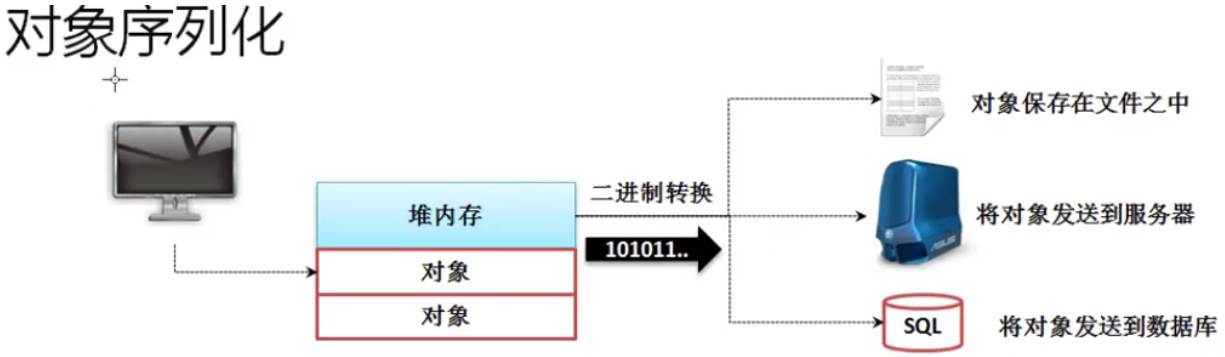
博客： https://www.cnblogs.com/HOsystem/p/14116443.html

2、具体内容

几乎只要是Java开发就一定会存在有序列化的概念，而正是因为序列化的概念逐步发展，慢慢也有了更多的序列化标准。

■序列化简介

所谓的对象序列化指的是将内存中保存的对象以二进制数据流的形式进行处理，可以实现对象的保存或者是网络传输。



然而并不是所有的对象都可以被序列化，在Java里面有一个强制性的要求：如果要序列化的对象，那额对象所在的类一定要实现java.io.Serializable父接口，作为序列化的标记，这个接口并没有任何的方法，因为它描述的是一种类的能力。

范例：定义一个可以被序列化的类

```
@SuppressWarnings("serial")
class Person implements Serializable { // Person类可以被序列化
	private String name ;
	private int age ;
	public Person(String name,int age) {
		this.name = name ;
		this.age = age ;
	}
	// setter、getter略
```

```
    @Override
    public String toString() {
        return "姓名：" + this.name + "、年龄：" + this.age ;
    }
}
```

此时Person类产生的每一个对象都可以实现二进制的数据传输，属于可以被序列化的程序类。

■序列化与反序列化

有了序列化的支持类之后如果要想实现序列化与反序列化的操作，则就可以利用以下两个类完成：

类名称	序列化：ObjectOutputStream	反序列化：ObjectInputStream
类定义	public class ObjectOutputStream extends OutputStream implements ObjectOutput, ObjectStreamConstants	public class ObjectInputStream extends InputStream implements ObjectInput, ObjectStreamConstants
构造方法	public ObjectOutputStream (OutputStream out) throws IOException	public ObjectInputStream(InputStream in) throws IOException
操作方法	public final void writeObject(Object obj) throws IOException	public final Object readObject() throws IOException,ClassNotFoundException

范例：实现序列化与反序列化

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;
public class JavaAPIDemo {
    private static final File SAVE_FILE = new File("D:" + File.separator + "mldn.person") ;
    public static void main(String[] args) throws Exception {
        // saveObject(new Person("小喷嘘",78)) ;
        System.out.println(loadObject());
    }
    public static void saveObject(Object obj) throws Exception {
        ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(SAVE_FILE)) ;
        oos.writeObject(obj); // 序列化
        oos.close();
    }
    public static Object loadObject() throws Exception {
        ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(SAVE_FILE)) ;
        Object obj = ois.readObject() ; // 反序列化
        ois.close();
```

```
            return obj ;
        }
}
@SuppressWarnings("serial")
class Person implements Serializable {   // Person类可以被序列化
        private String name ;
        private int age ;
        public Person(String name,int age) {
            this.name = name ;
            this.age = age ;
        }
        // setter、getter略
        @Override
        public String toString() {
            return "姓名：" + this.name + "、年龄：" + this.age ;
        }
}
```

在Java中的对象序列化与反序列化必须使用内部提供的对象操作流，因为这里面牵扯到二进制数据的格式，所以不能够自定义处理，另外如果要想实现一组对象的序列化，则可以使用对象数组完成。

在很多的实际项目开发过程之中，开发者很少能够见到ObjectOutputStream、ObjectInputStrea m类的直接操作，因为会有一些程序容器帮助开发者自动实现。

■transient关键字

默认情况下当执行了对象序列化的时候会将类中的全部属性的内容进行全部的序列化，但是很多情况下有一些属性可能并不需要进行序列化的处理，这个时候就可以在属性定义上使用transient关键字来完成了。

```
private transient String name ;
```

在进行序列化处理的时候name属性的内容是不会被保存下来的，换言之，读取的数据name将使其对应数据类型的默认值“null”。如果假设类之中有一些是需要计算保存的属性内容往往是不需要被序列化的，这个时候就可以使用transient，但是在实际的开发之中大部分需要被序列化的类往往都是简单java类，所以这一关键字的出现频率并不高。