链滴

日常练习的代码

作者：xiaowangtongxue
原文链接：https://ld246.com/article/1628511522656
来源网站：链滴

关于注解的练习

```
package wzt;

import java.lang.annotation.*;

/**
 * 关于注解的练习
 * Retention(生命周期)有三种情况
 *          源代码中存在    RetentionPolicy.SOURCE
 *          字节码中存在    RetentionPolicy.CLASS
 *          运行时存在      RetentionPolicy.RUNTIME
 * Target(作用域)
 *          包 ElementType.PACKAGE
 *          类
 *          构造器 ElementType.CONSTRUCTOR
 *          方法  ElementType.METHOD
 *          参数  ElementType.PARAMETER
 *          成员变量
 *          局部变量    ElementType.LOCAL_VARIABLE
 *
 * Documented  生成文档
 * Inherited  继承   在父类使用该注解  子类自动使用
 *
 * @author wzt
 * @2021/5/10 -下午12:54
 */
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.PACKAGE,ElementType.CONSTRUCTOR,ElementType.METHOD,Elemen
Type.PARAMETER,ElementType.LOCAL_VARIABLE})
@Documented
```

```
@Inherited
public @interface MyAnnotation {
    String value() default "wzt";
}
```

Arrays.sort 对于整形数组的降序排序

```
package wzt;

import org.junit.Test;

import java.util.Arrays;
import java.util.Collections;
import java.util.Comparator;

/**
 * 对于整形数组的逆序排序 都要使用包装类
 * @author wzt
 * @2021/5/10 -下午12:43
 */
public class Test4 {
    @Test
    public void test1(){
        //使用包装类的方式 在用定制排序
        int[] a = {1,3,5,2,8,6,0};

        Integer[] b = new Integer[a.length];
        for (int i = 0; i < a.length; i++) {
            b[i] = a[i];
        }
        Arrays.sort(b, new Comparator<Integer>() {
            @Override
            public int compare(Integer o1, Integer o2) {
                return -o1.compareTo(o2);
            }
        });
        for (Integer integer : b) {
            System.out.print(integer + "   ");
        }
    }
    @Test
    public void test2(){
        //使用包装类 使用集合的工具
        int[] a = {1,3,5,2,8,6,0};
        Integer[] b = new Integer[a.length];
        for (int i = 0; i < a.length; i++) {
            b[i] = a[i];
        }
        Arrays.sort( b, Collections.reverseOrder());
        for (Integer integer : b) {
            System.out.print(integer + "   ");
        }
    }
```

```
}
```

关于枚举的练习

```
package wzt;

/**  1.私有的属性
 *   2.有参构造器
 *   3.getter方法
 *   4.实例化对象
 * @author wzt
 * @2021/5/10 -下午1:05
 */
public enum MyEnum {
    s("春天","春天描述"),
    x("夏天","夏天描述"),
    q("秋天","秋天描述"),
    d("冬天","冬天描述");

    private final String seasonName;
    private final String seasonDesc;

    MyEnum(String seasonName, String seasonDesc) {
        this.seasonName = seasonName;
        this.seasonDesc = seasonDesc;
    }

    public String getSeasonName() {
        return seasonName;
    }

    public String getSeasonDesc() {
        return seasonDesc;
    }
}



package wzt;

import java.util.Arrays;

/**
 * @author wzt
 * @2021/5/10 -下午1:09
 */
public class EnumTest {

    public static void main(String[] args) {
        //values() 获取枚举中所有的对象名字 返回的是一个数组 数组的遍历Arrays.toString
        //valuesOf("对象名字")  返回的是对象的名字  可以通过get+属性名 获取属性的内容
        System.out.println(Arrays.toString(MyEnum.values()));
```

```
        System.out.println(MyEnum.valueOf("d").getSeasonName());
        System.out.println(MyEnum.valueOf("d").getSeasonDesc());
    }
}
```

关于自然排序的练习

```
package wzt;

import java.util.Objects;

/**
 * @author wzt
 * @2021/5/10 -下午1:15
 */
public class Person implements Comparable<Person> {
    private String name;
    private int age;

    public Person() {
    }

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof Person)) return false;

        Person person = (Person) o;

        if (getAge() != person.getAge()) return false;
        return getName() != null ? getName().equals(person.getName()) : person.getName() ==
ull;
```

```
    }

    @Override
    public int hashCode() {
        int result = getName() != null ? getName().hashCode() : 0;
        result = 31 * result + getAge();
        return result;
    }

    @Override
    public String toString() {
        return "Person{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }

    @Override
    public int compareTo(Person o) {
        int res = this.getName().compareTo(o.getName());
        if (res != 0) {
            return res;
        }else {
            return Integer.compare(this.getAge(),o.getAge());
        }
    }
}
```

