API基础第二天：

回顾：

1. String类：不变对象，字符串对象一旦创建好，对象内容永远无法改变，但引用可以重新赋值(指向新的对象)
2. 常量池：字面量创建的字符串对象，都会缓存到常量池中，当再次使用相同字面量创建对象时，将会复用常量池中的引用，以减少内存开销。
3. String的常用方法：
    - length()：长度
    - trim()：去除两边的空白字符
    - toUpperCase()/toLowerCase()：转全大写/全小写
    - startsWith()/endsWith()：判断以...开始/结束的
    - charAt()：根据位置找字符
    - indexOf()/lastIndexOf()：获取字符串的位置
    - substring()：截取
    - 静态valueOf()：将其它类型转换为String字符串类型
4. StringBuilder类：频繁修改字符串
    - append()：追加
    - delete()：删除部分
    - replace()：修改部分
    - insert()：插入

精华笔记：

1. 正则表达式：
    - 用于描述字符串的内容格式，使用它通常用于匹配一个字符串是否符合格式要求
    - 正则表达式的语法：----------了解，不用纠结、不用深入研究
2. String支持正则表达式的方法：
    - matches()：匹配，使用给定的正则表达式验证当前字符串的格式是否符合要求，若符合则返回true，不符合则返回false
    - replaceAll()：替换，将当前字符串中满足正则表达式的部分给替换为指定字符串
    - split()：拆分，将当前字符串按照满足正则表达式的部分进行拆分，将拆分出的以String[]形式返回
3. Object：
    - 是所有类的鼻祖，所有类都直接或间接继承了Object，万物皆对象，为了多态
    - Object中有两个经常被派生类重写的方法：toString()和equals()
        - 调用Object类的toString()时默认返回：类的全称@hashcode值，没有参考意义，所以常常重写toString()来返回具体的属性值

            > 注意：String、StringBuilder都重写了toString()来返回具体的字符串内容了

        - 调用Object类的equals()时默认比较的还是==(即比较地址)，没有参考意义，所以常常重写equals()来比较具体的属性值

> 注意事项：
>
> 1. String类、包装类重写了equals()来比较内容是否相等，但StringBuilder并没有。
> 2. 重写equals()的基本原则：
>   - 原则上要比较两个对象的属性值是否相同
>   - 两个对象必须是同一类型的，若类型不同则直接返回false

4. 包装类：
   - java定义了8个包装类，目的就是为了解决基本类型不能直接参与面向对象开发的问题，使基本类型可以通过包装类的形式存在。
   - 包含：Integer、Character、Byte、Short、Long、Float、Double、Boolean，其中Character和Boolean是直接继承自Object类的，其余6个包装类继承自Number类。
   - JDK1.5时推出了一个新特性：自动拆装箱。当编译器编译时若发现是基本类型与包装类型之间的相互赋值，则自动补齐代码完成转换工作，这个过程叫做自动拆装箱。

笔记：

1. 正则表达式：
   - 用于描述字符串的内容格式，使用它通常用于匹配一个字符串是否符合格式要求
   - 正则表达式的语法：----------了解，不用纠结、不用深入研究

```
1.[]:表示一个字符,该字符可以是[]中指定的内容
  例如:
  [abc]:这个字符可以是a或b或c
  [a-z]:表示任意一个小写字母
  [a-zA-Z]:表示任意一个字母
  [a-zA-Z0-9]:表示任意一个字母数字
  [a-zA-Z0-9_]:表示任意一个数字字母下划线
  [^abc]:该字符只要不是a或b或c
2.预定义字符:
  .:表示任意一个字符,没有范围限制
  \d:表示任意一个数字,等同于[0-9]
  \w:表示任意一个单词字符,等同于[a-zA-Z0-9_]----单词字符指字母、数字和_
  \s:表示任意一个空白字符
  \D:不是数字
  \W:不是单词字符
  \S:不是空白字符
3.量词:
  ?:表示前面的内容出现0-1次
    例如: [abc]? 可以匹配:a 或 b 或 c 或什么也不写
          但是不能匹配：m或aaa
  +:表示前面的内容最少出现1次
    例如: [abc]+ 可以匹配:b或aaaaaaaaaa...或abcabcbabcbabcba....
          但是不能匹配:什么都不写 或 abcfdfsbbaqbb34bbwer...
  *:表示前面的内容出现任意次(0-多次)---匹配内容与+一致，只是可以一次都不写
    例如: [abc]* 可以匹配:b或aaaaaaaaaa...或abcabcba....或什么都不写
          但是不能匹配:abcfdfsbbaqbb34bbwer...
  {n}:表示前面的内容出现n次
      例如: [abc]{3} 可以匹配:aaa 或 bbb 或 aab 或abc 或bbc
            但是不能匹配: aaaa 或 aad
```

```
  {n,m}:表示前面的内容出现最少n次最多m次
        例如: [abc]{3,5} 可以匹配:aaa 或  abcab 或者 abcc
        但是不能匹配:aaaaaa 或 aabbd
  {n,}:表示前面的内容出现n次以上(含n次)
       例如: [abc]{3,} 可以匹配:aaa 或 aaaaa.... 或 abcbabbcbabcba....
            但是不能匹配:aa 或 abbdaw...
4.()用于分组,是将小括号里面的内容看做是一个整体
  例如: (abc){3} 表示abc整体出现3次. 可以匹配abcabcabc
        但是不能匹配aaa 或abc 或abcabccba
        (abc|def){3}表示abc或def整体出现3次.
        可以匹配: abcabcabc 或 defdefdef 或 abcdefabc
        但是不能匹配abcdef 或abcdfbdef
```

