СБЕРБАНК ТЕХНОЛОГИИ

Annotations

УЗНАЕМ

СБЕРБАНК ТЕХНОЛОГИИ

- Что такое аннотации?
- Где они применяются?
- Как создать свою собственную аннотацию?

ANNOTATION

СБЕРБАНК ТЕХНОЛОГИИ

Аннотация - форма метаданных, которая представляет данные об элементе программы, но не является частью программы. Аннотации не имеют прямого влияния на работу аннотируемого кода.

Используются:

- Компилятором(@Override,@SuppressWarnings, @FuncionalInterface)
- Средой разработки(@Nullable,@Notnull,@Contract)
- Процессором аннотаций (Lombok - @Getter)
- Анализатором байт кода (Sonar - @SuppressWarnings)
- Во время выполнения

- Типы:
- Маркерные(без параметров)
- Параметризованные

ПРИМЕР ОПРЕДЕЛЕНИЯ АННОТАЦИИ

СБЕРБАНК ТЕХНОЛОГИИ

```
@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Cache{

}

public interface Calculator{
    @Cache
    int calc(int arg);
}
```

ДОСТУПНОСТЬ @RETENTION

СБЕРБАНК ТЕХНОЛОГИИ

- SOURCE – только в исходном коде
- CLASS – в исходном коде и байт коде (значение по умолчанию)
- RUNTIME – в исходном коде, байт коде и во время исполнения программы через reflection

ЦЕЛЕВЫЕ ОБЪЕКТЫ (TARGET)

СБЕРБАНК ТЕХНОЛОГИИ

TYPE – класс, интерфейс, аннотацию, перечисление
ANNOTATION_TYPE- только аннотации
FIELD – поле
METHOD – метод
PARAMETER – параметр метода
CONSTRUCTOR – конструктор
LOCAL_VARIABLE- локальную переменную
PACKAGE – packageinfo.java и использовать там этот параметр
TYPE_PARAMETER – параметр типа, в угловых скобках (java 8)
TYPE_USE – любое использование типа (java 8)

ПАРАМЕТРЫ АННОТАЦИЙ

СБЕРБАНК ТЕХНОЛОГИИ

- Примитивные типы
- Class, Class<? extends X>
- String
- Enum
- Аннотация (без циклических зависимостей)
- Одномерный массив из чего-то перечисленного

Все это значения – константы времени компиляции

СБЕРБАНК ТЕХНОЛОГИИ

Reflection

УЗНАЕМ

СБЕРБАНК ТЕХНОЛОГИИ

- Какую метаинформацию можно получить в рантайме о классах?
- Можно ли звать приватные методы класса из других классов?
- Зачем и как это делать ?

REFLECTION

СБЕРБАНК ТЕХНОЛОГИИ

Reflection – это функционал языка Java, который позволяет получить информацию о программе из программы, «анализировать» себя, а также управлять внутренним состояние программы.

- Иерархия класса
- Методы (список, тип возвращаемого значения, имена и типы аргументов, модификаторы видимости)
- Поля (список, имена, типы, модификаторы видимости)
- Типы
- Аннотации
- Пакеты
- Возможность вызвать методы, изменить поля
- Модули (Java 9+)

КЛАСС JAVA.LANG.CLASS

СБЕРБАНК ТЕХНОЛОГИИ

Содержит методы для получения полной информации о классе, вызова методов и изменения полей.

```
Class<Integer> c = Integer.class;


Class<String> c = String.class;


someObject.getClass();
```

Примеры, которые мы рассмотрим:

СБЕРБАНК ТЕХНОЛОГИИ

- Создание класса через рефлекшен
- Вызов метода через рефлекшен
- Получим список всех полей и проверим их значение на null
- Склонируем состояние одного объекта в другой объект

ВАЖНЫЕ МЕТОДЫ КЛАССА

СБЕРБАНК ТЕХНОЛОГИИ

```
//Список всех public методов, объявленных в классе или унаследованных
public Method[] getMethods()


//Список всех методов, объявленных в классе
public Method[] getDeclaredMethods()
```

ВАЖНЫЕ МЕТОДЫ КЛАССА

СБЕРБАНК ТЕХНОЛОГИИ

```
//Метод с заданным именем и аргументами
public Method getMethod (String name,
                Class<?>...parameterTypes)
```

```
Method m = String.class.getMethod("replaceAll",
        String.class, String.class)
```

ВАЖНЫЕ МЕТОДЫ КЛАССА

СБЕРБАНК ТЕХНОЛОГИИ

```
//Список всех public полей, объявленных в классе или унаследованных
public Field[] getFields()

//Список всех полей, объявленных в классе
public Field[] getDeclaredFields()

//поле по имени
public Field getField(String name)

//Поле, объявленное в классе
public Field getDeclaredField(String name)
```

ВАЖНЫЕ МЕТОДЫ КЛАССА

СБЕРБАНК ТЕХНОЛОГИИ

```
//Возвращает класс родителя
public native Class<? super T> getSuperclass();
```

ЧТО БУДЕТ НА КОНСОЛИ?

