Язык Java

Глава VII. Рефлексия

Виталий Витальевич Перевощиков

Осенний семестр 2021

Мотивация

- Можно ли писать динамический код на Java?

```java
interface DynamicObject {
    // возвращает значение, хранящееся в поле fieldName объекта obj
    Object getValue(Object obj, String fieldName);

    // присваивает полю fieldName объекта obj значение value
    void setValue(Object obj, String fieldName, String value);

    // создает объект класса с названием className
    Object createObject(String className);
}
```

- Можно ли получить доступ к приватным полям?

```java
class User {
    private String password;
}

class Hacker {
    String getPassword(User user) {
        return user.password;
    }
}
```

Рефлексия

- **Рефлексия** - механизм исследования данных о программе во время ее выполнения.
- Рефлексия позволяет:
  - определить класс объекта;
  - получить информацию о модификаторах доступа, полях, методах, конструкторах, ...;
  - создать объект класса по имени;
  - получить и установить значение поля объекта по имени;
  - вызвать метод объекта по имени;
  - ...
- Реализация рефлексии в Java: пакет `java.lang.reflect`

Информация о классе

```java
package ru.bfu.ipmit;

class Student {
}

public class Main {

    public static void main(String[] args) {
        Object object = new Student();

        Class clazz = object.getClass(); // класс для получения данных о классе в процессе выполнения программы

        String className = clazz.getName(); // полное имя класса: "ru.bfu.ipmit.Student"
        String simpleClassName = clazz.getSimpleName(); // простое имя класса: "Student"
        String packageName = clazz.getPackageName(); // пакет, содержащий класс Student: "ru.bfu.ipmit"

    }
}
```

Информация о публичных методах

```java
import java.lang.reflect.Method;

class Student {
    public String getName() { return "Иван"; }
    private void setName(String name) { /* сделать что-то */ }
}

public class Main {

    public static void main(String[] args) {
        Class<Student> clazz = Student.class;

        Method[] methods = clazz.getMethods(); // все публичные методы класса
        for (Method method : methods) {
            System.out.println(method.getName());
        }
    }
}
```

```
getName
wait
wait
wait
equals
toString
hashCode
getClass
notify
notifyAll
```

Методы класса-родителя Object

Информация об объявленных методах

```java
import java.lang.reflect.Method;

class Student {
    public String getName() { return "Иван"; }
    private void setName(String name) { /* сделать что-то */ }
}

public class Main {

    public static void main(String[] args) {
        Class<Student> clazz = Student.class;

        Method[] methods = clazz.getDeclaredMethods(); // все объявленные методы класса
        for (Method method : methods) {
            System.out.println(method.getName());
        }
    }
}
```

```
getName
setName
```

← Все методы, объявленные в классе Student

Информация о модификаторах

```java
import java.lang.reflect.Method;
import java.lang.reflect.Modifier;

class Student {
    public static String getName() { return "Иван"; }
}

public class Main {

    public static void main(String[] args) throws NoSuchMethodException {
        Class<Student> clazz = Student.class;

        Method method = clazz.getMethod("getName");

        // модификаторы метода
        int fieldModifiers = method.getModifiers();
        boolean isPublic = Modifier.isPublic(fieldModifiers);      // true
        boolean isPrivate = Modifier.isPrivate(fieldModifiers);    // false
        boolean isStatic = Modifier.isStatic(fieldModifiers);      // true
        boolean isAbstract = Modifier.isAbstract(fieldModifiers); // false
    }
}
```

Вызов метода

```java
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;

class Student {
    private String name;
    private Integer age;

    public void setInfo(String name, Integer age) {
        this.name = name;
        this.age = age;
    }

    public String getInfo() {
        return "имя: " + name + ", возраст: " + age;
    }
}

public class Main {

    public static void main(String[] args) throws NoSuchMethodException, InvocationTargetException, IllegalAccessException {
        Student student = new Student();
        Class clazz = student.getClass();

        Method method = clazz.getMethod("setInfo", String.class, Integer.class);
        method.invoke(student, "Иван", 20); // равносильно student.setInfo("Иван", 20)


        String info = student.getInfo(); // "имя: Иван, возраст: 20"
    }
}
```

Типы аргументов метода

Значения аргументов метода

Информация о полях класса

```java
import java.lang.reflect.Field;

class Student {
    public String role;
    private String name;
    private Integer age;
}

public class Main {

    public static void main(String[] args) {
        Student student = new Student();
        Class clazz = student.getClass();

        Field[] publicFields = clazz.getFields(); // все публичные поля
        for (Field field : publicFields) {
            System.out.println(field.getName()); // выведет role
        }

        Field[] declaredFields = clazz.getDeclaredFields(); // все объявленные поля
        for (Field field : declaredFields) {
            System.out.println(field.getName()); // выведет role, name, age
        }
    }
}
```

