THE SYS4U DODUMENT

Java Reflection & Introspection

김진아 사원

목차

I. 개념

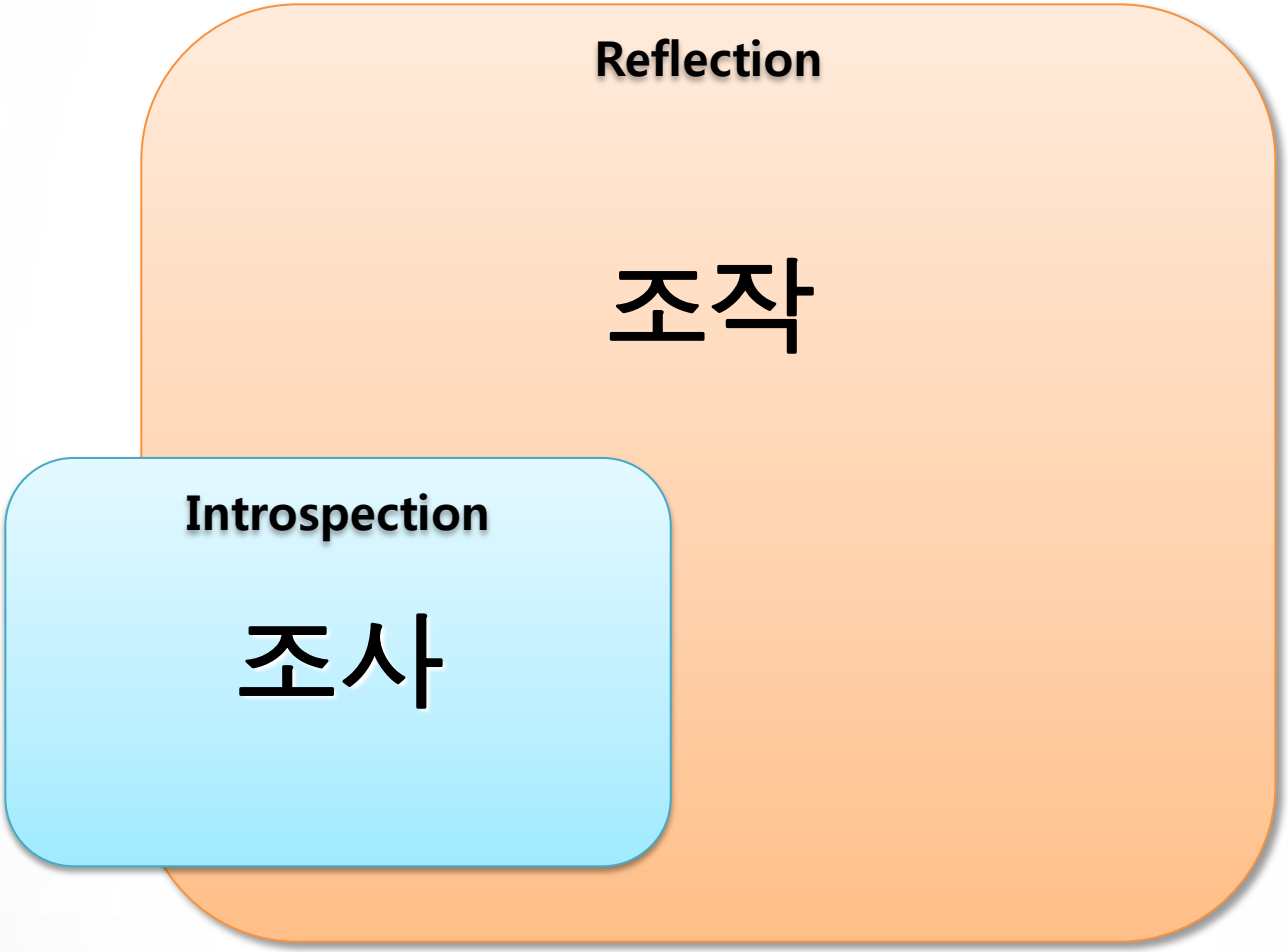
Java.lang.reflect 패키지 사용

클래스 인스턴스로부터 그 인스턴스가 표현하는 멤버, 필드, 메소드에 접근할 수 있는 기능

클래스 구조에 대한 정보 뿐만 아니라 객체의 생성, 메소드 호출, 필드 접근까지 가능

빈의 프로퍼티와 메소드 등을 알려주기 위해 빈의 디자인 패턴을 자동으로 분석하는 과정

객체의 클래스, 구현 메소드, 필드 등의 객체정보를 조사하는 과정을 의미


Reflection
조작
Introspection
조사

II. 실제 사용 예

- new

```
public class Sample {

    private void outputMessage(){

            System.out.println("outputMessage");
    }

    public static void main(String[] args) throws Exception  {

            Sample sample1 = new Sample();
            sample1.outputMessage();

    }
}
```

