在Java中，**StringBuilder**是一个用于创建可变字符串的类。与String类不同，StringBuilder允许字符串被修改，而不需要创建新的字符串对象，这在处理大量字符串操作时能显著提高性能。以下是关于StringBuilder的一些常用方法及其介绍：

1. **构造方法**
   * StringBuilder(): 创建一个空的StringBuilder对象，初始容量为16个字符（但这一细节可能会因JVM实现而异）。
   * StringBuilder(int capacity): 创建一个指定初始容量的StringBuilder对象。
   * StringBuilder(String str): 创建一个包含指定字符串内容的StringBuilder对象，初始容量会自动设置为字符串长度加16。
2. **追加内容**
   * append(...): 将指定内容追加到StringBuilder的末尾。这个方法支持多种数据类型，如String、int、char、boolean等。每次追加后，StringBuilder的内部容量可能会自动调整以容纳新内容。
3. **插入内容**
   * insert(int offset, ...): 在指定位置插入内容。同样支持多种数据类型。插入操作可能会导致StringBuilder内部数组的扩容和内容的移动。
4. **替换内容**
   * replace(int start, int end, String str): 用新字符串替换指定范围内的字符。注意，这里的范围是不包含结束位置的。
5. **删除内容**
   * delete(int start, int end): 删除指定范围内的字符。同样，这里的范围是不包含结束位置的。
   * deleteCharAt(int index): 删除指定索引处的字符。
6. **反转字符串**
   * reverse(): 反转StringBuilder中的字符顺序。
7. **获取字符**
   * charAt(int index): 获取指定索引处的字符。
8. **设置字符**
   * setCharAt(int index, char ch): 将指定索引处的字符替换为新的字符。
9. **获取子字符串**
   * substring(int start, int end): 返回一个新的字符串，其中包含StringBuilder中指定范围内的字符。同样，这里的范围是不包含结束位置的。
10. **查找子字符串**
    * indexOf(String str): 返回指定子字符串第一次出现的索引。
    * lastIndexOf(String str): 返回指定子字符串最后一次出现的索引。
11. **长度和容量**
    * length(): 返回StringBuilder中字符的数量。
    * capacity(): 返回StringBuilder的当前容量，即可以存储的字符数。
    * ensureCapacity(int minCapacity): 确保StringBuilder的容量至少等于指定的最小值。如果当前容量小于最小值，则会进行扩容。
12. **设置长度**
    * setLength(int newLength): 设置StringBuilder的长度。如果新长度小于当前长度，那么多余的部分会被截断；如果新长度大于当前长度，那么超出部分的字符将会以null字符（实际上是'\u0000'）填充。
13. **转换为字符串**
    * toString(): 将StringBuilder对象转换为String对象。这是获取StringBuilder最终内容的最常用方法。
14. **比较**
    * equals(Object obj): 比较两个StringBuilder对象的引用是否相等。需要注意的是，这个方法并没有重写Object类中的equals方法，而是沿用了Object类的实现，因此它比较的是对象的引用而不是内容。如果要比较两个StringBuilder对象的内容是否相等，应该先将它们转换为String类型，然后使用String类的equals方法进行比较。

在Java中，**Math**类的具体使用方法涵盖了多个数学运算领域。以下是对Math类中各方法和功能的详细叙述：

**一、基本数学运算方法**

1. **求绝对值**
   * 使用Math.abs(x)方法，其中x可以是任意数值类型（如int、double等），该方法返回x的绝对值。
2. **平方根**
   * 使用Math.sqrt(x)方法，其中x是一个非负数，该方法返回x的平方根。如果x为负数，则返回NaN（Not-a-Number）。
3. **幂运算**
   * 使用Math.pow(a, b)方法，其中a是底数，b是指数，该方法返回a的b次方。
4. **取余**
   * 使用Math.remainder(dividend, divisor)方法，其中dividend是被除数，divisor是除数，该方法返回dividend除以divisor后的余数。这与使用%运算符的取余操作略有不同，因为Math.remainder方法遵循IEEE 754标准，能够处理浮点数的取余。