Доступ к значениям полей класса

```java
import java.lang.reflect.Field;

class Student {
    private int age = 20;
}

public class Main {

    public static void main(String[] args) throws NoSuchFieldException, IllegalAccessException {
        Student student = new Student();
        Class clazz = student.getClass();

        Field field = clazz.getDeclaredField("age");
        field.setAccessible(true); // для получения доступа к значению приватного поля
        int age = (int)field.get(student); // 20
    }
}
```

Вызов конструкторов

```java
import java.lang.reflect.Constructor;
import java.lang.reflect.InvocationTargetException;

class Student {
    private String name;
    private Integer age;

    private Student() { name = "Иван"; age = 20; }

    public Student(String name, Integer age) { this.name = name; this.age = age; }

    public String getInfo() {
        return "имя: " + name + ", возраст: " + age;
    }
}

public class Main {

    public static void main(String[] args) throws NoSuchMethodException, InvocationTargetException, InstantiationException, IllegalAccessException {
        Class<Student> clazz = Student.class;

        // создание объекта с помощью приватного пустого конструктора
        Constructor<Student> emptyConstructor = clazz.getDeclaredConstructor();
        emptyConstructor.setAccessible(true);
        Student ivan = emptyConstructor.newInstance();
        System.out.println(ivan.getInfo()); // "имя: Иван, возраст: 20"

        // создание объекта с помощью конструктора с аргументами
        Constructor<Student> constructorWithParams = clazz.getDeclaredConstructor(String.class, Integer.class);
        Student boris = constructorWithParams.newInstance("Борис", 25);
        System.out.println(boris.getInfo()); // "имя: Борис, возраст: 25"
    }
}
```

Информация о реализуемых интерфейсах

```java
interface Person {
    String getName();
}

interface User {
    String getPassword();
}

class Student implements Person, User {
    public String getName() { return "Иван"; }
    public String getPassword() { return "qwerty"; }
}

public class Main {

    public static void main(String[] args) {
        Class<Student> clazz = Student.class;

        Class[] interfaces = clazz.getInterfaces();
        for (Class inter : interfaces) {
            System.out.println(inter.getSimpleName()); // выведет Person, User
        }
    }
}
```

Информация о классах-родителях

```java
class Student {
    private String getName() { return "Иван"; }
}

class ProgrammingStudent extends Student {
    private String getGithubLink() { return "github.com/ivan"; }
}

public class Main {

    public static void main(String[] args) {
        Class<ProgrammingStudent> clazz = ProgrammingStudent.class;

        for (Method method : clazz.getDeclaredMethods()) {
            System.out.println(method.getName()); // выведет getGithubLink
        }

        Class parentClazz = clazz.getSuperclass(); // "отец" класса ProgrammingStudent
        String parentClazzName = parentClazz.getSimpleName(); // "Student"
        for (Method method : parentClazz.getDeclaredMethods()) {
            System.out.println(method.getName()); // выведет getName
        }

        Class grandparentClazz = parentClazz.getSuperclass(); // "дед" класса ProgrammingStudent
        String grandparentClazzName = grandparentClazz.getSimpleName(); // "Object"

        Class greatGrandparentClazz = grandparentClazz.getSuperclass(); // null
    }
}
```

Аннотация

- **Аннотация** - это форма метаданных, представляющая дополнительную информацию об элементах программы (классах, полях, методах, аргументах, ...)
- Пример: Spring REST controller

```java
@RestController
@RequestMapping(value = "/api/v1/students")
public class StudentController {

    @Autowired
    private StudentService studentService;

    @RequestMapping(value = "/{id}", method = RequestMethod.GET)
    @ResponseStatus(HttpStatus.OK)
    public @ResponseBody Student getStudent(@PathVariable("id") Long id) {
        return studentService.getStudent(id);
    }

}
```

Виды аннотаций

- Аннотации-маркеры

```java
@RestController
@RequestMapping(value = "/api/v1/students")
public class StudentController {

    @Autowired
    private StudentService studentService;

    @RequestMapping(value = "/{id}", method = RequestMethod.GET)
    @ResponseStatus(HttpStatus.OK)
    public @ResponseBody Student getStudent(@PathVariable("id") Long id) {
        return studentService.getStudent(id);
    }

}
```

- Аннотации с одним параметром

```java
@RestController
@RequestMapping(value = "/api/v1/students")
public class StudentController {

    @Autowired
    private StudentService studentService;

    @RequestMapping(value = "/{id}", method = RequestMethod.GET)
    @ResponseStatus(HttpStatus.OK)
    public @ResponseBody Student getStudent(@PathVariable("id") Long id) {
        return studentService.getStudent(id);
    }

}
```

