1. **Перечислите Spring модули и их назначение.**

Spring представляет собой набор «модулей», независимых друг от друга (программист может выбрать набор модулей необходимых ему и не использовать остальные). Основные модули, пришедшие еще из первой версии Spring, делятся на две части. Первая часть — это модули, которые поддерживают особые техники или стили программирования: Inversion of Control и аспектно­-ориентированное программирование. Вторая часть — это модули, содержащие классы­-обертки для распространенных инструментов решения типичных задач (взаимодействие с БД, создание веб-­интерфейса). В основном они полагаются на более низкоуровневые в некотором смысле, интрузивные API. При использовании оберток, предоставляемых Spring, достигается большая свобода от конкретных API, повышается логическая прозрачность кода. Перечислим модули, дав общую характеристику каждому.

**IoC-контейнер** — это особый модуль, являющийся ядром Spring (часто называется Core Container). Он составляет исключение сказанному выше о независимости: от него зависит большая часть остальных модулей. Техника Inversion of Control подразумевает следующее: если некоторый объект A в приложении зависит от другого объекта B (скажем, имеет поле типа B), то получением объекта B для A занимается IoC­ контейнер, а не сам объект A; контейнер, исходя из своих настроек, решает, нужно ли создать новый 3 объект B, или получить его из пула, который также может поддерживаться контейнером, или еще что-­то. Эта простая идея (в духе «разделяй и властвуй») дает немало преимуществ, как­-то: снижается зацепление разных частей программного обеспечения друг между другом; пропадает необходимость многократного повторения кода поиска зависимостей в прикладном коде, последний становится более прозрачным для понимания; облегчает модульное тестирование.

**АОП-модуль**. Модуль для поддержки аспектно-­ориентированного программирования. АОП стало ответом на несостоятельность объектного подхода в отношении так называемой сквозной функциональности (cross­cutting concerns), которая не поддавалась объектной декомпозиции и способствовала превращению кода, написанного на чистом ОО­языке, в «слоенный пирог». Типичными примерами сквозной функциональности являются логирование, управление транзакциями, управление авторизацией: обращение к этим средствам пронизывает прикладной код, хотя не имеет прямого отношения к решаемой им задаче. Spring предоставляет две альтернативы: свою реализацию ключевых сущностей АОП, как они определены AOP Alliance, объединением, разрабатывающим подходы к реализации АО­ парадигмы в Java, или возможность относительно бесшовной интеграции вашего прикладного кода на Java со средствами наиболее популярного АОП­ языка AspectJ.

**Модуль доступа к данным**. Java располагает стандартным программным пакетом для взаимодействия с реляционными базами данных JDBC.

**Веб-модуль и Spring MVC**. Рассчитано, как ясно из названия, на использование при построении веб-­приложений. Model­View­Controller весьма популярная модель архитектуры веб­-приложения (создавалась, правда, задолго до появления службы www), призванная, разумеется, четко разделять обязанности в нем. Ее не стоит путать с моделью бизнес-­приложения, на которую мы решили ориентироваться. Если попытаться отобразить одно на другое, то получится так: View и Controller располагаются в презентационном слое (первое связано с конкретной технологией отображения информации, а второй инкапсулирует поведение вашего приложения с точки зрения пользователя: реакцию на нажатие кнопок, перехода по ссылкам), на долю Model приходится слой бизнес-­логики. Для Java существует огромное количество веб­-фреймверков, так или иначе реализующих MVC, 5 Spring MVC стал одним из них, а веб-­модуль Spring поддерживает интеграцию вашего приложения с одним из таких фреймверков, подтягивает возможности IoC ­контейнера к более специфическим особенностям web и, кроме того, предоставляет готовые решения для некоторых мелких, часто встречающихся здесь задач, таких как, к примеру, загрузка файлов по частям. Перечислены основные модули Spring, однако далеко не все. Я упомяну более мелкие или мало используемые (отчасти по причине того, что появились они недавно) средства совсем бегло. Но для начала требуется сказать следующее. Одну из главных ролей в мире создания бизнес­приложений на Java играет, как нетрудно догадаться, компания Sun, которая выпускает многочисленные спецификации, детально описывающие способы решения широкого круга задач. Эти спецификации объединены под общим названием Java Enterprise Edition (Java EE). Spring изначально создавался в противовес наиболее тяжеловесным частям Java EE. Однако с существенной частью Java EE Spring успешно взаимодействует (не следует путать взаимодействие с системами, реализующими спецификации Sun, и саму реализацию: Spring предоставляет первое); часть оставшихся модулей поддерживают взаимодействие с такими спецификациями: Java Management Extensions (JMX, стандарт на средства мониторинга и конфигурирования приложений), Java EE Connector API (JCA, стандарт на взаимодействие с legacy­системами, в том числе не­Java системами, и вообще на интеграцию создававшихся независимо информационных систем). И остается группа модулей, направленных на поддержку создания распределенных систем. Модуль удаленного взаимодействия (Remoting): сюда входит абстрагирование от конкретных технологий, упомянем две наибо6 лее типичные: RMI и JAX­RPC (соответственно, бинарный и текстовый XML протоколы удаленных взаимодействий).

