INTERVIEW JAVA CORE JAVA DEVELOPER

Ответы на вопросы на собеседование Java core (часть 4).

Vasyl K 8:02:00 2 Комментарии

- **ЧТО ТАКОЕ РЕФЛЕКСИЯ?**

Рефлексия используется для получения или модификации информации о типах во время выполнения программы. Этот механизм позволяет получить сведения о классах, интерфейсах, полях, методах, конструкторах во время исполнения программы. При этом не нужно знать имена классов, методов или интерфейсов. Также этот механизм позволяет создавать новые объекты, выполнять методы и получать и устанавливать значения полей.

- **ЧТО ПРОИЗОЙДЕТ СО СБОРЩИКОМ МУСОРА (GC), ЕСЛИ ВО ВРЕМЯ ВЫПОЛНЕНИЯ МЕТОДА FINALIZE() НЕКОТОРОГО ОБЪЕКТА ПРОИЗОЙДЕТ ИСКЛЮЧЕНИЕ?**

Во время старта JVM запускается поток finalizer, который работает в фоне. Этот поток имеет метод runFinalizer, который игнорирует все исключения методов finalize объектов перед сборкой мусора.
То есть если во время выполнения метода finalize возникнет исключительная ситуация, его выполнение будет остановлено и это никак не скажется на работоспособности самого сборщика мусора (garbage collector).

- **ЧТО ТАКОЕ ИНТЕРНАЦИОНАЛИЗАЦИЯ, ЛОКАЛИЗАЦИЯ?**

Интернационализация (internationalization, а для краткости – i18n) – такой способ создания приложений, при котором их можно легко адаптировать для разных аудиторий, говорящих на разных языках.
Локализацияlocalization а для краткости – l10n) – адаптация интерфейса приложения под несколько языков. Добавление нового языка может внести определенные сложности в локализацию интерфейса.

- **ЧТО ТАКОЕ АННОТАЦИИ В JAVA?**

Аннотации – это своего рода метатеги, которые добавляются к коду и применяются к объявлению пакетов, классов, конструкторов, методов, полей, параметров и локальных переменных. Аннотации всегда обладают некоторой информацией и связывают эти "дополнительные данные" и все перечисленные конструкции языка.
Фактически аннотации представляют собой их дополнительные модификаторы, применение которых не влечет за собой изменений ранее созданного кода.

- **КАКИЕ ФУНКЦИИ ВЫПОЛНЯЕТ АННОТАЦИИ?**

Аннотация выполняет следующие функции:

1. дает необходимую информацию для компилятора;
2. дает информацию различным инструментам для генерации другого кода, конфигураций и т. д.;
3. может использоваться во время работы кода;

Самая часто встречаемая аннотация, которую встречал любой программист, даже начинающий это @Override.

- **КАКИЕ ВСТРОЕННЫЕ АННОТАЦИИ В JAVA ВЫ ЗНАЕТЕ?**

В языке Java SE определено несколько встроенных аннотаций, большинство из их являются специализированными. Четыре типа @Retention, @Documented, @Target и @Inherited – из пакета java.lang.annotation.
Из оставшиеся выделяются – @Override, @Deprecated, @SafeVarargs и @SuppressWarnings – из пакета java.lang. Широкое использование аннотаций в различных технологиях и фреймворках обуславливается возможностью сокращения кода и снижения его связанности.

