第一章：集合

教学内容：

1. JavaBean 的概念、属性、使用
2. 集合的概念
3. Collection 接口
4. List 接口
5. Set 接口
6. Map 接口

一、 JavaBean 的概念、属性、使用

1. JavaBean 的基础

JavaBean 是一种 JAVA 语言写成的可重用组件。为写成 JavaBean，类必须是具体的和公共的，并且具有无参数的构造器。JavaBeans 通过提供符合一致性设计模式（桥模式）的公共方法将内部域暴露称为属性。众所周知，属性名称符合这种模式，其他 Java 类可以通过自省机制发现和操作 JavaBean 属性。

2. JavaBean 的特征

1) 无参的构造函数；

2) 私有的属性；

3) 公共的 set/get 方法；

4) 可序列化(实现 `Serializable` 接口)；

3. JavaBean 的属性：

1) 简单属性：就是它只接受单个值，比如说字符串或者一个数字；

javabean 的示例：

```
public class User {

    private String name;
    private int age;
    private String address;
```

```
    public User() {
        // TODO Auto-generated constructor stub
    }

    public String getName() {
        return name;
    }

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

    public int getAge() {
        return age;
    }

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

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

}
```

4. javaBean 的使用

用户可以使用 JavaBean 将功能、处理、值、数据库访问和其他任何可以用 java 代码创造的对象进行打包，并且其他的开发者可以通过内部的 JSP 页面、Servlet、其他 JavaBean、applet 程序或者应用来使用这些对象。用户可以认为 JavaBean 提供了一种随时随地的复制和粘贴的功能，而不用关心任何改变。

JavaBean 可以有多个构造函数，但其中必须有一个是无参的；

JavaBean 可以有 main()方法，它的作用是用来对 JavaBean 本身进行测试；

可以使用 JavaBean 将功能、业务处理、数据值、数据库访问和其它任何 java 对象进行打包，以提供代码复用当把 JavaBean 应用到 WEB 环境中时，要保证它支持多线程环境 - - 可将 Bean 中的访问数据值的方法设为同步方法 ；

了解一下：pojo、javabean、vo、dto 的区别

POJO（Plain Old Java Object）：普通 java 对象，而不是一个特殊的 java 对象。几个特点：1.不是扩展预定的类；2.不实现预定的接口；3.不包含预定的标注；

javabean：语言的可重写组件；必须有无参的构造函数，私有的属性，公共的 set/get 方法，通常可序列化，简单来理解，当一个 POJO 满足了 javabean 的条件就是一个 javabean；