- 실행결과

```
outputMessage
```

- clone

```
public class Sample implements Cloneable{

    private void outputMessage(){

            System.out.println("outputMessage");
    }

    public static void main(String[] args) throws Exception  {
            Sample sample1 = new Sample();
            Sample sample3 = (Sample) sample1.clone();
            sample3.outputMessage();
    }
}
```

- 실행결과

```
outputMessage
```

- Class<T>.newInstance

```
public class Sample {

    private void outputMessage(){

            System.out.println("outputMessage");
    }

    public static void main(String[] args) throws Exception  {

            Class c = Class.forName("Sample");
            Sample sample2 = c.newInstance();
            sample2.outputMessage();
    }
}
```

- 실행결과

```
outputMessage
```

```
public static void main(String[] args) {
        Class c1 = Class.forName("java.lang.String");
}
```

```
public static void main(String[] args) {
        Class<Integer> c2 = int.class;
}
```

```
public static void main(String[] args) {
        Class<Integer> c3 = Integer.TYPE;
}
```

```
public class A {

    public static void main(String[] args) throws Exception  {

            Class c = Class.forName("A");

            boolean b1 = c.isInstance(new Integer(65));
            System.out.println("new Integer(65)  == >  "+b1);


            boolean b2 = c.isInstance(new A());
            System.out.println("new A()          == >  "+b2);
    }
}
```

- 실행결과

```
new Integer(65)  == >  false
new A()          == >  true
```

- 실행결과

```
Class Name       :  class Const
Constructor Name :  Const
-----------------------------------------------
param 0 :  int
param 1 :  class java.lang.String
```

```
Class : class F
Field Name : a
Field Type : int
Field Modifiers : private
---------------------------------------------
Class : class F
Field Name : b
Field Type : class java.lang.String
Field Modifiers : public static final
---------------------------------------------
Class : class F
Field Name : c
Field Type : double
Field Modifiers :
---------------------------------------------
```

```
Class Name : class M
Method Name : main
Param 0 : class [Ljava.lang.String;
exc 0 : class java.lang.Exception
ReturnType : void
-----------------------------------------
Class Name : class M
Method Name : test
Param 0 : class java.lang.String
Param 1 : int
exc 0 : class java.lang.NullPointerException
ReturnType : int
-----------------------------------------
```

Ⅶ. Method의 이름으로 Method 실행하기 (1/2)

```
public class A {
      public int add(int a, int b) { return a+b; }
}
```

```
public class B {
    public static void main(String[] args) throws Exception  {

        Class c = Class.forName("A");

        Class param[] = new Class[2];
        param[0] = Integer.TYPE;
        param[1] = Integer.TYPE;

        Method method = c.getDeclaredMethod("add", param);

        A a = new A();
        Object argList[] = new Object[2];
        argList[0] = new Integer(1);
        argList[1] = new Integer(1);
        Object obj = method.invoke(a, argList);
        Integer value = (Integer) obj;

       System.out.println(value.intValue());
    }
}
```

- 실행결과

2

Ⅶ. Method의 이름으로 private Method 실행하기 (2/2)

```
public class A {
    private int add(int a, int b){ return a+b; }
}
```

```
public class B {
    public static void main(String[] args) throws Exception  {

        Class c = Class.forName("A");

        Class param[] = new Class[2];
        param[0] = Integer.TYPE;
        param[1] = Integer.TYPE;

        Method method = c.getDeclaredMethod("add", param);

        A a = new A();
        method.setAccessible(true);
        Object argList[] = new Object[2];
        argList[0] = new Integer(1);
        argList[1] = new Integer(1);
        Object obj = method.invoke(a, argList);
        Integer value = (Integer) obj;

        System.out.println(value.intValue());
    }
}
```