**Модуль для асинхронного взаимодействия** (взаимодействия на основе передачи сообщений), предназначенный для связи с системами, поддерживающими еще одну спецификацию Java EE — Java Message Service (JMS). И, наконец, отдельный фреймверк, не входящий в состав Spring Framework, но доступный для скачивания на сайте Spring: Spring Web Services, который как следует из названия, предоставляет ряд средств, облегчающий создание веб­служб (распределенных систем, базирующихся на нескольких спецификациях W3C, описывающих формат сообщений, язык описания интерфейсов системы и другое, для которых характерно плотное использование XML). В заключение стоит упомянуть еще один фреймверк с которым у Spring давно сложились хорошие отношения, и сейчас он почти часть Spring, это Acegi Security System: система для предоставляющая решения авторизации и аутентификации в приложениях (в основном, веб).

1. **Расскажите о составе Spring Framework.**

**Spring Projects на веб-уровне**

Следующие проекты Spring так или иначе помогут вам в разработке интерфейса.

**Spring Session** позволяет обрабатывать веб-сессии в вашем приложении.

**Spring Web Services** помогает вам создать сначала контрактный веб-сервис на основе SOAP.

**Spring Social** позволяет Facebook, LinkedIn, Twitter и другим пользователям заходить на сайт.

**Spring для Android** помогает создавать нативные приложения для Android.

**Spring HATEOAS** помогает создавать представления REST, которые следуют принципу HATEOAS.

**Spring Mobile** помогает создавать страницы, удобные для мобильных устройств., если ваши запросы поступают с мобильного устройства.

**Spring Web Flow** разбивает данные на множество экранов для удобной навигации.

**Spring Security** в центре всех этих проектов по причине все проекты веб-слоя могут использовать Spring Security.

**Common layer**

**Spring Framework** (или Spring Core) Ядро платформы, предоставляет базовые средства для создания приложений — управление компонентами (бинами, **beans**), внедрение зависимостей, MVC фреймворк, транзакции, базовый доступ к БД. В основном это низкоуровневые компоненты и абстракции. По сути, неявно используется всеми другими компонентами.

**Service layer Projects**

**Spring Integration** - подходит для обработки событий. Вы можете подключить его поверх любых проектов Spring Data layer. Это как обработка данных из разных источников. Если надо раз в час брать файл с FTP, разбивать его на строки, которые потом фильтровать, а дальше отправлять в какую-то очередь — это к Spring Integration.

**Spring Cloud**. - это инструмент для создания распределенных приложений

Много полезного для микросервисной архитектуры — service discovery, трасировка и диагностика, балансировщики запросов, circuit breaker-ы, роутеры и т.п

**Data layer Projects**

Проекты в этом слое имеют дело с обработкой данных.

**Spring AMQP** обобщает работу с отправкой и получением сообщений. Он предоставляет низкоуровневый шаблон для взаимодействия с данными. Это обеспечивает фабрику соединений. Он обеспечивает готовую реализацию в RabbitMQ.