VO（value object）：主要体现视图对象。将页面的数据封装成一个对象，使用 VO 对象进行数据传输；简单理解当 POJO 用于视图层它就是 VO；

DTO（Data Transfer Object）：数据传输对象，当 POJO 用来传递，传递过程中就是 DTO；

二、 集合的概念

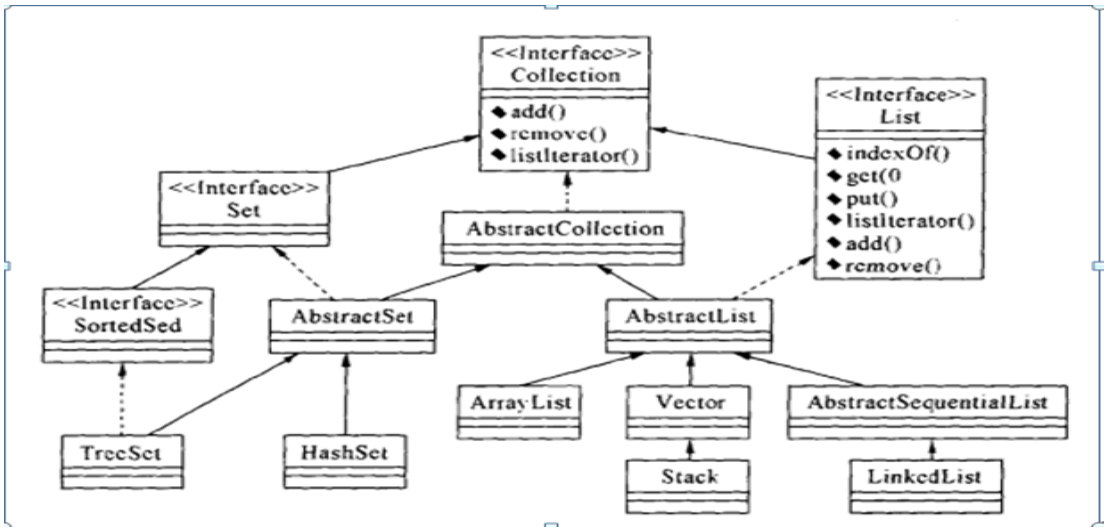
1. 集合的概述

集合用于在类中对数据进行组织。集合是一种容器对象，用于按照一定的规则在其中保存一组对象。

在 J2SE 中，引入了集合框架，它由集合库包构成（集合库包位于 java.util 包中）。该框架定义了许多用于实现集合的接口和抽象类，并且描述了某些机制，比如迭代协议等。

在 Java2 中，有一套设计优良的接口和类组成了 Java 集合框架 Collection，使程序员操作成批的数据或对象元素极为方便。

这些接口和类有很多对抽象数据类型操作的 API，而这是我们常用的且在数据结构中熟知的。例如 Map，Set，List 等。并且 Java 用面向对象的设计对这些数据结构和算法进行了封装，这就极大的减化了程序员编程时的负担。程序员也可以以这个集合框架为基础，定义更高级别的数据抽象，比如栈、队列和线程安全的集合等，从而满足自己的需要。



2. java 的集合框架

Java2 的集合框架，抽其核心，主要有三种：List、Set 和 Map

常用集合类的继承结构如下：
Collection<--List<--Vector
Collection<--List<--ArrayList
Collection<--List<--LinkedList
Collection<--Set<--HashSet
Collection<--Set<--HashSet<--LinkedHashSet
Collection<--Set<--SortedSet<--TreeSet
Map<--SortedMap<--TreeMap
Map<--HashMap 。

三、 Collection 接口

1. Collection 接口有下面两个基本方法：

boolean add(Object obj)

Iterator iterator()

add 方法用于将对象添加给集合。如果添加对象后，该集合确实发生了变化，那么该方法返回 true；如果该集合没有变化，则返回 false。例如，如果你试图将一个对象添加给一个集合，而该集合中已经有该对象了，那么 add 请求将被拒绝，因为该集合拒绝纳入重复的对象

iterator 方法用于返回一个实现了 Iterator 接口的类的对象。 Iterator 类型的对象称为迭代子对象，它专门用于访问集合中的各个元素

2. 具体的集合

Java 库提供了下面 10 个具体的集合类：LinkedList 、ArrayList 、HashSet 、TreeSet 、HashMap 、TreeMap 、Vector 、Stack 、HashTable 、Properties

四、 List 接口

List 是有序的 Collection，使用此接口能够精确的控制每个元素插入的位置。用户能够使用索引（元素在 List 中的位置，类似于数组下标）来访问 List 中的元素，这类似于 Java 的数组。

和下面要提到的 Set 不同，List 允许有相同的元素。 除了具有 Collection 接口必备的 iterator()方法外，List 还提供一个 listIterator()方法，返回一个 ListIterator 接口，和标准的 Iterator 接口相比，ListIterator 多了一些 add()之类的方法，允许添加，删除，设定元素，还能向前或向后遍历。