- **ЧТО ДЕЛАЮТ АННОТАЦИИ @RETENTION, @DOCUMENTED, @TARGET И @INHERITED?**

Эти аннотации, имеют следующее значение:

- @Retention – эта аннотация предназначена для применения только в качестве аннотации к другим аннотациям, позволяет указать жизненный цикл аннотации: будет она присутствовать только в исходном коде, в скомпилированном файле, или она будет также видна и в процессе выполнения. Выбор нужного типа зависит от того, как вы хотите использовать аннотацию.
- @Documented – это маркер-интерфейс, который сообщает инструменту, что аннотация должна быть документирована.
- @Target – эта аннотация задает тип объявления, к которым может быть применима аннотация. Принимает один аргумент, который должен быть константой из перечисления ElementType, это может быть поле, метод, тип и т.д. Например, чтобы

указать, что аннотация применима только к полям и локальным переменным: @Targer({ ElementType.FIELD, ElementTyle.LOCAL_VARIABLE })
- @Inherited – это аннотация-маркер, которая может применяться в другом объявление аннотации, она касается только тех аннотаций, что будут использованы в объявлениях классов. Эта аннотация позволяет аннотации супер класса быть унаследованной в подклассе.

- **ЧТО ДЕЛАЮТ АННОТАЦИИ @OVERRIDE, @DEPRECATED, @SAFEVARARGS И @SUPPRESSWARNINGS?**

Эти аннотации предназначены для:
- @Override – аннотация-маркер, которая может применяться только к методам. Метод, аннотированный как @Override, должен переопределять метод супер класса.
- @Deprecated – указывает, что объявление устарело и должно быть заменено более новой формой.
- @SafeVarargs – аннотация-маркер, применяется к методам и конструкторам. Она указывает, что никакие небезопасные действия, связанные с параметром переменного количества аргументов, недопустимы. Применяется только к методам и конструкторам с переменным количеством аргументов, которые объявлены как static или final.
- @SuppressWarnings – эта аннотация указывает, что одно или более предупреждений, которые могут быть выданы компилятором следует подавить.

- **КАКОЙ ЖИЗНЕННЫЙ ЦИКЛ АННОТАЦИИ МОЖНО УКАЗАТЬ С ПОМОЩЬЮ @RETENTION?**

Существует 3 возможные варианты чтобы указать где аннотация будет жить. Они инкапсулированы в перечисление java.lang.annotation.RetentionPolicy. Это SOURSE, CLASS, RUNTIME.
- SOURCE – содержаться только в исходном файле и отбрасываются при компиляции.
- CLASS – сохраняются в файле, однако они недоступны JVM во время выполнения.
- RUNTIME – сохраняются в файле во время компиляции и остаются доступными JVM во время выполнения.

- **К КАКИМ ЭЛЕМЕНТАМ МОЖНО ПРИМЕНЯТЬ АННОТАЦИЮ, КАК ЭТО УКАЗАТЬ?**

Для того чтобы ограничить использование аннотации её нужно проаннотировать. Для этого существует аннотация @Target.
- @Target(ElementType.PACKAGE) – только для пакетов;
- @Target(ElementType.TYPE) – только для классов;
- @Target(ElementType.CONSTRUCTOR) – только для конструкторов;
- @Target(ElementType.METHOD) – только для методов;
- @Target(ElementType.FIELD) – только для атрибутов(переменных) класса;
- @Target(ElementType.PARAMATER) – только для параметров метода;
- @Target(ElementType.LOCAL_VARIABLE) – только для локальных переменных.

В случае если вы хотите, что бы ваша аннотация использовалась больше чем для одного типа параметров, то можно указать @Target следующим образом:

```
@Target({ ElementType.PARAMETER, ElementType.LOCAL_VARIABLE })
```

тут мы говорим, аннотацию можно использовать только для параметров метода и для локальных переменных.

- **КАК СОЗДАТЬ СВОЮ АННОТАЦИЮ?**

Написать свою аннотацию не так сложно, как могло бы казаться. В следующем коде приведено объявление аннотации.

```
public @interface About{
    String info() default "";
}
```

как вы видите на месте где обычно пишут class или interface у нас написано @interface.

Структура практически та же, что и у интерфейсов, только пишется @interface.

- @interface – указывает на то, что это аннотация
- default – говорит про то, что метод по умолчанию будет возвращать определённое значение.

Аннотация готова теперь ею можно пользоваться, также аннотацию можно сконфигурировать.

