包装类

基础数据类型的包装类

`.parse`

如果想要将一个字符串转变成相应的数据类型，可以使用每一个包装类`TYPE`的`.parseTYPE`方法，比如下面：

```
String str="1234";
System.out.println(Integer.parseInt(str));
```

一些常量

对于`Integer`和`Double`这种类，可以通过一些`final`的量了解其数据范围之类的信息

以`Integer`为例：

```
Integer.MAX_VALUE; // 最大值，一般为 0x7fffffff
Integer.MIN_VALUE; // 最小值
Integer.SIZE; // 大小
```

包装类的优化

Java中所有的模板都必须使用包装类，比如`List<Integer>`

IDE会自动帮助完成一些初始化功能，并且也不需要像一个正常的类一样调用方法，比如下面的代码可以正常运行：

```
Integer i=1;
for(;i<12;++i){
    System.out.println(i);
}
```

不需要`compareTo`，不需要`new`，也可以直接使用`++`

但是在方法调用的时候还是传值

```java
public static void fun(Integer i){
    i++;
}
// in method main
Integer i=10;
fun(i);
System.out.println(i); // 10, fun的Integer i 接收的是值不是指针/引用
```

数组的包装类

数组也有对应的包装类，也就是`Arrays`

`{}`初始化

我们可以使用`{}`初始化

```java
int [] vecr={-1,0,1,0};int []vecc={0,1,0,-1};
```

这就是经典的初始化方法

`toString()`

看下数组的打印

如果我们直接打印一个数组，会发现输出的是它的地址

```java
int [] var={1,2,3,4,5,6,7,8,9,10}; // 注意这是一种初始化数组的方法
System.out.println(var);
```

这个输出是`[I@2752f6e2`，是一个地址

如果想要方便地、和其他类统一的`toString()`方式，需要下面的写法

```java
int [] var={1,2,3,4,5,6,7,8,9,10};
System.out.println(Arrays.toString(var));
```

这里调用的是`Arrays`里面写好的函数：

```java
public static String toString(int[] a) {
    if (a == null)
        return "null";
    int iMax = a.length - 1;
    if (iMax == -1)
        return "[]";
```

```
    StringBuilder b = new StringBuilder();
    b.append('[');
    for (int i = 0; ; i++) {
        b.append(a[i]);
        if (i == iMax)
            return b.append(']').toString();
        b.append(", ");
    }
}
```

也就是Java对于基础类型都做了一个这种方法

那么对于一般的类呢？也可以，见下：

```
class Oneclass{
    int x;

    @Override
    public String toString() {
        return "Oneclass{" +
                "x=" + x +
                '}';
    }

    public Oneclass(){}
}
Oneclass [] varoc=new Oneclass[10];
for(int i=0;i<10;++i){
    varoc[i]=new Oneclass();
}
System.out.println(Arrays.toString(varoc));
```

这里调用的是下面这个重载：

```
public static String toString(Object[] a) {
    if (a == null)
        return "null";

    int iMax = a.length - 1;
    if (iMax == -1)
        return "[]";

    StringBuilder b = new StringBuilder();
    b.append('[');
    for (int i = 0; ; i++) {
        b.append(String.valueOf(a[i])); // 这里最后会调用的是(a[i] == null) ? "null" :
a[i].toString();
        if (i == iMax)
```

```
            return b.append(']').toString();
        b.append(", ");
    }
}
```

所以一个二维数组的`toString()`是下面这样的：

```
int n=3;
int [][]a=new int[n][n];
for(int r=0;r<n;++r){
    a[r]=new int[n]; // 记得二维数组这里要new
    for(int c=0;c<n;++c){
        a[r][c]=r*10+c;
    }
}
System.out.println(Arrays.toString(a));
```

输出：

```
[[I@2752f6e2, [I@e580929, [I@1cd072a9]
```

`.equals()`

我们知道，由于Java并没有运算符重载，所以对于两个Java的对象，`==`表示的是两个地址是不是一样。如果想要判断是不是相等，需要覆写`.equals()`方法。对于数组对象`var`，可以使用`.equals()`来比较，但是IDEA不喜欢这样，它的建议如下：