实现 List 接口的常用类有 LinkedList，ArrayList，Vector 和 Stack

1. ArrayList 类

数组列表 ArrayList 实现了 List 接口。

ArrayList 封装了一个动态再分配的 Object[]数组，可以使用 get 和 set 方法来随机地访问其中的元素。

2. LinkedList 类

链接列表 LinkedList 实现了 List 接口。

链接列表是一个有序集合，将每个对象存放在独立的链接中，每个链接中还存放着序列中下一个链接的索引。在 java 中，所有的链接列表实际上是双重链接的，即每个链接中还存放着对它的前面的链接的索引链接列表适合于处理数据序列中

数据数目不定，且频繁进行插入和删除操作的问题 - - 插入或删除一个元素时，只需要更新其它元素的索引即可，不必移动元素的位置，效率很高。

ListIterator（列表迭代子）接口中定义了许多方法用于访问链接列表 LinkedList，LinkedList 类中有一个 listIterator 方法，可以返回一个 ListIterator 对象:

```
ListIterator  iter=list. listIterator();
Object  oldValue=iter. next();
Iter. set(newValue);
```

链接列表不支持快速随机访问。如果你想查看列表中的第 n 个元素，你必须从头开始查看，然后跳过前面的 n-1 个元素

如果你需要随机访问某个集合的话，请使用数组或者 ArrayList，而不要使用 LinkedList 。

3. Vector 类

Vector 非常类似 ArrayList，但是 Vector 是同步的。由 Vector 创建的 Iterator，虽然和 ArrayList 创建的 Iterator 是同一接口，但是，因为 Vector 是同步的，当一个 Iterator 被创建而且正在被使用，另一个线程改变了 Vector 的状态（例如，添加或删除了一些元素），这时调用 Iterator 的方法时将抛出 ConcurrentModificationException，因此必须捕获该异常。

Vector 是一个"旧的"集合类。它与 ArrayList 类一样，封装了一个动态的 Object[]数组。

二者最大的区别在于：Vector 类的所有方法都是同步方法，而 ArrayList 类的方法都是非同步方法；

第二个区别在于：Vector 类的方法要比 ArrayList 类的方法多，在某些情况下使用起来比 ArrayList 类方便，创建了一个 Vector 对象后，可以往其中随意地插入不同的类的对象，既不需顾及类型也不需预先选定向量的容量，并可方便地进行查找。对于预先不知或不愿预先定义数组大小，并需频繁进行查找、插入和删除工作的情况，可以考虑使用 Vector ，Vector 常用于保存从数据库中读取的记录。

4. ArrayList， Vector， LinkedList 区别

1) ArrayList 底层是数组结构，查询速度比较快，由于每添加或者删除一个元素，该元素后面的索引都需要向后或者向前移动一位，导致 ArrayList 增删元素都是比较慢的。ArrayList 默认初始化的容量大小为 10，当容器的空间不够是以约 50%的增量增加容量；

2) LinkedList 底层是链表结构，由于每次查询元素时都需要从第一个元素开始，导致查询元素比较慢；其增删元素不需要重新设置元素的索引，索引增删元素比较快。

3) Vector 底层和 ArrayList 一样是数组，其区别在于方法都是同步方法，在多线程的情况下只能同时被一个对象调用，所以其增删查元素都是比较慢的。其初始化容器大小是 10，当容器大小不够是会以 100%的增量增加容量。

五、 Set 接口

1、set

- Set 接口是 Collection 接口的子接口。
- Set 类型的集合具有以下特点：
  - 不允许包含相同的元素

- 至多有一个 null 元素
- 无序

```
public class TestSet {

    public static void main(String[] args) {
        Set<Object> set = new HashSet<Object>();// Set是接口，只能通过子类进行实例化
        // 添加元素的方法add()
        set.add("hello");
        set.add("world");
        set.add("world");// 重复元素
        set.add(null);
        set.add(null);
        set.add('C');
        set.add('A');
        set.add('B');
        System.out.println(set);// 输出结果：[null, hello, A, B, C, world]
    }

}
```

