API基础第三天：

回顾：

1. 正则表达式：
    - 验证字符串格式是否符合要求
2. String支持的正则表达式的方法：
    - matches()：验证匹配
    - replaceAll()：替换
    - split()：拆分
3. Object：是所有类的顶级超类，万物皆对象，为了多态
    - 输出引用类型变量时，默认调用Object类的toString()方法，输出格式为：类的全称@hashcode，但是没有参考意义，所以常常重写toString()来返回对象的属性值
    - Object类的equals()方法默认比较的还是==(即比较地址)，但是没有参考意义，所以常量重写equals()来比较对象属性的值是否相同
4. 包装类：
    - 给8种基本类型套了个壳，使基本类型可以通过包装类的形式存在，直接参与面向对象的开发
    - jdk1.5提供了自动拆装箱特性，当基本类型与包装类型之间赋值时，会自动触发拆装箱，但本质上自动补齐代码

精华笔记：

1. 什么是集合：
    - 集合和数组一样，可以保存一组数据，并且提供了操作数组元素的相关方法，使用用更加方便
2. 集合框架中的相关接口：
    - java.util.Collection接口：是所有集合的顶级接口，封装了所有集合所共有的方法，下面有很多实现类，因此我们可以有更多的数据结构来选择。
    - Collection接口下面有两种常见的子接口：----------明天详细介绍
        - java.util.List：线性表，是可以重复集合，并且有序
        - java.util.Set：不可重复集合，大部分实现类是无序的
3. Collection的常用方法：
    - add()：向集合中添加一个元素，成功添加则返回true，否则返回false
    - size()：返回当前集合的元素个数
    - isEmpty()：判断当前集合是否为空集，当且仅当size()为0时返回true
    - clear()：清空集合
    - contains()：判断集合是否包含某元素(equals()相等即为true)
    - remove()：从集合中删除给定元素(equals()相等即为true)，成功删除则返回true
    - addAll()：将参数集合中的元素添加到当前集合中，添加后当前集合发生改变则返回true
    - containsAll()：判断当前集合中是否包含参数集合中的所有元素
    - retainAll()：取交集(交集元素留着)
    - removeAll()：删交集(交集元素删除)
    - iterator()：获取一个用于遍历当前集合元素的迭代器
    - toArray()：将集合转换为数组

4. 集合的遍历：
    - Collection接口提供了统一的遍历集合的方式：迭代器模式。通过iterator()方法可以获取一个用于遍历当前集合元素的迭代器(Iterator接口)
    - java.util.Iterator接口：定义了迭代器遍历集合的相关操作，不同的集合都实现了用于遍历自身元素的迭代器实现类，但我们无需记住它们的名字，从多态的角度把它们看成Iterator即可。
    - 迭代器遍历遵循的步骤为：问(hasNext())、取(next())、删(remove())，但删除并不是必要操作
5. 增强for循环/新循环：
    - jdk1.5时推出了一个特性：增强型for循环，也称为新循环，让我们使用相同的语法遍历集合和数组。它的内部是通过Iterator实现的，所以不能动态的增删元素。
    - 语法：

```
for(元素类型 变量名 : 集合或数组){
    循环体
}
```

6. 泛型：
    - JDK1.5时推出了一个特性：泛型
    - 泛型也称为参数化类型，允许我们在使用一个类时，传入某个类型来规定其内部的属性、方法参数或返回值类型，使得我们使用时更加方便。
        - 泛型在集合中被广泛使用，用来指定集合中元素的类型
        - 若不指定泛型的具体类型，则默认为Object
        - 若指定了泛型的具体类型，则在获取泛型的值时，编译器会自动补充强转操作
7. 集合与数组的互转：
    - 集合转换为数组：
    - 数组转换为集合：

笔记：

1. 什么是集合：
    - 集合和数组一样，可以保存一组数据，并且提供了操作数组元素的相关方法，使用用更加方便
2. 集合框架中的相关接口：
    - java.util.Collection接口：是所有集合的顶级接口，封装了所有集合所共有的方法，下面有很多实现类，因此我们可以有更多的数据结构来选择。
    - Collection接口下面有两种常见的子接口：----------明天详细介绍
        - java.util.List：线性表，是可以重复集合，并且有序
        - java.util.Set：不可重复集合，大部分实现类是无序的
3. Collection的常用方法：
    - add()：向集合中添加一个元素，成功添加则返回true，否则返回false
    - size()：返回当前集合的元素个数
    - isEmpty()：判断当前集合是否为空集，当且仅当size()为0时返回true
    - clear()：清空集合
    - contains()：判断集合是否包含某元素(equals()相等即为true)

