数据以二进制序列的形式在网络上传送。比如，我们可以通过http协议发送字符串信息;我们也可以在网络上直接发送Java对象。发送方需要把这个Java对象转换为字节序列，才能在网络上传送;接收方则需要把字节序列再恢复为Java对象才能正常读取。

把Java对象转换为字节序列的过程称为对象的序列化。把字节序列恢复为Java对象的过程称为对象的反序列化。

对象序列化的作用：

1. 持久化： 把对象的字节序列永久地保存到硬盘上，通常存放在一个文件中，比如：休眠的实现。以后服务器session管理，hibernate将对象持久化实现。

2. 网络通信：在网络上传送对象的字节序列。比如：服务器之间的数据通信、对象传递。

序列化涉及的类和接口

ObjectOutputStream代表对象输出流，它的writeObject(Object obj)方法可对参数指定的obj对象进行序列化，把得到的字节序列写到一个目标输出流中。

ObjectInputStream代表对象输入流，它的readObject()方法从一个源输入流中读取字节序列，再把它们反序列化为一个对象，并将其返回。

只有实现了Serializable接口的类的对象才能被序列化。Serializable接口是一个空接口，只起到标记作用。

实例进行序列化和反序列化

```
import  java.io.FileInputStream;
import  java.io.FileOutputStream;
import  java.io.IOException;
import  java.io.ObjectInputStream;
import  java.io.ObjectOutputStream;
import  java.io.Serializable;

//Person类实现Serializable接口后，Person对象才能被序列化
class  Person implements  Serializable {
    // 添加序列化ID，它决定着是否能够成功反序列化！
```

```
    private static final long serialVersionUID = 1L;
    int age;
    boolean isMan;
    String name;

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

    @Override
    public String toString() {
        return "Person [age=" + age + ", isMan=" + isMan + ", name=" + name + "]";
    }
}

public class TestSerializable {
    public static void main(String[] args) {
        FileOutputStream fos = null;
        ObjectOutputStream oos = null;
        ObjectInputStream ois = null;
        FileInputStream fis = null;
        try {
            // 通过ObjectOutputStream将Person对象的数据写入到文件中，即序列化。
            Person person = new Person(18, true, "高淇");
            // 序列化
            fos = new FileOutputStream("d:/c.txt");
            oos = new ObjectOutputStream(fos);
            oos.writeObject(person);
            oos.flush();
            // 反序列化
            fis = new FileInputStream("d:/c.txt");
            // 通过ObjectInputStream将文件中二进制数据反序列化成Person对象：
            ois = new ObjectInputStream(fis);
            Person p = (Person) ois.readObject();
            System.out.println(p);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
```

```
            if (oos != null) {
                try {
                    oos.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            if (fos != null) {
                try {
                    fos.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            if (ois != null) {
                try {
                    ois.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            if (fis != null) {
                try {
                    fis.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

1. static属性不参与序列化。

2. 对象中的某些属性如果不想被序列化，不能使用static，而是使用transient修饰。

3. 为了防止读和写的序列化ID不一致，一般指定一个固定的序列化ID。