Lambda

```
interface Function<T, R> {
    R apply(T t);
    Function<T, R> compose(Function<> func);
}

interface Predicate<T> {
    boolean test(T t);
}

interface Consumer<T> {
    void accept(T t);
}

interface Supplier<T> {
    T get();
}
```

Stream

```
// 创建方式
List<String> strings = Arrays.asList("Hello", "World", "小黑");
Stream<String> stream = strings.stream();

Stream<String> stream = Stream.of("Hello", "World", "小黑");

// 筛选（Filter）：对Stream中的元素进行条件筛选。
```

```
List<String> filtered = stream.filter(s -> s.contains("小")) // fileter(Predicate<T> pre);

.collect(Collectors.toList());
System.out.println(filtered); // 输出包含“小”的字符串

// 映射（Map）：将Stream中的每一个元素映射成另外的形式。
List<Integer> lengths = stream.map(String::length)
.collect(Collectors.toList()); // map(Function<T, R> func);
System.out.println(lengths); // 输出每个字符串的长度

// 收集（Collect）：是一个终端操作，它可以将Stream转换成其他形式，比如一个List或者一个Set。
List<String> list = stream.collect(Collectors.toList());

// 扁平化映射（flatMap）
List<List<String>> listOflists = Arrays.asList(
    Arrays.asList("Hello", "World"),
    Arrays.asList("小黑", "在此")
);
List<String> allUpperCase = listOflists.stream()
    .flatMap(Collection::stream)
    .map(String::toUpperCase)
    .collect(Collectors.toList());
```

```java
System.out.println(allUpperCase); // 输出所有字符串转换成大写后的结果

// 数据汇总（reduce）
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);
int sum = numbers.stream()
    .reduce(0, (a, b) -> a + b);
System.out.println(sum); // 输出 15
// 这里，reduce的第一个参数是初始值，第二个参数是一个二元操作，用来定义如何合并两个元素。

int parallelSum = numbers.parallelStream()
    .reduce(0, Integer::sum);
System.out.println(parallelSum); // 输出与上面相同的结果，但是可能通过并行处理更快完成



// 案例一：批量处理数据
List<Product> products = productRepository.findAll(); // 从数据库获取商品列表
products.stream()
    .filter(product -> product.getCategory().equals("Books")) // 只选择书籍类商品
    .map(product -> {
        product.setPrice(product.getPrice() * 0.9); // 对书籍类商品打9折
        return product;
    })
```

```
51      .forEach(productRepository::save); // 保存修改后的商品信息回数据库
52  // forEach(Consumer<T> con)
```

Arrays

```
1  Arrays.asList(Object[] obj);
2  Arrays.stream(Object[] obj);
3  String str = Arrays.toString(arr); // Arrays类的toString()方法能将数组中的内容全部打印出来
4  System.out.print(str); //输出：[4, 4, 4, 4, 4]
5
6  1.数字排序
7      int[] intArray = new int[] { 4, 1, 3, -23 };
8      Arrays.sort(intArray);
9      //输出：  [-23, 1, 3, 4]
10
11 2.字符串排序，先大写后小写
12     String[] strArray = new String[] { “z”, “a”, “C” };
13     Arrays.sort(strArray);
14     //输出：  [C, a, z]
15
16 3.严格按字母表顺序排序，也就是忽略大小写排序 Case-insensitive sort
17     Arrays.sort(strArray, String.CASE_INSENSITIVE_ORDER);
18     //输出：  [a, C, z]
19
20 4.反向排序，  Reverse-order sort
```

```
    Arrays.sort(strArray, Collections.reverseOrder());
    //输出：[z, a, C]

5.忽略大小写反向排序 Case-insensitive reverse-order sort
    Arrays.sort(strArray, String.CASE_INSENSITIVE_ORDER);

 Collections.reverse(Arrays.asList(strArray));
    //输出：  [z, C, a]

6.选择数组指定位置进行排序
    int[] arr = {3,2,1,5,4};
    Arrays.sort(arr,0,3);//给第0位（0开始）到第3位（不包括）排序
    String str = Arrays.toString(arr); //Arrays类的toString()方法能将数组中的内容全部打印出来
    System.out.print(str);
    //输出：[1, 2, 3, 5, 4]

7.将数组中的内容全部打印出来
    int[] arr = {3,2,1,5,4};
    System.out.print(arr);//直接将数组打印输出
    //输出：[I@7852e922 (数组的地址)

    String str = Arrays.toString(arr); //Arrays类的toString()方法能将数组中的内容全部打印出来
    //System.out.print(str);
    //输出：[3, 2, 1, 5, 4]
```

```
44
45  8.比较数组元素是否相等
46      int[] arr1 = {1,2,3};
47      int[] arr2 = {1,2,3};
48
    System.out.println(Arrays.equals(arr1,arr2))
    ;
49      //输出：true
50      //如果是arr1.equals(arr2),则返回false，因为
    equals比较的是两个对象的地址，不是里面的数，而
    Arrays.equals重写了equals，所以，这里能比较元素是
    否相等。
51
52  9.二分查找法找指定元素的索引值（下标）
53      // 数组一定是排好序的，否则会出错。找到元素，只
    会返回最后一个位置
54      int[] arr = {10,20,30,40,50};
55
    System.out.println(Arrays.binarySearch(arr,
    30));
56      //输出：2 （下标索引值从0开始）
57
58      int[] arr = {10,20,30,40,50};
59
    System.out.println(Arrays.binarySearch(arr,
    36));
60      //输出：-4 （找不到元素，返回-x，从-1开始数，如
    题，返回-4）
61
62      int []arr = {10,20,30,40,50};
63
    System.out.println(Arrays.binarySearch(arr,
    0,3,30));
```

```
64      //输出：2 （从0到3位（不包括）找30，找到了，在第2位，返回2）
65
66      int []arr = {10,20,30,40,50};
67
 System.out.println(Arrays.binarySearch(arr,0,3,40));
68      //输出：-4 （从0到3位（不包括）找40，找不到，从-1开始数，返回-4）
69
70  10.截取数组
71      int[] arr = {10,20,30,40,50};
72      int[] arr1 = Arrays.copyOf(arr, 3);
73      String str = Arrays.toString(arr1); //Arrays类的toString()方法能将数组中的内容全部打印出来
74      System.out.print(str);
75      //输出：[10, 20, 30] （截取arr数组的3个元素赋值给新数组arr1）
76
77      int []arr = {10,20,30,40,50};
78      int []arr1 = Arrays.copyOfRange(arr,1,3);
79      String str = Arrays.toString(arr1); //Arrays类的toString()方法能将数组中的内容全部打印出来
80      System.out.print(str);
81      //输出：[20, 30] （从第1位（0开始）截取到第3位（不包括））
82
```

