или платформу, на которой установлена **JRE**.
По организационным и историческим причинам вы также можете развернуть приложение Spring Boot как файл **.war** в существующем контейнере сервлетов или сервере приложений.

И последнее, но не менее важное: вы, конечно, также можете поместить свой **.jar** файл в образ **Docker** и даже развернуть его с помощью **Kubernetes**.

74). Что такое Реактивное программирование?

Реактивное программирование – это парадигма программирования, которая основывается на запрограммированных действиях, запускаемых в связи с событиями, а не на хронологическом порядке кода. Реактивные программы эффективно используют компьютерные ресурсы и хорошо масштабируются всего несколькими потоками. Его непоследовательная форма позволяет избежать блокировки стека и поддерживать оперативность реагирования.



75). Что такое Spring webflux?

Webflux – это реактивный веб-фреймворк, который служит альтернативой MVC. Webflux обеспечивает лучшую масштабируемость и предотвращает блокировку стека.