2. String支持正则表达式的方法：

- matches()：匹配，使用给定的正则表达式验证当前字符串的格式是否符合要求，若符合则返回true，不符合则返回false

```
public class MatchesDemo {
    public static void main(String[] args) {
        /*
          邮箱正则表达式:
            [a-zA-Z0-9_]+@[a-zA-Z0-9]+(\.[a-zA-Z]+)+
          注意:
            \.中的\是正则表达式中的转义字符
            \\.中的第1个\，是java中的转义符，是在转义正则表达式中的\
         */
        String email = "wangkj@tedu.cn";
        String regex = "[a-zA-Z0-9_]+@[a-zA-Z0-9]+(\\.[a-zA-Z]+)+";
        //使用regex匹配email是否符合格式要求
        boolean match = email.matches(regex);
        if(match){
            System.out.println("是正确的邮箱格式");
        }else{
            System.out.println("不是正确的邮箱格式");
        }
    }
}
```

- replaceAll()：替换，将当前字符串中满足正则表达式的部分给替换为指定字符串

```
public class ReplaceAllDemo {
    public static void main(String[] args) {
        String line = "abc123def456ghi78";
        //将数字部分替换为#NUMBER#
        line = line.replaceAll("[0-9]+","#NUMBER#");
        System.out.println(line);
    }
}
```

- split()：拆分，将当前字符串按照满足正则表达式的部分进行拆分，将拆分出的以String[]形式返回

```
public class SplitDemo {
```

```
    public static void main(String[] args) {
        String line = "abc123def456ghi";
        String[] data = line.split("[0-9]+"); //按数字拆分(数字就拆没了)
        System.out.println(Arrays.toString(data)); //将data数组转换为字符串
并输出

        line = "123.456.78";
        data = line.split("\\."); //按.拆(.就拆没了)
        System.out.println(Arrays.toString(data));

        //最开始就是可拆分项，那么数组第1个元素为空字符串----------""
        //如果连续两个(两个以上)可拆分项，那么中间也会拆出一个空字符串------""
        //如果末尾连续多个可拆分项，那么拆出的空字符串被忽略(不要了)
        line = ".123.456..78........";
        data = line.split("\\."); //按.拆(.就拆没了)
        System.out.println(Arrays.toString(data));
        System.out.println(data.length); //5
    }
}
```

3. Object：
   - 是所有类的鼻祖，所有类都直接或间接继承了Object，万物皆对象，为了多态
   - Object中有两个经常被派生类重写的方法：toString()和equals()
     - 调用Object类的toString()时默认返回：类的全称@hashcode值，没有参考意义，所以常常重写toString()来返回具体的属性值

       > 注意：String、StringBuilder都重写了toString()来返回具体的字符串内容了

       ```
       package object;

       import java.util.Objects;

       public class Point {
           private int x;
           private int y;

           public Point() {
           }

           public Point(int x, int y) {
               this.x = x;
               this.y = y;
           }

           @Override
           public String toString() {
               return "Point{" +
                       "x=" + x +
                       ", y=" + y +
                       '}';
           }

           public int getX() {
               return x;
       ```

```
    }

    public void setX(int x) {
        this.x = x;
    }

    public int getY() {
        return y;
    }

    public void setY(int y) {
        this.y = y;
    }
}

public class ObjectDemo {
    public static void main(String[] args) {
        /*
          输出引用变量时默认调用Object类的toString()方法
          该方法返回的字符串格式为: 类的全称@hashcode值
          但通常这个返回结果对我们的开发是没有意义的
          我们真正想输出的应该是对象的属性值
          我们认为Object类的toString()并不能满足需求
          因此我们常常重写toString()来返回具体的属性值
         */
        Point p = new Point(100,200);
        System.out.println(p);
        System.out.println(p.toString());
    }
}
```