- remove()：从集合中删除给定元素(equals()相等即为true)，成功删除则返回true

```
public class CollectionDemo {
    public static void main(String[] args) {
        Collection c = new ArrayList();
        c.add(new Point(1,2));
        c.add(new Point(3,4));
        c.add(new Point(5,6));
        c.add(new Point(7,8));
        c.add(new Point(9,0));
        c.add(new Point(1,2));
        //[元素1.toString(), 元素2.toString(), 元素3.toString(), ......]
        System.out.println(c); //[(1,2), (3,4), (5,6), (7,8), (9,0),
(1,2)]

        /*
          boolean contains(Object o):
          判断当前集合是否包含给定元素o
          判断依据是给定元素是否与当前集合存在equals()比较为true的情况
         */
        Point p = new Point(1,2);
        boolean contains = c.contains(p);
        System.out.println("是否包含:"+contains); //true

        /*
          boolean remove(Object o):-----一般都不接收boolean结果
          从当前集合中删除与给定元素o的equals()比较为true的元素
          若存在重复元素则只删除一次
         */
        c.remove(p);
        System.out.println(c);


        //集合中存放的是元素的引用
        Collection cc = new ArrayList();
        Point pp = new Point(1,2);
        cc.add(pp); //将pp添加到cc中-----------将pp的引用装到了cc中
        System.out.println("pp:"+pp); //(1,2)
        System.out.println("cc:"+cc); //[(1,2)]

        pp.setX(100);
        System.out.println("pp:"+pp); //(100,2)
        System.out.println("cc:"+cc); //[(100,2)]

        /*
        Collection c = new ArrayList();
        c.add("one");
        c.add("two");
        c.add("three");
        c.add("four");
        c.add("five");
        //集合重写了Object的toString()方法，返回格式如下:
        //[元素1.toString(), 元素2.toString(), 元素3.toString(), ......]
        System.out.println(c); //[one, two, three, four, five]
```

```
        System.out.println("size:"+c.size()); //5，输出集合的元素个数
        //isEmpty()判断集合是否为空集(size()为0表示空集)
        System.out.println("是否为空集"+c.isEmpty()); //false

        c.clear(); //清空集合
        System.out.println("集合已清空");
        System.out.println(c); //[]
        System.out.println("size:"+c.size()); //0
        System.out.println("是否为空集:"+c.isEmpty()); //true
        */
    }
}
```

- addAll()：将参数集合中的元素添加到当前集合中，添加后当前集合发生改变则返回true
- containsAll()：判断当前集合中是否包含参数集合中的所有元素
- ratainAll()：取交集(交集元素留着)
- removeAll()：删交集(交集元素删除)

```
public class CollectionOperDemo {
    public static void main(String[] args) {
        Collection c1 = new ArrayList();
        c1.add("java");
        c1.add("c++");
        c1.add(".net");
        System.out.println("c1:"+c1); //c1:[java, c++, .net]

        Collection c2 = new ArrayList();
        c2.add("android");
        c2.add("ios");
        c2.add("java");
        System.out.println("c2:"+c2); //c2:[android, ios, java]

        c1.addAll(c2); //将c2添加到c1中
        System.out.println("c1:"+c1); //c1:[java, c++, .net, android,
ios, java]
        System.out.println("c2:"+c2); //c2:[android, ios, java]

        Collection c3 = new ArrayList();
        c3.add("c++");
        c3.add("android");
        c3.add("php");
        System.out.println("c3:"+c3); //[c++, android, php]

        boolean contains = c1.containsAll(c3); //判断c1中是否包含c3中的所有
元素
        System.out.println("是否包含:"+contains); //false

        /*
        //取交集:c1中仅保留与c3所共有的元素，而c3不变
        c1.retainAll(c3);
        System.out.println("c1:"+c1); //c1:[c++, android]
        System.out.println("c3:"+c3); //c3:[c++, android, php]
        */
```

```
        //删交集:将c1中与c3共有的元素删除，c3不变
        c1.removeAll(c3);
        System.out.println("c1:"+c1); //c1:[java, .net, ios, java]
        System.out.println("c3:"+c3); //c3:[c++, android, php]
    }
}
```

- iterator()：获取一个用于遍历当前集合元素的迭代器
- toArray()：将集合转换为数组

4. 集合的遍历：
   - Collection接口提供了统一的遍历集合的方式：迭代器模式。通过iterator()方法可以获取一个用于遍历当前集合元素的迭代器(Iterator接口)
   - java.util.Iterator接口：定义了迭代器遍历集合的相关操作，不同的集合都实现了用于遍历自身元素的迭代器实现类，但我们无需记住它们的名字，从多态的角度把它们看成Iterator即可。
   - 迭代器遍历遵循的步骤为：问(hasNext())、取(next())、删(remove())，但删除并不是必要操作

```
public class IteratorDemo {
    public static void main(String[] args) {
        Collection c = new ArrayList();
        c.add("one");
        c.add("#");
        c.add("two");
        c.add("#");
        c.add("three");
        c.add("#");
        c.add("four");
        c.add("#");
        c.add("five");
        System.out.println(c); //[one, #, two, #, three, #, four, #,
five]

        /*
          迭代器的常用方法:
          1)boolean hasNext()----------问(必要操作)
            询问集合是否还有"下一个"元素可供迭代
            注意：迭代器默认开始位置在集合第1个元素之前
                  无论调用了多少次的hasNext()，迭代器的位置都不会改变
          2)Object next()--------------取(必要操作)
            迭代器向后移动一个位置来指向集合的下一个元素并将其获取
          3)void remove()--------------删(删除并非必要操作)
            删除next()方法所获取的元素
         */
        Iterator it = c.iterator(); //获取集合c的迭代器
        while(it.hasNext()){ //若有下一个元素
            String str = (String)it.next(); //获取下一个元素并强转为String类
型
            System.out.println(str);
            if("#".equals(str)){
                //c.remove(str); //迭代器遍历过程中不允许通过集合的方式来增删元
素，否则会报异常
```

```
                it.remove(); //删除next()方法所获取的元素
            }
        }
        System.out.println(c);
    }
}
```