Виды аннотаций

- Полные аннотации

```java
@RestController
@RequestMapping(value = "/api/v1/students")
public class StudentController {

    @Autowired
    private StudentService studentService;

    @RequestMapping(value = "/{id}", method = RequestMethod.GET)
    @ResponseStatus(HttpStatus.OK)
    public @ResponseBody Student getStudent(@PathVariable("id") Long id) {
        return studentService.getStudent(id);
    }

}
```

- Аннотации типов

```java
@RestController
@RequestMapping(value = "/api/v1/students")
public class StudentController {

    @Autowired
    private StudentService studentService;

    @RequestMapping(value = "/{id}", method = RequestMethod.GET)
    @ResponseStatus(HttpStatus.OK)
    public @ResponseBody Student getStudent(@PathVariable("id") Long id) {
        return studentService.getStudent(id);
    }

}
```

Создание пользовательских аннотаций

- Пример: сериализация объектов в JSON

Java-объект

```
class Student {
    private String firstName = "Иван";
    private String lastName = "Иванов";
    private int age = 20;
    // игнорировать при сериализации в JSON
    private String password = "123456";
}
```

JSON-объект

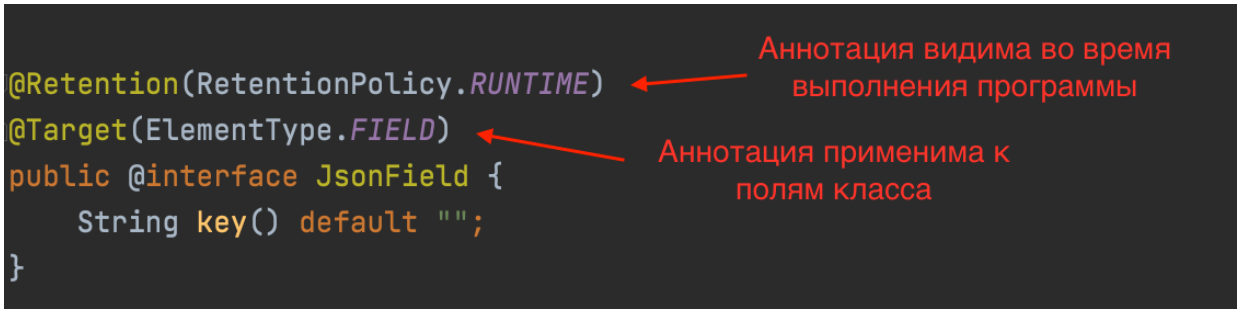
```
{
  "first_name": "Иван",
  "last_name": "Иванов",
  "age": 20
}
```

- Использование аннотаций:

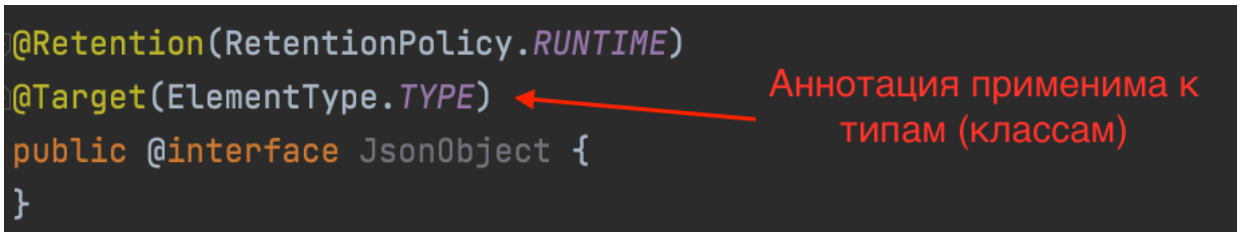
```
@JsonObject
class Student {
    @JsonField(key = "first_name")
    private String firstName = "Иван";
    @JsonField(key = "last_name")
    private String lastName = "Иванов";
    @JsonField
    private int age = 20;

    private String password = "123456";
}
```

Создание пользовательских аннотаций

- Аннотация для полей класса



- Аннотация для класса



Пример использования аннотаций

```java
// аннотации класса
Annotation[] classAnnotations = Student.class.getAnnotations();
String name = classAnnotations[0].annotationType().getName(); // "ru.bfu.ipmit.JsonObject"
```

```java
// аннотации полей
for (Field field : Student.class.getDeclaredFields()) {
    for (Annotation fieldAnnotation : field.getAnnotations()) {
        String fieldName = field.getName();
        String annotationName = fieldAnnotation.annotationType().getName();
        System.out.println(fieldName + ": " + annotationName);
    }
}
```

```
firstName: ru.bfu.ipmit.JsonField
lastName: ru.bfu.ipmit.JsonField
age: ru.bfu.ipmit.JsonField
```