**Spring Data** - общий API для выполнения операций CRUD для разных баз данных. Он поддерживает все популярные базы данных SQL, а также базы данных NoSQL.

**Spring Batch** - это фреймворк с множеством функций. Используется для выполнения любой массовой обработки.

**Spring IO Platform** - позволяет легко вводить то, что вам нужно для вашего приложения

1. **Что такое Spring Boot? В чем его преимущества и для чего он используется?**

**Spring Boo**t является проектом на уровне IO **Execution** (уровень выполнения) **IO Spring Framework**

**Spring Boot** - дает вам инструмент командной строки для создания приложения. Позволяет избежать XML конфигурации. Boot позволяет быстро создать и сконфигурить (т.е. настроить зависимости между компонентами) приложение, упаковать его в исполняемый самодостаточный артефакт. Это то связующее звено, которое объединяет вместе набор компонентов в готовое приложение.

Spring Boot:

* Он не использует кодогенерацию. Из кода, который генерится, присутствует только метод main.
* Не использует XML для конфигурации. Все конфигурируется через аннотации
* Используются автоконфигурации по максимуму. Используется *convention over configuration*. Для большинства конфигураций не нужно ничего настраивать.
* Его легко отодвинуть в сторону и "перекрыть" конфигурацию по умолчанию.

Преимущества:

1. Легко используется для развития приложения на основе **Spring**с **Java**или **Groovy Spring**
2. Минимизирует время развития и поднимает производительность
3. Избегает написание многих кодов прототипа (boilerplate), Annotations и конфигурации XML
4. Легко позволяет вам взаимодействовать с приложениями Spring Boot c экологическими системами **Spring**как **Spring JDBC**, **Spring ORM**, **Spring Data**, **Spring Security** и т.д
5. Обеспечивает встроенный Server (Embedded HTTP servers) как **Tomcat**, **Jetty**.... чтобы быстро и легко развивать  и тестировать веб-приложения
6. Предоставляет инструменты **CLI**(Command Line Interface) для развития и тестирования приложений Spring Boot (**Java**или **Groovy**) из командных строк (command prompt) очень легко и быстро
7. Обеспечивает много плагинов для быстрого развития и тестирования приложения **Spring Boot** используя инструменты **Build**, как **Maven**и **Gradle**
8. Предлагает много плагинов для легкой работы с контейнерами встроенными базами данных (embedded database) и базами данных хранящиеся в памяти (in- memory Databases).

**4. Для чего используется аннотация @SpringBootApplication?**

Здесь @SpringBootApplication - составная аннотация, которая объединяет три другие аннотации Spring:

@ SpringBootConfiguration - обозначает класс как класс конфигурации.

@ EnableAutoConfiguration - включает автоматическую настройку Spring Boot. Эта аннотация говорит Spring Boot автоматически настраивать любые компоненты, которые, по ее мнению, вам понадобятся.

@ ComponentScan - включает сканирование компонентов. Это позволяет объявлять другие классы с аннотациями, такими как @Component, @Controller, @Service и другие, чтобы Spring автоматически обнаруживал их и регистрировал как компоненты в контексте приложения Spring.

1. **Объясните принцип IoC (Inversion of Control)? Какие формы используются в Spring для внедрения?**

Принцип **Inversion of Control (IoC)** гласит: *программировать надо на интерфейсах, а не на их реализациях.*

Ядро Spring Framework основано на принципе *инверсии управления* (lnversion of Control - IoC), когда создание и управление зависимостями между компонентами становятся внешними.

Компонент, который требует определенных зависимостей, часто называют зависимым объектом или, в случае IoC, целевым объектом. Вообще говоря, инверсия управления может быть разделена на два подтипа:

**1) внедрение зависимостей (Dependency lnjection) и**

**2) поиск зависимостей (Dependency Lookup).**

**6. В чем суть понятия lnversion of Control (IoC)?**

Принцип **Inversion of Control (IoC)** гласит: *программировать надо на интерфейсах, а не на их реализациях.*

Ядро Spring Framework основано на принципе *инверсии управления* (lnversion of Control - IoC), когда создание и управление зависимостями между компонентами становятся внешними.