Set 集合中不允许出现相同的项，Set 集合在用 Add()方法添加一个新项时，首先会调用 equals(Object o)来比较新项和已有的某项是否相等，而不是用==来判断相等性，所以对于字符串等已重写 equals 方法的类，是按值来比较相等性的 。

案例 1、

```
/**
 * 没有重写equals()
 * @author Administrator
 *
 */
public class User {



}


public class TestSet {

    public static void main(String[] args) {

        Set<User> set = new HashSet<User>();

        User u1 = new User();
        User u2 = new User();
        //User没有equals()方法，调用的是父类的(Object)的equals()方法
        System.out.println(u1.equals(u2));//false

        set.add(u1);
        set.add(u2);
        System.out.println(set.size());//2
```

```
    }

}
```

案例 2、

```
/**
 * 重写equals(),使得方法的返回值都是true
 * @author Administrator
 *
 */
public class User {

    @Override
    public boolean equals(Object obj) {
        // TODO Auto-generated method stub
        return true;
    }

}
```

```
package com.mxp.set;

import java.util.HashSet;
import java.util.Set;
```

```java
public class TestSet {

    public static void main(String[] args) {

        Set<User> set = new HashSet<User>();

        User u1 = new User();
        User u2 = new User();
        //User有equals()方法，调用的是自己的equals()方法
        System.out.println(u1.equals(u2));//true

        set.add(u1);
        set.add(u2);
        System.out.println(set.size());//2

    }

}
```

迭代遍历:

```java
public class TestHashSet {

    public static void main(String[] args) {
        Set<String> set = new HashSet<String>();

        set.add("A");
        set.add("B");
```

```
            set.add("C");
            set.add("D");
            Iterator<String> its = set.iterator();

            while(its.hasNext()){
                System.out.println(its.next());
            }
        }

}
```

增强 for 循环遍历

```
public class TestHashSet {

    public static void main(String[] args) {
        Set<String> set = new HashSet<String>();

        set.add("A");
        set.add("B");
        set.add("C");
        set.add("D");
        for (String s : set) {
            System.out.println(s);
        }
    }

}
```

2、HashSet

使用 HashSet 的优点就是：查询效率特别的高，而且在增删元素的时候，效率也很高，因为是通过哈希码来实现的。

使用 HashSet 的缺点就是：使用的空间比较大，这是为了避免散列冲突。

java 中的哈希码：哈希码是一种算法，让同一个类的对象按照自己不同的特征尽量有不同的哈希码(int 值)；

1：Object 类的 hashCode.返回对象的内存地址经过处理后的结构，由于每个对象的内存地址都不一样，所以哈希码也不一样。

```
/**
 * 重写hashCode()使得每个对象的hashCode返回值都是1
 * @author Administrator
 *
 */
public class User {

    @Override
    public int hashCode() {
        return 1;
    }
}


public class TestHashCode {

    public static void main(String[] args) {
```

```
        User u1 = new User();

        User u2 = new User();

        System.out.println(u1.hashCode());//1
        System.out.println(u2.hashCode());//1
    }

}
```

2：String 类的 hashCode.根据 String 类包含的字符串的内容，根据一种特殊算法返回哈希码，只要字符串所在的堆空间相同，返回的哈希码也相同，不同的字符串也可能会返回相同的 hash 值，不同的堆空间地址也可能会有相同的 hash 值。

```
public class TestHashCode {

    public static void main(String[] args) {
        String a = "gdejicbegh";
        String b = "hgebcijedg";

        System.out.println(a.hashCode());//-801038016
        System.out.println(b.hashCode());//-801038016
        System.out.println(a.hashCode()==b.hashCode());

    }

}
```

3：Integer 类，返回的哈希码就是 Integer 对象里所包含的那个整数的数值，例如 Integer i1=new Integer(100),i1.hashCode 的值就是 100 。由此可见，2 个一样大小的 Integer 对象，返回的哈希码也一样。

