十、集合类框架

- 概念：对象通过组合构成更高级别的对象。集合类框架就是用来描述这种组合关系，它由一系列的集合类和接口所组成，主要存放到java.util包中。

Collection接口类型

- Collection是集合类的基本接口，它用来说明作为一个集合类应有的结构特征属性和带有共性的操作方法。
- 子接口主要包含 **List** 和 **Set**。
- List接口规定实现其的集合类元素具有可以控制的顺序，但并没有定义或限制什么顺序
- Set不能包含重复的元素

常用集合类特性对比

集合类	主要功能	实现接口	继承关系	安全性
Vector	可自动增加容量来容纳所需的对象元素，可以通过get方法得到对象元素，也可以通过迭代器遍历对象	Collection，List	从AbstractList派生而来	线程安全
Stack	增加了栈的方法	Collection，List	从Vector派生而来	线程安全
ArrayList	规模可变并且能像链表	Collection，List	从AbstractList派生而来	线程不安全
LinkedList	实现一个链表，由这个类定义的链表也可以像栈或队列一样使用	Collection，List	AbstractList派生而来	线程不安全
HashSet	虽然定义成无序，但可在固定时间内完成集合对象元素的储存和检索，也就是说性能不受储存大小的限制	Collection，List	AbstractList派生而来	线程不安全

集合类	主要功能	实现接口	继承关系	安全性
TreeSet	在集合中以升序对对象进行排序，这意味着从一位TreeSet对象获得的迭代器将按升序遍历对象	Collection，List	AbstractList派生而来	线程不安全
HashTable	实现key和value之间的映射，通过不重复的key来确定value，效率较高	Map	从Dictionary派生而来	线程安全
HashMap	实现key和value之间的映射，通过不重复的key来确定value，效率较高	Map	从AbstractMap派生而来	线程不安全
TreeMap	实现按key升序排列的一个映射，即从TreeMap得到的key的Set集合按照升序进行排列	Map	从AbstractMap派生而来	线程不安全

常见集合类比较

1. ArrayList和LinkedList比较

name	**ArrayList**	**LinkedList**
共同点	都是用来存储一组数量可变的对象集合	
不同点	内部显示基于内部数组Object[]随机访问元素时，可以通过索引直接定位，效率较高。插入、删除等操作需移动数组元素，效率较低	内部实现基于一组连接的Node节点。随机访问元素时，必须从表头开始访问，效率较低 插入、删除操作不需移动Node节点，比ArrayList效率高

```
// 将10个数字顺序加入链表当中，然后倒序输出
import java.util.*;
public class IntObjectLink(String args[]) {
    Integer tem;
    try {
        LinkedList lst = new LinkedList();
        for(int i=0;i<10;i++) {
            lst.addFirst(new Integer(i));
        }
        for(int i=0;i<10;i++) {
```

```
            tem = (Integer)lst.removeFirst();
            System.out.println(tem.intValue());
        }
    }
    catch(Exception e) {
        System.out.println("error");
    }
}
```

2. Enumeration接口

接口操作	描述
boolean hasMoreElements()	检索到有下一个对象返回true，否则返回false
Object nextElement()	在hasMoreElements方法为true的条件下，返回一个Object类型的对象引用；如果上一个方法为false，调用此方法将产生NoSuchElementException。

3. Iteration

    - hasNext()
    - next()
    - remove()

4. Lislteraor接口：不仅可以实现向后遍历，还可以实现向前遍历

5. Vector和ArrayList

name	**Vector**	**ArrayList**
共同点	都是用来存储一组数量可变的对象集合，并可通过get方法随机地访问其中的元素。	
不同点	方法是同步的，是线程安全的；当Vector中的元素超过它的初始大小时，vector会将它的容量翻倍	ArrayList的方法不是同步的，但性能很好（线程的同步必然要影响性能）。当ArrayList中的元素超过它的初始大小时，ArrayList只增加50%的大小

Map接口类型

1. 用于将一个键（key）映射到一个值（Value），且不允许有重复的键。

2. Map操作：

    - Map改版
    - Map查询
    - 三种不同的Map视图

3. Hashtable和HashMsp

name	Hashtable	HashMap
共同点	都可以实现多组key和value之间的映射	
不同点	Hashtable是同步的——线程安全的；Hashtable是不允许key和value的值为null	HashMap是异步的——效率很高（为了快速访问）；HashMap允许使用null关键字和null值

4. TreeMap和HashMap

name	HashMap	TreeMap
共同点	都可以实现多组key和value之间的映射	
不同点	如果没有按照关键字顺序提取Map元素的需求，那么HashMsp是更实用的结构	TreeMap在操作上需要比HashMap更多一些开销，这是由于树的结构造成的，它返回排序的关键字

十四、I/O输入输出

流的分类

1. 流从流动方向上看：一般分为输入流（System.in）和输出流（System.out）
2. 从读取类型上分：一般分为字节流（System.in）和字符流（new InputStreamReader(System.in)）
3. 流从发生的源头：分为节点流（直接操作目标设备对应的流，如文件流）和过滤流（继承带有关键字Filter的流）类