对于基础类型数组，可以直接调用`equals()`来比较两个数组是不是一样，以`int`为例：

```
int [] a1={1,2};int [] a2={1,3};
System.out.println(Arrays.equals(a1, a2));
```

对于`Object`类，也可以使用相应的方法，下面是对于`Object`的实现

```
public static boolean equals(Object[] a, Object[] a2) {
    if (a==a2)
        return true;
    if (a==null || a2==null)
        return false;

    int length = a.length;
```

```
    if (a2.length != length)
        return false;

    for (int i=0; i<length; i++) {
        if (!Objects.equals(a[i], a2[i])) // 必须实现equals
            return false;
    }
    return true;
}
```

.fill()

直接放代码：

```
public static void fill(int[] a, int val) {
        for (int i = 0, len = a.length; i < len; i++)
            a[i] = val;
    }
public static void fill(int[] a, int fromIndex, int toIndex, int val) {
    rangeCheck(a.length, fromIndex, toIndex);
    for (int i = fromIndex; i < toIndex; i++)
        a[i] = val;
}
public static void fill(Object[] a, Object val) {
    for (int i = 0, len = a.length; i < len; i++)
        a[i] = val;
}
public static void fill(Object[] a, int fromIndex, int toIndex, Object val) {
    rangeCheck(a.length, fromIndex, toIndex);
    for (int i = fromIndex; i < toIndex; i++)
        a[i] = val;
}
```

.copyOf()

直接上源码：

```
public static <T> T[] copyOf(T[] original, int newLength) {
    return (T[]) copyOf(original, newLength, original.getClass());
}
public static <T,U> T[] copyOf(U[] original, int newLength, Class<? extends T[]>
newType) {
    @SuppressWarnings("unchecked")
    T[] copy = ((Object)newType == (Object)Object[].class)
        ? (T[]) new Object[newLength]
        : (T[]) Array.newInstance(newType.getComponentType(), newLength);
    System.arraycopy(original, 0, copy, 0,
                     Math.min(original.length, newLength));
    return copy;
}
```

就是返回一个copy，长度为输入的`newLength`和被拷贝数组长度的较小值

包装类参与的类型转换

先声明：下面讲的东西我觉得很tm没用，但是老师考了，所以我也记录下：

```
public class TestWrapper {
    public static void main(String[] args)  {
        short a1=6;
        new Main().go(a1);
        new Main().go(new Integer(7));
    }
    static class Main{
        void go(Short x){System.out.print("S ");}
        void go(Long x){System.out.print("L ");}
        void go(int x){System.out.print("i ");}
        void go(Number n){System.out.print("N ");}
    }
}
```

这题的输出结果是：`i N`

经过测试，对于以下代码：

```
public class TestWrapper {
    public static void main(String[] args)  {
        short s=6;
        Short S=new Short((short)6);
        new Main().go(s);
//      new Main().go(S);
    }
    static class Main{
```

```
        void go(short s){ System.out.print("s ");}
        void go(int i){System.out.print("i ");}
        void go(Short S){System.out.print("S ");}
        void go(Number N){System.out.print("N");}
        void go(Long L){System.out.print("L ");}
        void go(long l){System.out.println("l ");}
    }
}
```

通过这个可以使用`new Main().go(s);`测试执行的是什么方法

经过测试：

`Short`的执行优先级是：`Short`->`Number`->`short`->`int`->`long`，注意不能传到`Integer`和`Long`

`short`的执行优先级是：`short`->`int`->`long`->`Short`->`Number`，注意还是不能传到`Integer`和`Long`

`Class`类和反射

`Class`类简介

`Class`类对象就相当于B超的探头，将一个类的方法、变量、接口、类名、类修饰符等信息告诉运行的程序，

`Class`相关方法

类名`.class`

对于一个类`Tes`，可以使用`Tes.class`的方式获得`Tes`的类型信息（一个`class`类）

《坑！》