Компонент, который требует определенных зависимостей, часто называют зависимым объектом или, в случае IoC, целевым объектом. Вообще говоря, инверсия управления может быть разделена на два подтипа:

**1) внедрение зависимостей (Dependency lnjection) и**

**2) поиск зависимостей (Dependency Lookup).**

**7. В чем различие внедрение зависимостей (Dependency lnjection) и поиска зависимостей (Dependency Lookup)?**

Первый способ— применить внедрение зависимостей. Внедрение зависимостей реализуется несколькими способами, среди которых можно выделить:

* Внедрение через конструктор
* Внедрение через set-метод
* Внедрение через интерфейс

Внедрение зависимостей через конструктор позволяет внедрять зависимости для final-свойств, а так же внедрять сразу несколько зависимостей.

**Поиск зависимостей**

В случае с поиском зависимостей класс должен самостоятельно реализовывать логику получения зависимостей извне. Для этого он должен иметь доступ к некоему источнику зависимостей.

Оба описанных способа позволяют снизить связанность между компонентами, но поиск зависимостей предполагает, что компонент, для которого он применяется, должен реализовывать логику получения необходимых зависимостей. Это далеко не всегда удобно, соответственно, рекомендуется применять первый способ — внедрение зависимостей, там, где это возможно.

Но в некоторых случаях использование поиска зависимостей всё же может быть более выгодно, например, при реализации шаблона проектирования «команда».

**8. Что такое JavaBean? Какие есть правила описания и использования?**

**JavaBeans** — классы в языке Java, написанные по определённым правилам. Чтобы класс мог работать как bean, он должен соответствовать определённым соглашениям об именах методов, конструкторе и поведении.

Правила описания гласят:

* Класс должен иметь конструктор без параметров, с модификатором доступа public. Такой конструктор позволяет инструментам создать объект без дополнительных сложностей с параметрами.
* Свойства класса должны быть доступны через get, set и другие методы (так называемые методы доступа), которые должны подчиняться стандартному соглашению об именах. Это легко позволяет инструментам автоматически определять и обновлять содержание bean’ов. Многие инструменты даже имеют специализированные редакторы для различных типов свойств.
* Класс должен быть сериализуем. Это даёт возможность надёжно сохранять, хранить и восстанавливать состояние bean независимым от платформы и виртуальной машины способом.
* Класс должен иметь переопределенные методы equals(), hashCode() и toString().

**9. Перечислите области видимости bean.**

Изначально, Spring Framework поддерживает несколько вариантов, некоторые доступны, только если вы используете web-aware ApplicationContext. Также вы можете создать свою собственную область видимости. Ниже приведен список областей видимостей:

**singleton** - По умолчанию. Spring IoC контейнер создает единственный экземпляр бина. Как правило, используется для бинов без сохранения состояния(stateless)

**prototype** - Spring IoC контейнер создает любое количество экземпляров бина. Новый экземпляр бина создается каждый раз, когда бин необходим в качестве зависимости, либо через вызов getBean(). Как правило, используется для бинов с сохранением состояния(stateful)

