Java

Аннотации, Reflection

Аннотации

- Аннотации - маркеры, хранящие некоторую информацию о коде
- Может использоваться для генерации данных
- Ключевое слово - @interface

Синтаксис

```
@Target(value = ElementType.FIELD)
@Retention(value = RetentionPolicy.RUNTIME)
public @interface MyAnnotation
{
    String name();
    String type() default  "string";
}
```

@Retention

@Retention - Определение жизненного цикла аннотации

- RetentionPolicy
  - SOURCE - Видна только в исходном коде (для программиста или IDE)
  - CLASS - Видна в скомпилированном классе (Обфускаторы, генераторы)
  - RUNTIME - Видна при исполнении

@Target

@Target - Определяет что мы можем пометить аннотацией

- ElementType
  - ANNOTATION_TYPE - другая аннотация
  - TYPE - класс
  - CONSTRUCTOR
  - FIELD
  - TYPE_USE
  - LOCAL_VARIABLE
  - METHOD
  - MODULE
  - PACKAGE
  - PARAMETER
  - TYPE_PARAMETER

@Documented

Включает аннотацию в документацию на сущность, у которой есть аннотация с @Documented

@Inherited

Помечает что аннотация будет унаследована

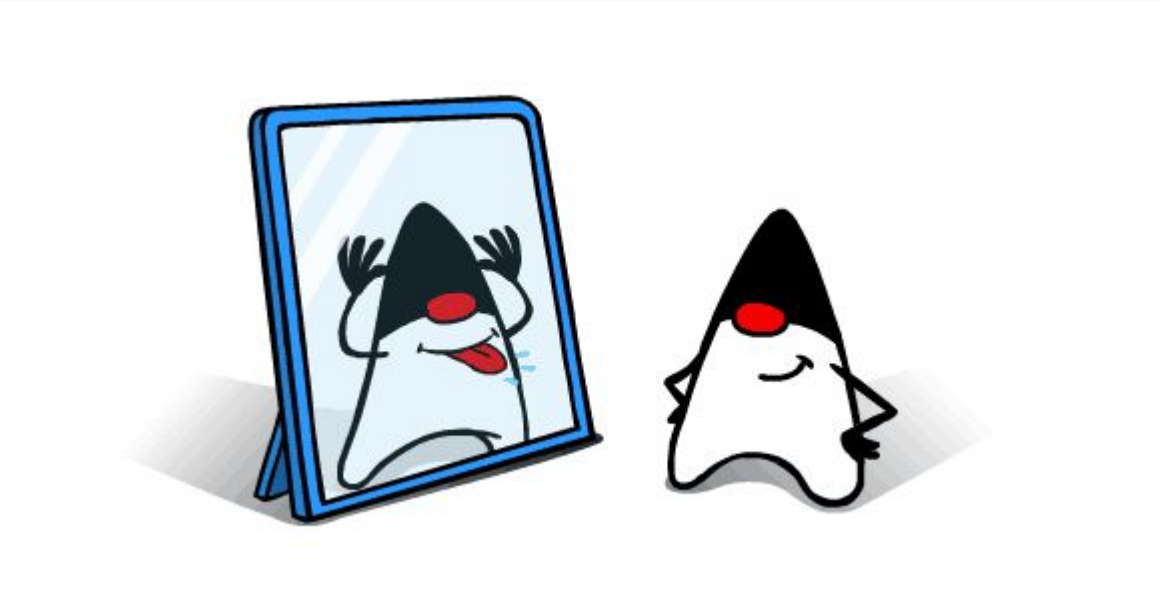
Распространённые аннотации процесса разработки

- @Override - Помечаем что метод переопределён
- @Nullable - Помечает что значение может быть null
- @Deprecated - Метод/Класс устарел
- @SupressWarning - Подавление предупреждений

Распространённые аннотации в рантайме

- @Test / @RunWith - Аннотации для тестирования
- @Inject - Добавление экземпляра объекта из DI
- @Singleton - Помечает что в DI объект является синглтоном
- @Optional - Помечает что в DI объект может не быть назначен (null)
- @Get / @Post /@Delete - Аннотации HTTP сервисов
- @Path - Аннотация пути доступа в URL

Reflection API



Reflection API

- Узнать реальный класс объекта
- Получить информацию о классе, поле или методе (видимость, параметры, типы, иерархия)
- Узнать состав класса
- Создать экземпляр произвольного класса
- Изменять значение и видимость поля или метода
- Вызвать метод по имени или сигнатуре

Reflection

- Используется для манипуляции над классами и объектами
- Часто используется для сериализации/десериализации
- Может использоваться фреймворками и библиотеками для дополнительной настройки сущностей

Класс Class<?>

- Можно получить у любого объекта через метод getClass() или поле class
- Можно загрузить по имени - Class.forName(..)

Методы Class

- cast(Object o)
- isInstance - является ли данным классом
- isAssignableFrom - переданный класс является этим классом или супер классом
- isAnnotation / isInterface / isArray / isEnum - Проверка типа класса
- getName() - Полное имя класса. Вложенные классы отделены $
- getSimpleName() - Короткое имя без пакета
- getCanonicalName() - Полное имя класса. Вложенные отделены “.”

Рефлексия через Class

- getFields() - Все публичные поля
- getDeclaredFields() - Все объявленные поля
- getField(String name) / getDeclaredField(String name)
- getMethods() …
- getConstructors() ...

Field

- get(Object o) - получить значение
- set(Object o, Object o) - установить значение
- get/set Accessible(boolean) - сменить видимость (теперь canAccess)
- isAnnotationPresent() - проверка наличия аннотации
- get<Primitive>(Object o) / set<Primitive>(Object o, ...) - получить примитивное значение

Method

- getReturnType()
- getParameterCount()
- getParameterTypes()
- invoke(Object o, Object args…)
- isAnnotationPresent()
- get/set Accessable()

Constructor

- Класс Class имеет метод newInstance() - он вызывает конструктор без параметров
- newInstance() - вызов через конкретный конструктор
- getParameterTypes() - получение параметров конструкторas