API基础第二天：

回顾：

1. String：

```
String s = new String("hello");
创建了2个对象，一个是"hello"字面量对象，一个是new String()对象，s指向new String()对象
```

2. String常用方法：
   - length：获取长度
   - trim()：去掉两边空白
   - toUpperCase()和toLowerCase()：转大写/小写
   - startsWith()和endsWith()：判断是否是以？开始/结束
   - charAt()：-----根据位置找字符
   - indexOf()：-----根据字符串找位置
   - substring()：截取
   - 静态方法valueOf()：将其它类型转换为字符串
3. StringBuilder：适用做频繁的修改，提供了增、删、改、插等操作
4. StringBuilder的常用方法：

```
1)append():追加内容
2)delete():删除内容
3)replace():替换内容
4)insert():插入内容
```

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

```
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/_
```

```
  \s:表示任意一个空白字符
  \D:表示不是数字
  \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
  +: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,是将括号内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abcabc
        (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支持与正则表达式相关的方法：

- matches()：
- replaceAll()：
- split()：

3. Object：对象/东西

- 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中有几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@地址，没有参考意义，所以常常重写toString()来返回具体属性的值

    > 注意：String、StringBuilder等都重写toString()来返回字符串内容了

  - 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：
    >
    > 1. String类已经重写equals()来比较字符串内容了，但StringBuilder并没有
    > 2. 重写equals()的基本规则：
    >   - 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，使得基本类型可以通过包装类的形式存在。
- 包括：Integer、Character、Byte、Short、Long、Float、Double、Boolean，其中Character和Boolean是直接继承自Object的，而其余6个包装类继承自java.lang.Number类。
- JDK1.5推出了一个新的特性：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相互赋值，将自动补充代码来完成转换工作，这个过程称为自动拆装箱。

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

```
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/_
  \s:表示任意一个空白字符
  \D:表示不是数字
  \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
  +: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,是将括号内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abcabc
        (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支持与正则表达式相关的方法:

- matches(): 使用给定的正则表达式(regex)验证当前字符串的格式是否符合要求，符合则返回true，否则返回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: \.中的这个\是正则表达式中的转义符
                \\.中的第1个\，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(): 将当前字符串中满足正则表达式(regex)的部分给替换为给定的字符串(s)

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

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

```
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
```

```
        line = ".123.456..78.......";
        data = line.split("\\."); //按.拆(.就拆没了)
        System.out.println(Arrays.toString(data));
    }
}
```

3. Object：对象/东西
   - 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中有几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@地址，没有参考意义，所以常常重写toString()来返回具体属性的值

       > 注意：String、StringBuilder等都重写toString()来返回字符串内容了

```
package apiday02;

import java.util.Objects;

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

    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会调用Object类的toString()方法
          该方法返回的字符串格式为: 类的全称@地址
          但通常这个返回结果对我们的开发是没有任何意义
          我们真正想输出的应该是对象的属性值，Object类的toString()并不能满足
需求
          因此常常需要重写toString()来返回具体的属性值
         */

        Point p = new Point(100,200);
        System.out.println(p); //输出引用变量时默认调用Object类的
toString()
        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:
>
> 1. String类已经重写equals()来比较字符串内容了，但StringBuilder并没有
> 2. 重写equals()的基本规则:
>   - 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，若类型不同则返回false

```
package apiday02;

import java.util.Objects;

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

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

    @Override
    public String toString() {
        return "Point{" +
                "x=" + x +
                ", 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() { //hashcode是散列值，现在先不用纠结，后面才讲到
        return Objects.hash(x, 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) {
        /*
          调用Object类的equals()，内部还是在使用==比较地址，没有实际意义
          若想比较对象的属性值是否相同，我们认为Object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
        //因为调用的是Point类重写之后的equals()，内部比较的是属性的值是否相同
        System.out.println(p1.equals(p2)); //true
        */

        String s1 = new String("hello");
        String s2 = new String("hello");
        //String重写equals()来比较字符串内容是否相同了
        System.out.println(s1.equals(s2)); //true

        StringBuilder builder1 = new StringBuilder("hello");
        StringBuilder builder2 = new StringBuilder("hello");
        //StringBuilder没有重写equals()，所以调用的还是Object的
equals()，还是比较地址
        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
```

```
    }
}
```

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，使得基本类型可以通过包装类的形式存在。
- 包括：Integer、Character、Byte、Short、Long、Float、Double、Boolean，其中Character和Boolean是直接继承自Object的，而其余6个包装类继承自java.lang.Number类。
- JDK1.5推出了一个新的特性：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相互赋值，将自动补充代码来完成转换工作，这个过程称为自动拆装箱。

```
public class IntegerDemo {
    public static void main(String[] args) {
        //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)包装类型可以将字符串转换为对应的基本类型------必须熟练掌握
        String s1 = "38";
        int age = Integer.parseInt(s1); //将字符串s1转换为int类型
        System.out.println(age); //38------int
        String s2 = "123.456";
        double price = Double.parseDouble(s2); //将字符串s2转换为double类型
        System.out.println(price); //123.456-------double

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

        /*
        Integer i1 = new Integer(5);
        Integer i2 = new Integer(5);
        System.out.println(i1==i2); //false，因为==是比较地址

        //Integer.valueOf()会复用-128到127范围内的数据---使用valueOf()方式更多一些
        Integer i3 = Integer.valueOf(5);
        Integer i4 = Integer.valueOf(5);
        System.out.println(i3==i4); //true
         */
    }
}
```

补充：

1. 将数组转换为字符串：
    - Arrays.toString(数组名)----将某个数组转换为字符串
2. 进制：

```
1)十进制：
  1.1)规则:逢10进1
  1.2)数字:0 1 2 3 4 5 6 7 8 9
  1.3)基数:10
  1.4)权:万   千   百   十   个
2)二进制：
  2.1)规则:逢2进1
  2.2)数字:0 1
  2.3)基数:2
  2.4)权:128 64 32 16 8 4 2 1
3)十六进制：
  3.1)规则:逢16进1
  3.2)数字:0 1 2 3 4 5 6 7 8 9 a b c d e f
  3.3)基数:16
  3.4)权:65536 4096 256 16 1
```

3. 十六进制的权:

```
16的0次幂-------------1
16的1次幂-------------16
16的2次幂-------------256
16的3次幂-------------4096
16的4次幂-------------65536
```

二进制的权：

```
2的0次幂-------------1
2的1次幂-------------2
2的2次幂-------------4
2的3次幂-------------8
2的4次幂-------------16
```

十进制的权：

```
10的0次幂------------1
10的1次幂------------10
10的2次幂------------100
10的3次幂------------1000
10的4次幂------------10000
```

4. 二进制转换为十进制的规则：所有为1的权相加----------正数

> 要求：今天必须熟练记住最后4个权(8421)

```
权:    32  16  8  4  2  1
二进制: 1   1   0  1  0  1
十进制: 32+16+4+1---------------53

权:    32  16  8  4  2  1
二进制: 0   0   1  1  0  1
十进制: 8+4+1-------------------13

权:    32  16  8  4  2  1
二进制: 1   0   1  0  1  0
十进制: 32+8+2------------------42
```

5. 明日单词:

```
1)binary:二进制
```