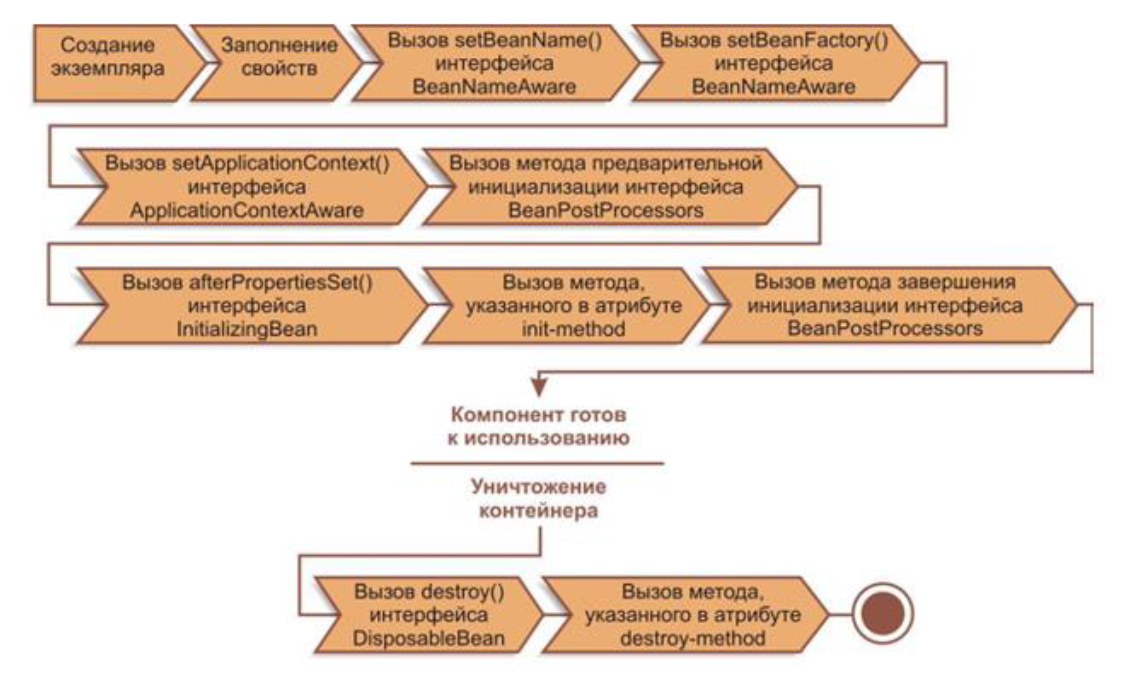
**request** - Жизненный цикл экземпляра ограничен единственным HTTP запросом; для каждого нового HTTP запроса создается новый экземпляр бина. Действует, только если вы используете web-aware ApplicationContext

**session** - Жизненный цикл экземпляра ограничен в пределах одной и той же HTTP Session. Действует, только если вы используете web-aware ApplicationContext

**global session** - Жизненный цикл экземпляра ограничен в пределах глобальной HTTP Session(обычно при использовании portlet контекста). Действует, только если вы используете web-aware ApplicationContext

**application** - Жизненный цикл экземпляра ограничен в пределах ServletContext. Действует, только если вы используете web-aware ApplicationContext.

1. **Опишите ЖЦ бина.**



**11. Поясните значения аннотаций: @Configuration, @Bean, @Component, @Service, @Repository, @Controller.**

Аннотация **@Controller** как раз и сообщает *Spring MVC*, что данный класс является контроллером

**@Configuration** сообщает Spring что данный класс является конфигурационным, содержит определения и зависимости bean-компонентов.

Бины (bean) — это объекты, которые управляются Spring'ом. Для определения бина используется аннотация **@Bean**.

**@Repository -** Это указывает на то, что класс определяет хранилище данных.

**@Service** содержат бизнес-логику и вызывают методы на уровне хранилища.

**@Component** — универсальный компонент

**12. Spring Expression Language (SpEL): расскажите об особенности и области использования.**

**SpEL** — это язык выражений, созданный для Spring Framework, который поддерживает запросы и управление графом объектов во время выполнения.  
Также важно отметить, что SpEL создан в виде API-интерфейса, позволяющего интегрировать его в другие приложения и фрэймворки.

## Возможности SpEL

SpEL поддерживает следущую функциональность:

* Символьные выражения
* Булевые и реляционные операторы
* Регулярные выражения
* Class выражения
* Доступ к свойствам, массивам, спискам, картам
* Вызов методов
* Вызов конструкторов
* Встроенные списки и карты
* Тернарные операторы
* Переменные
* Пользовательские функции
* Шаблонные выражения
* и др.

13**. Охарактеризуйте основные Core Container Spring.**

Ключевым элементом Spring Framework является Spring Container. Container создаёт объекты, связывает их вместе, настраивает и управляет ими от создания до момента уничтожения.