5. 增强for循环/新循环：

- jdk1.5时推出了一个特性：增强型for循环，也称为新循环，让我们使用相同的语法遍历集合和数组。它的内部是通过Iterator实现的，所以不能动态的增删元素。
- 语法：

```
for(元素类型 变量名 : 集合或数组){
    循环体
}
```

```
public class NewForDemo {
    public static void main(String[] args) {
        String[] array = {"one","two","three","four","five"};
        for(int i=0;i<array.length;i++){
            System.out.println(array[i]);
        }
        for(String str : array){ //str表示的就是数组中的每个元素
            System.out.println(str);
        }

        Collection c = new ArrayList();
        c.add("one");
        c.add("two");
        c.add("three");
        c.add("four");
        c.add("five");
        Iterator it = c.iterator();
        while(it.hasNext()){
            String str = (String)it.next();
            System.out.println(str);
        }
        for(Object obj : c){ //obj代表集合中的每个元素
            String str = (String)obj;
            System.out.println(str);
        }

    }
}
```

6. 泛型：

- JDK1.5时推出了一个特性：泛型
- 泛型也称为参数化类型，允许我们在使用一个类时，传入某个类型来规定其内部的属性、方法参数或返回值类型，使得我们使用时更加方便。
  - 泛型在集合中被广泛使用，用来指定集合中元素的类型
  - 若不指定泛型的具体类型，则默认为Object

- 若指定了泛型的具体类型，则在获取泛型的值时，编译器会自动补充强转操作

```
public class GenericDemo {
    public static void main(String[] args) {
        Collection<Point> c = new ArrayList<>();
        c.add(new Point(1,2));
        c.add(new Point(3,4));
        c.add(new Point(5,6));
        c.add(new Point(7,8));

        Iterator<Point> it = c.iterator();
        while(it.hasNext()){
            Point p = it.next();
            System.out.println(p);
        }

        for(Point p : c){
            System.out.println(p);
        }

        /*
        Collection<String> c = new ArrayList<>(); //泛型集合
        c.add("one");
        c.add("two");
        c.add("three");
        c.add("four");
        c.add("five");
        //c.add(123); //编译错误，123的类型违背了集合c所指向的泛型的实际类型
String

        //迭代器所指定的泛型应当与其遍历的集合的泛型一致
        Iterator<String> it = c.iterator();
        while(it.hasNext()){
            String str = it.next();
            System.out.println(str);
        }

        for(String str : c){
            System.out.println(str);
        }
         */
    }
}
```

7. 集合与数组的互转：
    - 集合转换为数组：

```
public class CollectionToArray {
    public static void main(String[] args) {
        Collection<String> c = new ArrayList<>();
        c.add("one");
        c.add("two");
        c.add("three");
        c.add("four");
```

```
        c.add("five");
        System.out.println(c); //[one, two, three, four, five]

        //若参数数组元素个数==集合元素个数，那就正常转换
        //若参数数组元素个数<集合元素个数，那也正常转换(按照集合大小给数组)
        //若参数数组元素个数>集合元素个数，那也正常转换，同时在末尾补默认值
        String[] array = c.toArray(new String[5]);
        System.out.println(Arrays.toString(array)); //[one, two, three,
four, five]

    }
}
```

- 数组转换为集合：

```
public class ArrayToList {
    public static void main(String[] args) {
        String[] array = {"one", "two", "three", "four", "five"};
        System.out.println("array:"+ Arrays.toString(array)); //[one,
two, three, four, five]

        //asList()方法会返回内部的ArrayList，内部直接引用给定数组array
        List<String> list = Arrays.asList(array);
        System.out.println("list:"+list); //[one, two, three, four,
five]

        //对数组操作后，集合也会相应的改变
        array[1] = "six";
        System.out.println("array:"+ Arrays.toString(array));
        System.out.println("list:"+list);

        //对集合操作后，数组也会做相应的改变
        list.set(2,"seven"); //将集合的第3个元素修改为seven
        System.out.println("array:"+ Arrays.toString(array));
        System.out.println("list:"+list);

        //给集合添加/删除元素相当于给数组添加/删除元素
        //而数组是定长的，不会自动扩容/缩容，因此发生不支持操作异常
        list.add("!!!!"); //运行时会发生不支持操作异常
    }
}
```

补充：

1. 无