СБЕРБАНК ТЕХНОЛОГИИ

```
System.out.println(String.class.getSuperclass());

System.out.println(Object.class.getSuperclass());
```

У КЛАССА OBJECT SUPERCLASS == NULL

СБЕРБАНК ТЕХНОЛОГИИ

```
//Object
System.out.println(String.class.getSuperclass());

//null
System.out.println(Object.class.getSuperclass());
```

ПОЛУЧЕНИЕ ПОЛНОЙ ИЕРАРХИИ

СБЕРБАНК ТЕХНОЛОГИИ

```
public static void printHierarchy(Class<?> clazz) {
  while (clazz != null) {
    System.out.println(clazz);
    clazz = clazz.getSuperclass();
  }
}
```

КАК СОЗДАТЬ ЭКЗЕМПЛЯР КЛАССА ?

СБЕРБАНК ТЕХНОЛОГИИ

```
try {
    //Зовется конструктор без параметров
    Person p = Person.class.newInstance();
} catch(InstantiationException | IllegalAccessException e){
  …
}
```

КАК СОЗДАТЬ ЭКЗЕМПЛЯР КЛАССА ?

СБЕРБАНК ТЕХНОЛОГИИ

```
// Зовется конструктор со String аргументом
Person p2 = Person.class.getConstructor(String.class)
            .newInstance("Alex");
```

ВЫЗОВ МЕТОДА

СБЕРБАНК ТЕХНОЛОГИИ

```
private void setName(Object o, String name)
    throws NoSuchMethodException, InvocationTargetException, IllegalAccessException {

  Class<?> clazz = o.getClass();
  Method m = clazz.getMethod("setName", String.class);


  //мы передаем объект, у которого вызовется метод,
                          и параметры метода
  m.invoke(o, name);

}
```

ВЫЗОВ ПРИВАТНОГО МЕТОДА

СБЕРБАНК ТЕХНОЛОГИИ

```
Method m = clazz.getDeclaredMethod("setName",
                String.class);
m.setAccessible(true);
m.invoke(o, name);
```

ИЗМЕНЕНИЕ FINAL ПОЛЕЙ

СБЕРБАНК ТЕХНОЛОГИИ

```
public class Person {
  private final String name;//Можно поменять?

        ...
}
```

МОЖНО!

СБЕРБАНК ТЕХНОЛОГИИ

```
Person person = get();

Field name = Person.class.getDeclaredField("name");
name.setAccessible(true);
name.set(person, "Julia");
```

ИСПОЛЬЗОВАНИЕ REFLECTION

СБЕРБАНК ТЕХНОЛОГИИ

- Динамическая загрузка (плагины, расширения)
- Поддержка нескольких версий зависимостей в runtime
- Dependency injection
- Сериализация
- Грязные хаки

ДЖЕНЕРИКИ ЧЕРЕЗ REFLECTION

СБЕРБАНК ТЕХНОЛОГИИ

Можно достать метаинформацию о дженериках на уровне **класса**.

Информация, чем параметризованны **локальные объекты** стирается.
Нельзя узнать стертый тип в рантайм

ДЖЕНЕРИКИ, ДОСТУПНЫЕ ЧЕРЕЗ REFLECTION

СБЕРБАНК ТЕХНОЛОГИИ

```
public class Runtime<T extends Number>
                    implements Callable<Double> {
    private final List<Integer> integers = emptyList();

    public List<T> numbers() {return emptyList();}

    public List<String> strings() {return emptyList();}

    @Override
  public Double call() {return 0d;}
}
```

ПОЛУЧЕНИЕ ИНФОРМАЦИИ ОБ АННОТАЦИЯХ

СБЕРБАНК ТЕХНОЛОГИИ

```
Field f = …

if (f.isAnnotationPresent(ValidLength.class)) {
  ValidLength an=f.getAnnotation(ValidLength.class);
  int max = an.max();
  int min = an.min();
      …
}
```

ПРИМЕР ПРОВЕРКИ ПОЛЕЙ ПО АННОТАЦИЯМ

СБЕРБАНК ТЕХНОЛОГИИ

```
public void validateStringLength(Object o) throws Exception {
  Class<?> clazz = o.getClass();
  for (Field field : clazz.getDeclaredFields()) {
    if (field.isAnnotationPresent(ValidLength.class)) {
     ValidLength an= field.getAnnotation(ValidLength.class);
      int max = an.max();
      int min = an.min();

      String value = field.get(o).toString();
      if (value.length() < min) {
        throw new IllegalStateException(field.getName()
            + " length should be between " + min + " and " + max);
                }
            }
      }
}
```

СБЕРБАНК ТЕХНОЛОГИИ

Proxy

DYNAMIC PROXY

СБЕРБАНК ТЕХНОЛОГИИ

Позволяет перехватывать в рантайме вызовы методов интерфейса и обрабатывать их.