- **АТРИБУТЫ КАКИХ ТИПОВ ДОПУСТИМЫ В АННОТАЦИЯХ?**

Атрибуты могут иметь только следующие типы:
примитивы

- String
- Class или «any parameterized invocation of Class»
- enum
- annotation
- массив элементов любого из вышеперечисленных типов

Последний пункт надо понимать как то, что допустимы только одномерные массивы.

- **ЧТО ТАКОЕ JMX?**

Управленческие расширения Java (Java Management Extensions, JMX) – API при помощи которого можно контролировать работу приложений и управлять различными параметрами удаленно в реальном времени. Причем управлять можно фактически чем угодно – лишь бы это было написано на Java. Это может быть микро-устройство типа считывателя отпечатка или система, включающая тысячи машин, каждая из которых предоставляет определенные сервисы. Данные ресурсы представляются MBean-объектами (управляемый Java Bean). JMX вошла в поставку Java начиная с версии 5.

- **КАКИЕ ВЫГОДЫ ПРЕДЛАГАЕТ JMX?**

Вот как эти выгоды описывает Sun

- Простота реализации. Архитектура JMX основана на понятии "сервера управляемых объектов" который выступает как управляющий агент и может быть запущен на многих устройствах/компьютерах, которые поддерживают JAVA.
- Масштабируемость. Службы агентов JXM являются независимыми и могут быть встроены наподобие plug-in'ов в агента JMX. Компонентно-основанаая система позволяет создавать масштабируемые решения от крохотных устройств до очень крупных систем.
- Возможность расширять концепцию в будущем. JMX позволяет создавать гибкие решения. Например, JMX позволяет создавать удобные решения, которые могут находить различные сервисы.
- Концентрация на управлении. JMX предоставляет сервися, разработанные для работы в распределенных средах и его API спроектировано для решений, которые управляют приложениями, сетями, сервисами и т.д.

- **ЧТО ЕЩЕ УМЕЕТ JMX КРОМЕ ДИСТАНЦИОННОГО УПРАВЛЕНИЯ?**

JMX делает гораздо больше, чем просто предоставляет рабочую оболочку для дистанционного управления. Она обеспечивает дополнительные услуги (services), способные занять ключевое место в процессе разработки. Приведу лишь краткое описание:

- Event notification: Интерфейсы оповещают исполнителей и слушателей о событиях типа изменения атрибута, что позволяет MBean-компонентам общаться с другими MBean-компонентами или удалённым "командным пунктом" и докладывать об изменениях своего состояния
- Monitor service: Monitor MBeans может посылать уведомления о событиях зарегистрированным слушателям. Слушателем может выступать другой MBean или управляющее приложение. В качестве основных атрибутов, для которых используется данное свойство, являются counter, gauge или string.
- Timer service: Timer MBean будет посылать уведомления зарегистрированным слушателям, с учётом определённого числа или временного промежутка.
- M-let service: M-let service может создавать и регистрировать экземпляры MBean-серверов. Список MBean-компонентов и имён из классов определяются в m-let-файле с помощью MLET -меток. URL указывает на месторасположения m-let-файла.

- **ЧТО ТАКОЕ MBEAN?**

MBeans - это Java-объекты, которые реализуют определенный интерфейс. Интерфейс включает:

1. некие величины, которые могут быть доступны;
2. операции, которые могут быть вызваны;
3. извещения, которые могут быть посланы;
4. конструкторы.