- 调用Object类的equals()时默认比较的还是==(即比较地址)，没有参考意义，所以常常重写equals()来比较具体的属性值

> 注意事项:
>
> 1. String类、包装类重写了equals()来比较内容是否相等，但StringBuilder并没有。
> 2. 重写equals()的基本原则:
>   - 原则上要比较两个对象的属性值是否相同
>   - 两个对象必须是同一类型的，若类型不同则直接返回false

```
package object;

import java.util.Objects;

public class Point {
    private int x;
    private int y;

    public Point() {
    }

    public Point(int x, int y) {
```

```
        this.x = x;
        this.y = y;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Point point = (Point) o;
        return x == point.x && y == point.y;
    }

    @Override
    public int hashCode() {
        return Objects.hash(x, y);
    }

    @Override
    public String toString() {
        return "Point{" +
                "x=" + x +
                ", y=" + y +
                '}';
    }

    public int getX() {
        return x;
    }

    public void setX(int x) {
        this.x = x;
    }

    public int getY() {
        return y;
    }

    public void setY(int y) {
        this.y = y;
    }
}

public class ObjectDemo {
    public static void main(String[] args) {
        String s1 = new String("hello");
        String s2 = new String("hello");
        //String类重写了equals()来比较字符串内容是否相同
        System.out.println(s1.equals(s2)); //true

        StringBuilder builder1 = new StringBuilder("hello");
        StringBuilder builder2 = new StringBuilder("hello");
        //StringBuilder类没有重写equals()，因此调用的还是Object类的
equals()比较地址
        System.out.println(builder1.equals(builder2)); //false
```

```
        //s1与builder1的类型不同，所以equals()一定是false
        System.out.println(s1.equals(builder1)); //false

        /*
          调用Object类的equals()，内部还是在使用==比较地址，没有参考意义
          若想比较对象的属性值是否相同，我们认为Object类的equals()并不能满足
需求
          因此常常需要重写equals()
         */
        /*
        Point p1 = new Point(100,200);
        Point p2 = new Point(100,200);
        System.out.println(p1==p2); //false，==比较的是地址
        //因为Point类重写了equals()方法
        //所以此处调用的是重写之后的equals()，比较的是属性的值是否相同
        System.out.println(p1.equals(p2)); //true
        */
    }
}
```

4. 包装类：

   - java定义了8个包装类，目的就是为了解决基本类型不能直接参与面向对象开发的问题，使基本类型可以通过包装类的形式存在。
   - 包含：Integer、Character、Byte、Short、Long、Float、Double、Boolean，其中Character和Boolean是直接继承自Object类的，其余6个包装类继承自Number类。

```
//演示包装类的定义：
Integer i1 = new Integer(5);
Integer i2 = new Integer(5);
System.out.println(i1==i2); //false，==比较的是地址
System.out.println(i1.equals(i2)); //true，包装类重写equals()比较值了

//valueOf()方法会复用1个字节(-128到127)范围内的数据，建议使用valueOf()
Integer i3 = Integer.valueOf(5);
Integer i4 = Integer.valueOf(5);
System.out.println(i3==i4); //true
System.out.println(i3.equals(i4)); //true，包装类重写equals()比较值了

//将包装类转换为基本类型
int i = i4.intValue();
System.out.println(i);

//演示包装类的常用操作：
//1)可以通过包装类来获取基本类型的取值范围：
int max = Integer.MAX_VALUE; //获取int的最大值
int min = Integer.MIN_VALUE; //获取int的最小值
System.out.println("int的最大值为:"+max+"，最小值为:"+min);

long max1 = Long.MAX_VALUE; //获取long的最大值
long min1 = Long.MIN_VALUE; //获取long的最小值
System.out.println("long的最大值为:"+max1+"，最小值为:"+min1);

//2)包装类可以将字符串转换为对应的基本类型----必须熟练掌握
String s1 = "39";
```

```
int age = Integer.parseInt(s1); //将字符串s1转换为int类型并赋值给age
System.out.println(age); //39---int类型

String s2 = "123.456";
double price = Double.parseDouble(s2); //将字符串s2转换为double类型并赋值给
price
System.out.println(price); //123.456----double类型
```

- JDK1.5时推出了一个新特性：自动拆装箱。当编译器编译时若发现是基本类型与包装类型之间的相互赋值，则自动补齐代码完成转换工作，这个过程叫做自动拆装箱。

```
//触发了自动装箱特性，会被编译为: Integer i = Integer.valueOf(5);
Integer i = 5; //基本类型到包装类-------装箱
//触发了自动拆箱特性，会被编译为: int j = i.intValue();
int j = i; //包装类到基本类型-----------拆箱
```

补充：

1. Arrays.toString(数组)----------可以将数组转换为字符串
2. 明后日单词：

```
1)collection:集合
2)list:列表
3)size:大小
4)empty:空
5)clear:清空
6)contains:包含
7)remove:删除
8)retain:保留
9)iterator:迭代
10)next:下一个
11)reverse:反转
12)compare:比较
13)link:连接
14)LinkedList:链表
```