FileInputStream和FileOutputStream

- FileInputStream(String name)：以文件路径名字构造一个文件输入流，打开一个与实际文件的连接，用于从该流中读取文件字节流
- FileOutoutStream：以文件路径名字构造一个文件输出流，打开一个与实际文件的连接，用于文件的写字节流操作

```
// 读入文件
FileInputStream rf = new FileInputStream("OpenFile.java");  // 创建文件输入流对象
int n=512,c=0;
byte buffer[] = new byte[n];
while((c=rf.read(buffer,0,n))!=-1) {    // 读取输入流
    System.out.print(new String(buffer,0.c));
}
rt.close();

// 写入文件
int count,n=512;
byte buffer[] = new byte[n];
count = System.in.read(buffer); // 读取标准输入流
FileOutputStream wf = new FileOutputStream("Write1.txt");
```

```
wf.write(buffer,0,count);
wf.close();
```

过滤流

DataInputStream和DataOutputStream,可从字节流中写入，读取Java基本数据类型，不依赖于机器的具体数据类型，方便存储和恢复数据

```
DataOutputStream dos = new DataOutputStream(new BufferedOutputStream(new
FileOutputStream("test.txt")));
dos.writeInt(3);
dos.writeUTF("hello");
dos.close();
DataInputStream dis = new DataInputStream(new BufferedInputStream(new
FileInputStream("test.txt")));
System.out.println(dis.readInt());
```

PrintWriter的使用

可以向该字符流中写入Java基本数据类型，用于包装输出字符类对象

```
PrintWriter out = new PrintWriter("foo.txt");
out.println("hello");   // 写入字符串
out.println(3); // 写入整形
```

流的串行化

- 串行化(Serialization)：又称序列化，将实现了Seriallizable接口的对象转换成一个字节序列，并能够在以后将这个字节序列完全恢复为原来的对象，后者又称反序列化
- 串行化的目的：便于介质存储和网络传输

文件操作

```
// 文件备份
public void copy(File f1,File f2) throws IOException {
    FileInputStream rf = new FileInputStream(f1);   // 创建文件输入流对象
Write1.txt
    FileOutputStream wf = new FileOutputStream(f2); // 创建文件输出流对象
backup\\Write1.txt
    int count,n=512;
    byte buffer[] = new byte[n];
    count = rf.read(buffer,0,n);    // 读取输入流
    while(count!=-1) {
        wf.write(buffer,0,count);   // 写入输出流
        count = rf.read(buffer,0,n);    // 继续从文件中读取数据到缓冲区
```

```
        }
        System.out.println("CopyFile"+f2.getName()+" !");
        rf.close();      // 关闭输入流
        wf.close();      // 关闭输出流
    }
```

十五、Java网络通信

TCP和UDP比较

name	TCP	UDP
通信方式	进行数据传输之前必然要**建立连接**，所以在TCP中多了一个连接建立的时间	每个数据报中都给出了完整的地址信息,因此无需建立发送方和接收方的连接
传输数据量	一旦连接建立起来，双方的socket就可以按统一的格式传输大量的数据	传输数据时有大小限制，每个被传输的数据报必须在64KB之内
传输数据可靠性	TCP是一个可靠的协议，它能确保接收方完全正确地获取发送方所发送的全部数据	UDP是一个不可靠的协议，发送方所发送的数据报并不一定以相同的次序到达接收方，也不能保证接收方一定能收到
各自特点	TCP传输量大，可靠性强。例如远程连接和文件传输都需要不定长度的数据被可靠地传输	UDP操作简单，传输效率高

Socket通讯

- 网络上的两个程序通过**一个双向的通讯连接实现数据的交换**，这个双向链路的一端称为一个Socket。
- 是TCP/IP协议的一个十分流行的编程接口，一个Socket由一个IP地址和一个端口号唯一确定。
- 是低层次网络编程。

创建Socket

```
// 客户端
try {
    // 向本机的4700端口发出客户请求
    Socket socket = new Socket('127.0.0.1',4700);
} catch(IOException e) {
    System.out.println("can not listen to:"+e);
}
// 服务器端
try {
    // 创建一个ServerSocket在端口4700
    server = new ServerSocket(4700)
} catch(IOException e) {
    System.out.println("can not listen to:"+e);
```

```
}
Socket socket = null;
try {
    // 使用accpet()阻塞等待客户请求，有客户请求到来则产生一个Socket对象，并继续执行
    socket = server.accept();
} catch(IOException e) {
    System.out.println("Error"+e);
}
```

Socket通讯的一般过程

1. 创建socket。（bind()没有调用则自动绑定）
2. 打开来凝结到Socket的输入/出流
3. 先启动服务端，再启动客户端，客户端向服务端发送字符串，之后处于阻塞等待状态；服务端收到客户端发送的字符串后显示，并处于向客户端发送字符状态。
4. 关闭Socket（先关闭流）

十六、JDBC

全称Java DataBase Connectivity，它是Java面向对象应用程序访问数据库的接口规范，它的出现使Java应用程序对各种关系数据库的访问方法得到统一。

使用JDBC操作数据库

- 对应的包为java.sql
- 包含以下几个步骤：
    1. 载入JDBC driver。
    2. 得到数据库的connection对象。
    3. 根据连接对象得到Statement进行查询或数据更新。
    4. 如果执行查询则对返回的记录集ResultSet进行遍历操作；如果执行更新则根据成功与否的返回结果进行数据库事务操作。
    5. 操作结束后，依次关闭ResultSet、Statement、Connection执行关闭操作。