HashSet:底层数据结构是哈希表，线程是不同步的。 无序，高效；

HashSet 集合保证元素唯一性：通过元素的 hashCode 方法，和 equals 方法完成的。

当元素的 hashCode 值相同时，才继续判断元素的 equals 是否为 true。

如果为 true，那么视为相同元素，不存。如果为 false，那么存储。

如果 hashCode 值不同，那么不判断 equals，从而提高对象比较的速度。

```
/**
 * 重写hashCode()使得每个对象的hashCode返回值都是1
 * 重写equals()使得返回值都是true
 * @author Administrator
 *
 */
public class User {


    @Override
    public int hashCode() {
        return 1;
    }

    @Override
    public boolean equals(Object obj) {
        // TODO Auto-generated method stub
        return true;
    }
}
```

```
public class TestHashSet {

    public static void main(String[] args) {
        Set<User> set = new HashSet<User>();

        set.add(new User());
        set.add(new User());
        set.add(new User());
        //仅仅重写hashCode或者equals方法后都是3，重写两个方法后是1
        System.out.println(set.size());
    }

}
```

3、TreeSet

Treeset 中的数据是自动排好序的，不允许放入 null 值．TreeSet 方法保证元素唯一性的方式： 就是参考比较方法的结果是否为 0，如果 return 0，视为两个对象重复，不存。

```
public class TestTreeSet {

    public static void main(String[] args) {

        Set<Object> set = new TreeSet<Object>();

        set.add(1);
        set.add("A");
```

```
        set.add(null);//java.lang.ClassCastException
        System.out.println(set);

    }

}
```

TreeSet 是依靠 TreeMap 来实现的。

TreeSet 是一个有序集合，TreeSet 中元素将按照升序排列，缺省是按照自然顺序进行排列，意味着 TreeSet 中元素要实现 Comparable 接口。

我们可以在构造 TreeSet 对象时，传递实现了 Comparator 接口的比较器对象。

TreeSet 集合排序有两种方式， Comparable 和 Comparator 区别：

1：让元素自身具备比较性，需要元素对象实现 Comparable 接口，覆盖 compareTo 方法。

2：让集合自身具备比较性，需要定义一个实现了 Comparator 接口的比较器，并覆盖 compare 方法，并将该类对象作为实际参数传递给 TreeSet 集合的构造函数。第二种方式较为灵活。

```
/**
 * 实现Comparable接口重写compareTo方法
 * @author Administrator
```

```
 *
 */
public class User implements Comparable<User> {

    private String name;

    private int age;

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

    @Override
    public int compareTo(User o) {
        if (o.name.compareTo(this.name) > 0)
            return -1;
        if (o.name.compareTo(this.name) < 0)
            return 1;
        if (o.age > this.age)
            return -1;
        if (o.age < this.age)
            return 1;
        return 0;
    }

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

```
    }

}

/**
 * 实现Comparator接口，在创建TreeSet对象的时候调用参数类型是Comparator的构造器
 * @author Administrator
 *
 */
public class MyComparator implements Comparator<User> {

    @Override
    public int compare(User o1, User o2) {
        if (o1.getName().compareTo(o2.getName()) > 0)
            return 1;
        if (o1.getName().compareTo(o2.getName()) < 0)
            return -1;
        if (o1.getAge() > o2.getAge())
            return 1;
        if (o1.getAge() < o2.getAge())
            return -1;
        return 0;
    }