关于定制排序的练习

```
@Test
    public void test3() {
        TreeSet<Person> people1 = new TreeSet<>(new Comparator<Person>() {
            @Override
            public int compare(Person o1, Person o2) {
                return o1.getName().compareTo(o2.getName());
            }
        });


        people1.add(new Person("b", 20));
        people1.add(new Person("a", 30));
        people1.add(new Person("r", 40));
        people1.add(new Person("c", 60));
        people1.add(new Person("c", 50));
        people1.add(new Person("f", 60));

        for (Person person : people1) {
            System.out.println(person);
        }
    }
```

lambda定制排序

```
TreeSet<Person> people1 = new TreeSet<>((o1,o2) -> o1.getName().compareTo(o2.getNam
()));

TreeSet<Person> people1 = new TreeSet<>(Comparator.comparing(Person::getName));
```

子类重写父类的方法 并不改变父类的方法

```
package com.atguigu.jprofiler;

/**
 * @author wzt
 * @2021/5/19 -下午7:13
 */
public class ClassTest {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.eat();

        Animal animal = new Animal();
        animal.eat();
    }
}
class Animal{
    public void eat(){
        System.out.println("父类...");
    }
}

class Dog extends Animal{
    @Override
    public void eat() {
        System.out.println("子类...");
    }
}
```

assert关键字的练习

```
package wzt;

public class AssertTest {
    public static void main(String[] args) {
        //开启断言 -ea
        System.out.println("断言之前");
        assert 1==2;
        System.out.println("断言之后");
    }
}
```

动态代理

```
package wzt;

import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Method;
import java.lang.reflect.Proxy;

interface Factory{
    void show();
}
class Proxy1 implements Factory{

    @Override
    public void show() {
        // TODO Auto-generated method stub
        System.out.println("被代理类..");
    }

}
class MyMethod{
    public void beforeMethod() {
        System.out.println("方法执行前...");
    }
    public void afterMethod() {
        System.out.println("方法执行后...");
    }

}
class MyInvokeHandle implements InvocationHandler{
    private Object object;

    public MyInvokeHandle(Object object) {
        super();
        this.object = object;
    }

    @Override
    public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
        // TODO Auto-generated method stub
        new MyMethod().beforeMethod();
        Object value = method.invoke(object, args);
        new MyMethod().afterMethod();
        return value;
    }

}
class ProxyFactory{

    public Object getProxy(Object object) {
        MyInvokeHandle mHandle = new MyInvokeHandle(object);
        return Proxy.newProxyInstance(object.getClass().getClassLoader(), object.getClass().getIn
erfaces(), mHandle);
    }
}
public class DynamicProxyTest {
```

```
    public static void main(String[] args) {
        Proxy1 proxy1 = new Proxy1();
        Factory proxy = (Factory)new ProxyFactory().getProxy(proxy1);

        proxy.show();


    }

}
```

通过反射创建对象+调用方法

```
package wzt;

import java.lang.reflect.Constructor;
import java.lang.reflect.Method;

import pojo.Person;

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

        Class<Person> classPerson = Person.class;
        //不能使用包装类  构造器里是什么类型的就传什么类型了 否则找不到该构造器
        Constructor<Person>  personC =classPerson.getConstructor(String.class,int.class);

        Person person = (Person) personC.newInstance("11",20);

        Method per =classPerson.getMethod("show", null);

        per.invoke(person, null);
        System.out.println(person.getName());
        System.out.println(person.getAge());
    }
}
```

Cglib代理

```
package com.wangzhitao.test;

/**
 * @author wzt
 * @date 2021/7/26 -上午11:52
 */
public class Teacher {
    public void teach(){
        System.out.println("被代理的对象");
    }
    public String say(){
        return "王志涛";
    }
```

```
}
```

```
package com.wangzhitao.test;

import net.sf.cglib.proxy.Enhancer;
import net.sf.cglib.proxy.MethodInterceptor;
import net.sf.cglib.proxy.MethodProxy;

import java.lang.reflect.Method;

/**
 * @author wzt
 * @date 2021/7/26 -上午11:52
 */
public class ProxyFactory implements MethodInterceptor {
    private Teacher teacher;

    public ProxyFactory(Teacher teacher) {
        this.teacher = teacher;
    }
    public Object getProxy(){
        //1、新建工具类
        Enhancer enhancer = new Enhancer();
        //2、设置父类
        enhancer.setSuperclass(teacher.getClass());
        //3、设置回调函数
        enhancer.setCallback(this);
        //4、创建
        return enhancer.create();
    }

    @Override
    public Object intercept(Object o, Method method, Object[] args, MethodProxy methodPro
y) throws Throwable {
        System.out.println("开始代理");
        Object invoke = method.invoke(teacher, args);
        System.out.println("开始结束");
        return invoke;
    }

    public static void main(String[] args) {
        Teacher teacher = new Teacher();
        ProxyFactory proxyFactory = new ProxyFactory(teacher);
        Teacher teacherProxy = (Teacher) proxyFactory.getProxy();
        System.out.println(teacherProxy.say());
    }
}
```

```
        <dependency>
            <groupId>cglib</groupId>
            <artifactId>cglib</artifactId>
            <version>3.3.0</version>
        </dependency>
```