- 실행결과

```
2
```

```
public class CF {

        public int i;

public static void main(String[] args) throws Exception  {

        Class c = Class.forName("CF");

        Field field = c.getField("i");

        CF cf = new CF();
        System.out.println ("set 하기 전 : " + cf.i);

         field.setInt(cf, 1);

         System.out.println ("----------------------------------------");
         System.out.println("set 한 후   : " +cf.i);          }

  }
}
```

- 실행결과

```
set 하기 전 : 0
----------------------------------------
set 한 후   : 1
```

```
public class CF {

    public static void main(String[] args) throws Exception  {

        Class c = Class.forName("java.lang.String");

        Object array = Array.newInstance(c, 10);

        Array.set(array, 5, "test");

        String str = (String) Array.get(array, 5);

        System.out.println(str);
    }
}
```

- 실행결과

```
test
```

```
public class PizzaFactoryy {
        public static final String PIZZA="Pizza";

        public Pizza getPizzaInstance(String pizzaType){
            Pizza pizza=null;
            try {

                String type = new StringBuffer(pizzaType).append(PIZZA).toString();

                Class c = Class.forName(type);
                pizza = (Pizza) c.newInstance();

            } catch (Exception e) {
                throw new IllegalArgumentException("No Such Pizza ["+pizzaType+"]");
            }
            return pizza;
        }

        public static void main(String[] args) {
            PizzaFactoryy f = new PizzaFactoryy();
            Pizza pizza = f.getPizzaInstance("Cheese");
            pizza.getPizzaName();
          }
}
```

III. 관련 라이브러리

cloneBean(Object bean)

Return	Object
Parameters	복제 할 bean
설명	해당 빈 복제

```
CloneBean bean = new CloneBean();

CloneBean clone = (CloneBean) BeanUtils.cloneBean(bean);
```

copyProperties(Object dest, Object orig)

Return	void
Parameters	원본 bean, 대상 bean,
설명	원본 대상 빈의 속성과 이름이 모두 같을 경우 속성값 복사

- 멤버 필드

```
public int i;
public String str;
```

```
CopyProperties1 orig = new CopyProperties1();
orig.setI(1);
orig.setStr("시스포유");
System.out.println("orig bean :" + orig);

CopyProperties1 dest = new CopyProperties1();
BeanUtils.copyProperties(dest, orig);
System.out.println("dest bean :" + dest);
```

- 실행결과

```
orig bean :CopyProperties1 [i=1, str=시스포유]

dest bean :CopyProperties1 [i=1, str=시스포유]
```

copyProperties(Object bean, String name, Object value)

Return	void
Parameters	원본 bean, 속성 이름, 대상 bean
설명	원본 빈의 해당 속성값을 대상 빈에 복사

- 멤버 필드

```
public int i;
public String str;
```

```
CopyProperties2 bean = new CopyProperties2();
System.out.println("copy 하기 전 bean : " + bean);

BeanUtils.copyProperty(bean, "str", "시스포유");

System.out.println("copy 한 후  bean  : " + bean);
```

- 실행결과

```
copy 하기 전 bean : CopyProperties2 [i=0, str=null]
copy 한    후 bean : CopyProperties2 [i=0, str=시스포유]
```

describe(Object bean)

Return	Map
Parameters	bean
설명	해당 빈의 속성이름과 속성값을 맵으로 리턴

- 멤버 필드

```
public int i;
public String str;
```

```
Describe bean = new Describe();
bean.setI(1);
bean.setStr("시스포유");

Map<String, String> map = BeanUtils.describe(bean);

System.out.println(map);
```

- 실행결과

```
{str=시스포유, class=class commons_beanutils.Describe, i=1}
```

getArrayProperty(Object bean, String name)

Return	String[]
Parameters	Bean, 배열의 속성 이름
설명	해당 빈의 배열속성값을 리턴

- 멤버 필드

```
public String[] arrayStr = new String[] {"시스포유", "김진아"};
```

```
GetArrayProperty bean = new GetArrayProperty();

String[] arrayStr = BeanUtils.getArrayProperty(bean, "arrayStr");

System.out.println("arrayStr[0] : " + arrayStr[0]);
System.out.println("arrayStr[1] : " + arrayStr[1]);
```