Для управления компонентами, из которых состоит приложение, Spring Container использует Внедрение Зависимостей (DI). Эти объекты называются Spring Beans, которые мы обсудим далее.

Spring Container получает инструкции какие объекты инстанциировать и как их конфигурировать через метаданные.

Метаданные могут быть получены 3 способами:  
–XML  
–Аннотации Java  
– Java код

**14. Как в Spring происходит разрешение зависимостей?**

IoC контейнер выполняет разрешение зависимостей бинов в следующем порядке:

1) Создается и инициализируется ApplicationContext с метаданными конфигурации, которые описывают все бины. Эти метаданные могут быть описаны через XML, Java-код или аннотации

2) Для каждого бина и его зависимостей вычисляются свойства, аргументы конструктора или аргументы статического фабричного метода, либо обычного (без аргументов) конструктора. Эти зависимости предоставляются бину, когда он(бин) уже создан. Сами зависимости инициализируются рекурсивно, в зависимости от вложенности в себе других бинов. Например, при инициализации бина А, который имеет зависимость В, а В зависит от С, сначала инициализируется бин С, потом В, а уже потом А

3) Каждому свойству или аргументу конструктора устанавливается значение или ссылка на другой бин в контейнере

4) Для каждого свойства или аргумента конструктора подставляемое значение конвертируется в тот формат, который указан для свойства или аргумента. По умолчанию Spring может конвертировать значения из строкового формата во все встроенные типы, такие как int, long, String, boolean и др.

**15. Поясните как работает DispatcherServlet, HandlerMapping, ViewResolver? Как происходит обработка запроса?**

Когда мы пишем в строке браузера запрос, его принимает ***Dispatcher Servlet***, далее он находит для обработки этого запроса подходящий контроллер с помощью ***HandlerMapping*** (это такой интерфейс для выбора контроллера, проверяет в каком из имеющихся контроллеров есть метод, принимающий такой адрес), вызывается подходящий метод и ***Controller*** возвращает информацию о представлении, затем диспетчер находит нужное представления по имени при помощи ***ViewResolver***'а, после чего на это представление передаются данные модели и на выход мы получаем нашу страничку.

**16. Как используется паттерн «Front Controller» в Spring?**

Spring предоставляет [DispatcherServlet](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/servlet/DispatcherServlet.html), чтобы гарантировать, что входящий запрос будет отправлен вашим контроллерам.

Паттерн Front Controller используется для обеспечения централизованного механизма обработки запросов, так что все запросы обрабатываются одним обработчиком. Этот обработчик может выполнить аутентификацию, авторизацию, регистрацию или отслеживание запроса, а затем передать запрос соответствующему контроллеру.

**17. Как происходит адресация в контроллере?**

В RequestMapping передаем тип метода и ссылку.

**18. Расскажите про Spring MVC архитектуру.**

Фреймворк **Spring MVC** обеспечивает архитектуру паттерна Model — View — Controller (Модель — Отображение (далее — Вид) — Контроллер) при помощи слабо связанных готовых компонентов. Паттерн MVC разделяет аспекты приложения (логику ввода, бизнес-логику и логику UI), обеспечивая при этом свободную связь между ними.

* **Model** (Модель) инкапсулирует (объединяет) данные приложения, в целом они будут состоять из POJO («Старых добрых Java-объектов», или бинов).
* **View** (Отображение, Вид) отвечает за отображение данных Модели, — как правило, генерируя HTML, которые мы видим в своём браузере.
* **Controller** (Контроллер) обрабатывает запрос пользователя, создаёт соответствующую Модель и передаёт её для отображения в Вид.

**19. За что отвечает WebApplicationContext?**

В определении WebApplicationContext сказано, что это интерфейс, который предоставляет конфигурацию для веб приложения. Он является read-only (т.е. его нельзя перезаписать) во время выполнения приложения, но может быть перезагружен, если это позволяет реализация.

WebApplicationContext содержит в себе Controllers, [HandlerMapping](https://javastudy.ru/spring-mvc/spring-mvc-handler-mapping/), ViewResolver. Всё это интерфейсы, которые отвечают за контроллеры, поиск представлений и обработку запросов.