数据结构之线性表

数据结构中线性表是零个或多个的数据的有限序列，每个元素除了头和尾之外都有自己唯一的前驱和后继
线性表分为顺序存储结构和链式存储结构

1. 顺序存储结构

在Java的ArrayList中用的就是顺序存储结构：数组。今天阅读了一下ArrayList的源码，发现了很多问题，在此记录一下，省的忘记。

首先是在源码中的transient这个关键字，作用是用于修饰类中的变量，防止被序列化.

2. 链式存储结构

链式存储结构中，节点处除了存储数据元素之外还需要存储指向下一个节点的指针，即后一个节点的位置。这是单向链表。如果在存储指向前一个节点的地址，这是双向链表

在ArrayList中存储元素的变量用transient修饰，所以理论上ArrayList不能被序列化。但是在ArrayList中却可以，因为在ArrayList中重写了writeObject 这个方法，所以可以转化为文件流，可以被序列化。 网上说这样设计的目的是因为在ArrayList中的存放元素的是一个素组，这个数组的容量大小基本上都会比实际的元素的个数要大，为了避免序列化没有元素的数组而重写。 实际上在我的测试中确实也是如此，阅读ObjectOutInputStream的源码会发现，如果你重写了writeObject 和readObjec 这个两个方法，在实际序列化的时候，会利用反射最终调用到你重写的writeObject和readObject 来序列化。

但是 我就有一点不明白了，既然重写writeObject 和readObject 就可以自己序列化，那么arrayList中为什么还是需要用到transient呢？

下面是我的实验： package com.zyd.serializable.test;

import java.io.IOException; import java.io.Serializable;

public class AnimalSerializable implements Serializable{

```
  /**
   *
   */
  private static final long serialVersionUID = -7461498806905104482L;

  public AnimalSerializable() {

  }

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

  private transient int age ;
  private transient String name;

    private void writeObject(java.io.ObjectOutputStream s){
        try {
            s.defaultWriteObject();
          //s.writeObject(name);
          //s.writeObject(age);
          System.out.println("重写了writeObject");
      } catch (IOException e) {
          // TODO Auto-generated catch block
          e.printStackTrace();
      }
    }
    private void readObject(java.io.ObjectInputStream s){
        try {
            s.defaultReadObject();
          //name = (String) s.readObject();
          //age = (int) s.readObject();
```

```
            System.out.println("重写了 readObject");
        } catch (ClassNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
      }


    public int getAge() {
        return age;
    }

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

    public String getName() {
        return name;
    }

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

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


}
```

main 方法：

```
    public class Test {
        public static void main(String[] args) {
            AnimalSerializable animal = new AnimalSerializable(10, "xiaohong");

        File file = new File("D:/test.txt");
        if(!file.exists()){
            try {
                file.createNewFile();
            } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }

        try {
            ObjectOutputStream output = new ObjectOutputStream(new
    FileOutputStream(file));
            output.writeObject(animal);
            output.close();

            ObjectInputStream input = new ObjectInputStream(new
    FileInputStream(file));
            AnimalSerializable anima = (AnimalSerializable) input.readObject();
            input.close();

            System.out.println(anima.toString());
        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (Exception e) {
```

```
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

    }

}
```

输出结果：

输出的结果： 重写了writeObject 重写了 readObject AnimalSerializable [age=0, name=null]

如果我确定writeObject和readObject中的注释，那么结果是： 重写了writeObject 重写了 readObject AnimalSerializable [age=10, name=xiaohong]

这说明s.defaultReadObject() 和s.defaultWriteObject()可以序列化那些没有被transient的变量，只有加上transient的变量才需要重写！ 此博文写的不错： 深入分析Java的序列化与反序列化