- **КАКИЕ ТИПЫ MBEANS СУЩЕСТВУЮТ?**

Существует 4 типа MBeans:

- Standard MBeans. Самые простые бины. Их управляющий интерфейс определяется набором методов
- Dynamic MBeans. Они реализуют специализированный интерфейс, который делают доступным во время исполнения.
- Open MBeans. Это Dynamic MBeans, которые используют только основные типы данных для универсального управления.
- Model MBeans. Это Dynamic MBeans, которые полностью конфигурируемы и могут показать свое описание во время исполнения (нечто вроде Reflection)

- **ЧТО ТАКОЕ MBEAN SERVER?**

MBean Server – это реестр объектов, которые используются для управления. Любой объект зарегистрированный на сервере становится доступным для приложений. Надо отметить, что сервер публикует только интерфейсы и не дает прямых ссылок на объекты. Любые ресурсы, которыми вы хотите управлять должны быть зарегистрированы на сервере как MBean. Сервер предоставляет стандартный интерфейс для доступа к MBean. Интересно, что регистрировать MBean может любой другой MBean, сам агент или удаленное приложение через распределенные сервисы. Когда вы регистрируете MBean вы должны дать ему уникальное имя, которое будет использовано для обращения к даному объекту.

- **КАКИЕ МЕХАНИЗМЫ ОБЕСПЕЧИВАЮТ БЕЗОПАСНОСТЬ В ТЕХНОЛОГИИ JAVA?**

В технологии Java безопасность обеспечивают следующие три механизма:

- структурные функциональные возможности языка (например, проверка границ массивов, запрет на преобразования непроверенных типов, отсутствие указателей и т.д.).
- средства контроля доступа, определяющие действия, которые разрешается или запрещается выполнять в коде (например, может ли код получать доступ к файлам, передавать данные по сети и т.д.).
- механизм цифровой подписи, предоставляющий авторам возможность применять стандартные алгоритмы для аутентификации своих программ, а пользователям – точно определять, кто создал код и изменился ли он с момента его подписания.

- **НАЗОВИТЕ НЕСКОЛЬКО ВИДОВ ПРОВЕРОК КОТОРЫЕ ВЫПОЛНЯЕТ ВЕРИФИКАТОР БАЙТ-КОДА JAVA?**

Ниже приведены некоторые виды проверок, выполняемых верификатором.

- инициализация переменных перед их использованием.
- согласование типов ссылок при вызове метода.
- соблюдение правил доступа к закрытым данным и методам.
- доступ к локальным переменным в стеке во время выполнения.
- отсутствие переполнения стека.

При невыполнении какой-нибудь из этих проверок класс считается поврежденным и загружаться не будет.

- **ЧТО ВЫ ЗНАЕТЕ О "ДИСПЕТЧЕРЕ ЗАЩИТЫ" В JAVA?**

В качестве диспетчера защиты служит класс, определяющий, разрешено ли коду выполнять ту или иную операцию. Ниже перечислены операции, подпадающие под контроль диспетчера защиты. Существует немало других проверок, выполняемых диспетчером защиты в библиотеке Java.

- создание нового загрузчика классов.
- выход из виртуальной машины.
- получение доступа к члену другого класса с помощью рефлексии.
- получение доступа к файлу.
- установление соединения через сокет.
- запуск задания на печать.
- получение доступа к системному буферу обмена.
- получение доступа к очереди событий в AWT.
- обращение к окну верхнего уровня.

- **ЧТО ТАКОЕ JAAS?**

JAAS (Java Authentication and Authorization Service – служба аутентификации и авторизации Java) – служба JAAS, по существу, представляет собой встраиваемый прикладной интерфейс API, отделяющий прикладные программы на Java от конкретной технологии, применяемой для реализации средств аутентификации. Помимо прочего, эта служба поддерживает механизмы регистрации в UNIX и NT, механизм аутентификации Kerberos и механизмы аутентификации по сертификатам.
После аутентификации за пользователем может быть закреплен определенный набор полномочий. Входит в состав платформы Java начиная с версии Java SE 1.4.

- **ЧТО ТАКОЕ РЕФАКТОРИНГ?**

Рефакторинг – процесс изменения внутренней структуры программы, не затрагивающий её внешнего поведения и имеющий целью облегчить понимание её работы. В основе рефакторинга лежит последовательность небольших эквивалентных (то есть сохраняющих поведение) преобразований..