JAVA泛型、序列化、复制 （非重点）

JAVA泛型

- 泛型提供了编译时类型安全检测机制，该机制允许程序员在编译时检测到非法的类型。泛型的本质是参数化类型，也就是说所操作的数据类型被指定为一个参数。比如我们要写一个排序方法，能够对整型数组、字符串数组甚至其他任何类型的数组进行排序，我们就可以使用 Java 泛型。

泛型方法

- 你可以写一个泛型方法，该方法在调用时可以接收不同类型的参数。根据传递给泛型方法的参数类型，编译器适当地处理每一个方法调用。

```
// 泛型方法 printArray
 public static < E > void printArray( E[] inputArray )
 {
    for ( E element : inputArray ){
        System.out.printf( "%s ", element );
    }
 }
```

- <? extends T>表示该通配符所代表的类型是 T 类型的子类。
- <? super T>表示该通配符所代表的类型是 T 类型的父类。

泛型类

- 泛型类的声明和非泛型类的声明类似，除了在类名后面添加了类型参数声明部分。和泛型方法一样，泛型类的类型参数声明部分也包含一个或多个类型参数，参数间用逗号隔开。一个泛型参数，也被称为一个类型变量，是用于指定一个泛型类型名称的标识符。因为他们接受一个或多个参数，这些类被称为参数化的类或参数化的类型。

```
public class Box<T> {
    private T t;
    public void add(T t) {
        this.t = t;
    }

    public T get() {
        return t;
    }
 }
```

类型通配符?

- 类型通配符一般是使用？代替具体的类型参数。例如 List<?> 在逻辑上是 List,List 等所有 List<具体类型实参>的父类。

类型擦除

- Java 中的泛型基本上都是在编译器这个层次来实现的。在生成的 Java 字节代码中是不包含泛型中的类型信息的。使用泛型的时候加上的类型参数，会被编译器在编译的时候去掉。这个过程就称为

类型擦除。如在代码中定义的 List和 List等类型，在编译之后 都会变成 List。JVM 看到的只是 List，而由泛型附加的类型信息对 JVM 来说是不可见的。类型擦除的基本过程也比较简单，首先是找到用来替换类型参数的具体类。这个具体类一般是 Object。如果指定了类型参数的上界的话，则使用这个上界。把代码中的类型参数都替换成具体的类。

JAVA序列化

- Java序列化是指把Java对象转换为字节序列的过程，而Java反序列化是指把字节序列恢复为Java对象的过程

Serializable 实现序列化

- 在 Java 中，只要一个类实现了 java.io.Serializable 接口，那么它就可以被序列化。

Transient 关键字阻止该变量被序列化到文件中

- 在变量声明前加上 Transient 关键字，可以阻止该变量被序列化到文件中，在被反序列化后，transient 变量的值被设为初始值，如 int 型的是 0，对象型的是 null。
- 服务器端给客户端发送序列化对象数据，对象中有一些数据是敏感的，比如密码字符串 等，希望对该密码字段在序列化时，进行加密，而客户端如果拥有解密的密钥，只有在 客户端进行反序列化时，才可以对密码进行读取，这样可以一定程度保证序列化对象的 数据安全。

JAVA 复制

- 将一个对象的引用复制给另外一个对象，一共有三种方式。第一种方式是直接赋值，第二种方式是浅拷贝，第三种是深拷贝。所以大家知道了哈，这三种概念实际上都是为了拷贝对象。

直接赋值复制

- 直接赋值。在 Java 中，A a1 = a2，我们需要理解的是这实际上复制的是引用，也就是说 a1 和 a2 指向的是同一个对象。因此，当 a1 变化的时候，a2 里面的成员变量也会跟着变化。

浅复制（复制引用但不复制引用的对象）

- 创建一个新对象，然后将当前对象的非静态字段复制到该新对象，如果字段是值类型的，那么对该字段执行复制；如果该字段是引用类型的话，则复制引用但不复制引用的对象。因此，原始对象及其副本引用同一个对象。

```
class Resume implements Cloneable{
    public Object clone() {
        try {
            return (Resume)super.clone();
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
}
```

深复制（复制对象和其应用对象）

- 深拷贝不仅复制对象本身，而且复制对象包含的引用指向的所有对象。

```
class Student implements Cloneable {
    String name;
```

```
    int age;
    Professor p;
    Student(String name, int age, Professor p) {
        this.name = name;
        this.age = age;
        this.p = p;
    }

    public Object clone() {
        Student o = null;
        try {
            o = (Student) super.clone();
        } catch (CloneNotSupportedException e) {
            System.out.println(e.toString());
        }
    o.p = (Professor) p.clone();
    return o;
    }
}
```

序列化（深 clone 一中实现）

- 在 Java 语言里深复制一个对象，常常可以先使对象实现 Serializable 接口，然后把对象（实际上只是对象的一个拷贝）写到一个流里，再从流里读出来，便可以重建对象。