}
```

六、 Map 接口

Map 以按键/数值对的形式存储数据，和数组非常相似，在数组中存在的索引，它们本身也是对象；

|--HashMap： 底层是哈希表数据结构，是线程不同步的。可以存储 null 键， null 值。替代了 Hashtable.
|--Hashtable： 底层是哈希表数据结构，是线程同步的。不可以存储 null 键， null 值。
|--TreeMap： 底层是二叉树结构，可以对 map 集合中的键进行指定顺序的排序。

Map 集合存储和 Collection 有着很大不同：
Collection 一次存一个元素； Map 一次存一对元素。
Collection 是单列集合； Map 是双列集合。
Map 中的存储的一对元素：一个是键，一个是值， 键与值之间有对应(映射)关系。
特点：要保证 map 集合中键的唯一性。

1，添加。
put(key,value)：当存储的键相同时，新的值会替换老的值，并将老值返回。如果键没有重复，返回 null。
void putAll(Map);

2，删除。
void clear()：清空
value remove(key) ：删除指定键。

3，判断。
boolean isEmpty()：
boolean containsKey(key)：是否包含 key
boolean containsValue(value) ：是否包含 value

4，取出。
int size()：返回长度

value get(key) ：通过指定键获取对应的值。如果返回 null，可以判断该键不存在。当然有特殊情况，就是在 hashmap 集合中，是可以存储 null 键 null 值的。Collection values()： 获取 map 集合中的所有的值。

5，想要获取 map 中的所有元素：

原理： map 中是没有迭代器的， collection 具备迭代器，只要将 map 集合转成 Set 集合，可以使用迭代器了。之所以转成 set，是因为 map 集合具备着键的唯一性，其实 set 集合就来自于 map， set 集合底层其实用的就是 map 的方法。

★ 把 map 集合转成 set 的方法：

Set keySet();

Set entrySet();//取的是键和值的映射关系。

Entry 就是 Map 接口中的内部接口；

为什么要定义在 map 内部呢？ entry 是访问键值关系的入口，是 map 的入口，访问的是 map 中的键值对。

Map 遍历的方法一、

```
Map<String, Object> map = new HashMap<String, Object>();

map.put("A", "hello");
map.put("B", "world");

Set<String> keys = map.keySet();

Iterator<String> its = keys.iterator();
while (its.hasNext()) {
    String key = its.next();

    System.out.println("key:" + key + "|value:" + map.get(key));
```

```
}
```

Map 遍历的方法二、

```
Map<String, Object> map = new HashMap<String, Object>();

map.put("A", "hello");
map.put("B", "world");

Set<Entry<String, Object>> sets = map.entrySet();

Iterator<Entry<String, Object>> its = sets.iterator();

while (its.hasNext()) {
    Entry<String, Object> en = its.next();
    System.out.println(en.getKey() + "==>" + en.getValue());
}
```

使用集合的技巧：

看到 Array 就是数组结构，有角标，查询速度很快。

看到 link 就是链表结构：增删速度快，而且有特有方法。addFirst； addLast； removeFirst()； removeLast()；getFirst()； getLast()；

看到 hash 就是哈希表，就要想要哈希值，就要想到唯一性，就要想到存入到该结构的中的元素必须覆盖 hashCode， equals 方法。

看到 tree 就是二叉树，就要想到排序，就想要用到比较。

比较的两种方式：

一个是 Comparable：覆盖 compareTo 方法；

一个是 Comparator：覆盖 compare 方法。
LinkedHashSet， LinkedHashMap:这两个集合可以保证哈希表有存入顺序和取出顺序一致，保证哈希表有序。
集合什么时候用？
当存储的是一个元素时，就用 Collection。当存储对象之间存在着映射关系时，就使用 Map 集合。
保证唯一，就用 Set。不保证唯一，就用 List。

Collections： 它的出现给集合操作提供了更多的功能。这个类不需要创建对象，内部提供的都是静态方法。

Collection 和 Collections 的区别：
Collections 是个 java.util 下的类，是针对集合类的一个工具类,提供一系列静态方法,实现对集合的查找、排序、替换、线程安全化（将非同步的集合转换成同步的） 等操作。Collection 是个 java.util 下的接口，它是各种集合结构的父接口，继承于它的接口主要有 Set 和 List,提供了关于集合的一些操作,如插入、删除、判断一个元素是否其成员、遍历等