Прокси может притворяться любым интерфейсом.

ПРИМЕР: КЕШИРУЮЩИЙ PROXY

СБЕРБАНК ТЕХНОЛОГИИ

Кеширующий прокси перехватывает вызовы интерфейса.

Если метод помечен аннотацией @Cache, то:
Проверяет есть ли в кеше результат, если есть, то возвращает его.
Иначе, вызывает реальный метод, кеширует результат и возвращает его.

Если метод не помечен аннотацией @Cache, просто делегирует метод реализации

ПРИМЕР CACHE PROXY

СБЕРБАНК ТЕХНОЛОГИИ

```
Calculator calculator = new CalculatorImpl();
calculator.calc(1);
calculator.calc(1); // повторный расчет


Calculator cached = ProxyUtils.makeCached(calculator);
cached.calc(1);
cached.calc(2);
cached.calc(1); // результат из кеша
```

ПРИМЕР: RMI ПРОКСИ

СБЕРБАНК ТЕХНОЛОГИИ

Прокси перехватывает вызовы интерфейса и перенаправляет их по сети другому серверу и возвращает результат.

```
Calculator calc = ProxyUtils.client(Calculator.class);
calc.calc(1); // перехват вызова и отправка удаленной
                                машине

Service service = ProxyUtils.client(Service.class);
service.run();
```

ПРИМЕР: RMI ПРОКСИ

СБЕРБАНК ТЕХНОЛОГИИ

Это позволяет быстро создать клиент любого интерфейса(На удаленной машине должны быть слушатели вызова, созданные, например, тоже через Proxy).

```
Calculator calc = ProxyUtils.client(Calculator.class);
calc.calc(1); // перехват вызова и отправка удаленной
машине

Service service = ProxyUtils.client(Service.class);
service.run();
```

КАК СОЗДАТЬ ПРОКСИ. java.lang.reflect.Proxy

СБЕРБАНК ТЕХНОЛОГИИ

```
public class Proxy {

//возвращает объект, который реализует интерфейсы interfaces[]
//вызов методов передается в реализацию InvocationHandler
public static Object newProxyInstance(ClassLoader loader,
                                       Class<?>[] interfaces,
                                       InvocationHandler h)
}
```

INVOCATION HANDLER

СБЕРБАНК ТЕХНОЛОГИИ

Поведение прокси задается в реализации интерфейса InvocationHandler

```
public interface InvocationHandler {
    Object invoke(Object proxy, Method method,
                  Object[] args)  throws Throwable;
}
```

ПРИМЕР LOG ХЕНДЛЕРА

СБЕРБАНК ТЕХНОЛОГИИ

```
public class LogHandler implements InvocationHandler {
    private final Object delegate;

    public LogHandler(Object delegate) {
        this.delegate = delegate;
    }

    @Override
    public Object invoke(Object proxy, Method method,
                Object[]args) throws Throwable {
        System.out.println("Started " + method.getName());
        Object result = method.invoke(delegate, args);
        System.out.println("Finished " + method.getName() + ".
              Result " + result);
        return result;
    }
```

ДОБАВЛЯЕМ ЛОГГИРОВАНИЕ ВСЕХ МЕТОДОВ ЛИСТА

СБЕРБАНК ТЕХНОЛОГИИ

```
List<String> loggedList = (List<String>)
      Proxy.newProxyInstance(
         ClassLoader.getSystemClassLoader(),
         new Class[]{List.class},
            new LogHandler(new ArrayList<String>())
            );


//реализация методов интерфейса List зависит
//от передаваемого класса в конструкторе LogHandler
```

ДЗ

СБЕРБАНК ТЕХНОЛОГИИ

Просмотреть основные моменты работы с reflection и dynamic proxy: http://tutorials.jenkov.com/java-reflection/index.html

Реализовать следующий класс по документации

```
public class BeanUtils {
   /**
    * Scans object "from" for all getters. If object "to"
    * contains correspondent setter, it will invoke it
    * to set property value for "to" which equals to the property
    * of "from".
    * <p/>
    * The type in setter should be compatible to the value returned
    * by getter (if not, no invocation performed).
    * Compatible means that parameter type in setter should
    * be the same or be superclass of the return type of the getter.
    * <p/>
    * The method takes care only about public methods.
    *
    * @param to   Object which properties will be set.
    * @param from Object which properties will be used to get values.
    */
   public static void assign(Object to, Object from) {... }
}
```