- 실행결과

```
arrayStr[0] : 시스포유

arrayStr[1] : 김진아
```

getIndexedProperty(Object bean, String name)

Return	String
Parameters	Bean, 배열 속성 이름
설명	빈의 배열 요소 리턴

- 멤버 필드

```
public String[] arrayStr = new String[] {"시스포유", "김진아"};
```

```
GetIndexedProperty bean = new GetIndexedProperty();

String value = BeanUtils.getIndexedProperty(bean, "arrayStr[0]");

System.out.println("arrayStr[0] : " + value);
```

- 실행결과

```
arrayStr[0] : 시스포유
```

getIndexedProperty(Object bean, String name, int index)

Return	String
Parameters	Bean, 배열 속성 이름, 인덱스
설명	빈의 해당 배열의 지정된 인덱스값을 리턴

- 멤버 필드

```
public String[] arrayStr = new String[] {"시스포유", "김진아"};
```

```
GetIndexedProperty2 bean = new GetIndexedProperty2();

String value = BeanUtils.getIndexedProperty(bean, "arrayStr" , 1);

System.out.println("arrayStr[1] : " + value);
```

- 실행결과

```
arrayStr[1] : 김진아
```

getMappedProperty(Object bean, String name)

Return	String
Parameters	Bean, 맵 속성이름
설명	빈의 해당 맵의 지정된 인덱스값을 리턴

- 멤버 필드

```
Public Map<String, String> map = new HashMap<String, String>();
  map.put("company", "시스포유");
  map.put("name", "김진아");
```

```
GetMappedProperty1 bean = new GetMappedProperty1();

String value = BeanUtils.getMappedProperty(bean, "map(company)");

System.out.println("map(company) : " + value);
```

- 실행결과

```
map(company) : 시스포유
```

getMappedProperty(Object bean, String name, String key)

Return	String
Parameters	Bean, 맵 속성 이름, 인덱스
설명	빈의 해당 맵의 지정된 인덱스값을 리턴

- 멤버 필드

```
Public Map<String, String> map = new HashMap<String, String>();
  map.put("company", "시스포유");
  map.put("name", "김진아");
```

```
GetMappedProperty2 bean = new GetMappedProperty2();

String value = BeanUtils.getMappedProperty(bean, "map", "name");

System.out.println("map(name) : " + value);
```

- 실행결과

```
map(name) : 김진아
```

getNestedProperty(Object bean, String name)

Return	String
Parameters	Bean, 속성 이름
설명	해당 빈의 중첩된 속성에 접근할 때 사용

* 소스 별첨

getProperty(Object bean, String name)

Return	String
Parameters	Bean , 속성 이름
설명	해당 빈의 속성이름에 해당하는 값 리턴

- 멤버 필드

```
int i = 1;
```

```
GetProperty bean = new GetProperty();

String value = BeanUtils.getProperty(bean, "i");

System.out.println("GetProperty.i  : " + value);
```

- 실행결과

```
GetProperty.i  : 1
```

populate(Object bean, Map properties)

Return	void
Parameters	Bean, 속성 map
설명	맵의 지정된 이름에 따라 해당 빈의 속성을 채움

- 멤버 필드

```
public String company;
public String name;
```

```
Map<String, String> properties = new HashMap<String, String>();
properties.put("company", "시스포유");
properties.put("name", "김진아");

Populate bean = new Populate();
System.out.println("populate 하기 전 bean : " + bean);

BeanUtils.populate(bean, properties);
System.out.println("populate 한 후  bean : " + bean);
```

- 실행결과

```
populate 하기 전 bean : Populate [company=null,       name=null    ]
populate 한 후    bean : Populate [company=시스포유, name=김진아]
```

setProperty(Object bean, String name, Object value)

Return	void
Parameters	Bean, 속성 이름, 속성이름의 값
설명	해당 빈의 속성값에 값을 설정

- 멤버 필드

```
public String company;
public String name;
```

```
SetProperty bean = new SetProperty();
System.out.println("set 하기 전 bean : "+ bean);

BeanUtils.setProperty(bean, "name", "김진아");

System.out.println("set 한 후    bean : "+ bean);
```

- 실행결과

```
set 하기 전 bean : SetProperty [company=null, name=null    ]
set 한 후    bean : SetProperty [company=null, name=김진아]
```

IV. Q&A

감사합니다.