**二、三角函数方法**

1. **正弦、余弦、正切**
   * 使用Math.sin(x)、Math.cos(x)、Math.tan(x)方法，其中x是一个以弧度为单位的数值，这些方法分别返回x的正弦值、余弦值和正切值。
2. **反正弦、反余弦、反正切**
   * 使用Math.asin(x)、Math.acos(x)、Math.atan(x)方法，其中x是一个在相应函数定义域内的数值，这些方法分别返回x的反正弦值、反余弦值和反正切值。这些值也是以弧度为单位的。

**三、对数函数方法**

1. **自然对数**
   * 使用Math.log(x)方法，其中x是一个正数，该方法返回x的自然对数（以e为底的对数）。
2. **以10为底的对数**
   * 使用Math.log10(x)方法，其中x是一个正数，该方法返回x的以10为底的对数。

**四、数学常数**

* Math.PI：表示圆周率π的值，约为3.141592653589793。
* Math.E：表示自然对数的底数e的值，约为2.718281828459045。

**五、其他数学方法**

1. **向上取整**
   * 使用Math.ceil(x)方法，其中x是一个数值，该方法返回大于或等于x的最小整数。
2. **向下取整**
   * 使用Math.floor(x)方法，其中x是一个数值，该方法返回小于或等于x的最大整数。
3. **四舍五入**
   * 使用Math.round(x)方法，其中x是一个数值，该方法返回最接近x的整数。对于浮点数值，它采用“银行家舍入”规则：如果两个整数与x的距离相等，则取偶数整数。
4. **返回最接近的整数（可能返回偶数）**
   * 使用Math.rint(x)方法，其中x是一个数值，该方法返回最接近x的整数，如果两个整数与x的距离相等，则根据IEEE 754标准返回偶数整数（这一行为在某些情况下与Math.round略有不同）。
5. **生成随机数**
   * 使用Math.random()方法，该方法返回一个0.0（包括）到1.0（不包括）之间的随机浮点数。
6. **弧度与角度的转换**
   * 使用Math.toRadians(x)方法，其中x是一个角度值（以度为单位），该方法返回与x角度对应的弧度值。
   * 使用Math.toDegrees(x)方法，其中x是一个弧度值，该方法返回与x弧度对应的角度值（以度为单位）。

**常用泛型集合类**

| 集合类型 | 描述 | 示例声明 |
| --- | --- | --- |
| ArrayList<E> | 动态数组，支持快速随机访问 | List<String> list = new ArrayList<>(); |
| LinkedList<E> | 双向链表，适合频繁插入/删除 | List<Integer> linkedList = new LinkedList<>(); |
| HashSet<E> | 无序集合，不允许重复元素 | Set<String> set = new HashSet<>(); |
| TreeSet<E> | 有序集合（基于红黑树） | Set<Integer> treeSet = new TreeSet<>(); |
| HashMap<K,V> | 键值对映射，允许 null 键和值 | Map<String, Integer> map = new HashMap<>(); |
| TreeMap<K,V> | 有序键值对映射（基于红黑树） | Map<Integer, String> treeMap = new TreeMap<>(); |

List<String> names = new ArrayList<>();

Map<Integer, String> userMap = new HashMap<>();

names.add("Alice");

names.add("Bob");

userMap.put(1, "Alice");

userMap.put(2, "Bob");

// 遍历 List

for (String name : names) {

System.out.println(name);

}

// 遍历 Map

for (Map.Entry<Integer, String> entry : userMap.entrySet()) {

System.out.println("ID: " + entry.getKey() + ", Name: " + entry.getValue());

}

String first = names.get(0); // ArrayList 支持索引访问

String name = userMap.get(1); // 获取键为1的值

List<int> invalidList = new ArrayList<>(); // 编译错误！

List<Integer> validList = new ArrayList<>(); // 正确

public class Pair<K, V> {

private K key;

private V value;

public Pair(K key, V value) {

this.key = key;

this.value = value;

}

// Getters and Setters...

}

// 使用自定义泛型类

Pair<String, Integer> pair = new Pair<>("Age", 25);