Iterator

```
List<String> list = List.of("Apple", "Orange", "Pear");
for (Iterator<String> it = list.iterator(); it.hasNext(); ) {
     String s = it.next();
     System.out.println(s);
}
```

队列

```
Queue<String> queue = new LinkedList<>();
queue.offer("hello"); //入队
queue.poll(); //出对
queue.peek(); //访问元素

queue.size();
queue.isEmpty();
queue.clear();
```

栈

```
Deque<String> deque = new LinkedList<>();
deque.push("hello"); //入栈
deque.pop(); //出栈
deque.peek(); //访问元素

deque.size();
deque.isEmpty();
deque.clear();
```

List

```
Iterator<String> iterator = list.iterator();
while (iterator.hasNext()) {
    System.out.println(iterator.next());
}

// add(E e) - 将指定的元素添加到此列表的末尾。
// addAll(int index, Collection<? extends E> c) - 从指定位置开始将指定集合中的所有元素按其顺序添加到列表中。
// remove(Object o) - 移除列表中首次出现的指定元素。
// remove(int index) - 移除指定索引处的元素。
// set(int index, E element) - 修改指定位置上的元素。
// get(int index) - 返回指定位置的元素。
// contains(Object o) - 判断列表是否包含指定元素。
```

```
// containsAll(Collection<?> c) - 判断列表是否包含指定集合中的所有元素。
// indexOf(Object o) - 返回指定元素在此列表中首次出现的位置（索引），如果列表不包含此元素，则返回 -1。
// lastIndexOf(Object o) - 返回指定元素在此列表中最后一次出现的位置（索引），如果列表不包含此元素，则返回 -1。
// subList(int fromIndex, int toIndex) - 返回从 fromIndex 开始到 toIndex 结束（不包括 toIndex）的一个视图。
// sort(Comparator<? super E> c) - 根据提供的 Comparator 对列表元素进行排序。

// size() - 返回列表中的元素数量。
// isEmpty() - 判断列表是否为空。
// clear() - 清空列表。
```

Set

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

        // 添加元素
        set.add("apple");
        set.add("banana");

        // 检查元素是否存在

 System.out.println(set.contains("banana"));
// 输出 true

```

```
        // 遍历集合
        for (String fruit : set) {
            System.out.println(fruit);
        }

        // 删除元素
        set.remove("apple");

        // 清空集合
        set.clear();
    }
}

// 集合操作
// addAll(Collection<? extends E> c) - 将指定集合中的所有元素添加到此集合中。
// removeAll(Collection<?> c) - 移除此集合中包含的所有指定元素。
// containsAll(Collection<?> c) - 如果此集合包含指定集合中的所有元素，则返回 true。

// size() - 返回集合中的元素个数。
// isEmpty() - 如果此集合不包含任何元素，则返回 true。
```

Map

```
// 遍历
public class Main {
    public static void main(String[] args) {
        Map<String, Integer> map = new HashMap<>();
```

```
        map.put("apple", 123);
        map.put("pear", 456);
        map.put("banana", 789);
        for (Map.Entry<String, Integer>
entry : map.entrySet()) {
            String key = entry.getKey();
            Integer value =
entry.getValue();
            System.out.println(key + " = " +
value);
        }
    }
}

// put(K key, V value) - 将指定的键值对添加到此映射中。
// get(Object key) - 返回指定键对应的值；如果不存在，则返回 null。
// getOrDefault(Object key, V defaultValue) - 返回指定键对应的值；如果不存在，则返回默认值。
// remove(Object key) - 移除指定键的映射关系。
// remove(Object key, Object value) - 如果给定键对应的值等于给定值，则移除此映射。
// replace(K key, V value) - 替换指定键的值。
// replace(K key, V oldValue, V newValue) - 当且仅当指定键映射到给定旧值时，替换该键的新值。
// containsKey(Object key) - 如果此映射包含指定键的映射关系，则返回 true。
// containsValue(Object value) - 如果此映射包含至少一个指定值的映射关系，则返回 true。
// keySet() - 返回此映射中包含的所有键的集合视图。
// values() - 返回此映射中包含的所有值的集合视图。
```

```
27 // entrySet() - 返回此映射中存在的键值对映射关系的集合视
28
29 // size() - 返回此映射中的键值对数。图。
30 // isEmpty() - 如果此映射不包含任何键值对，则返回true。
31 // clear() - 移除此映射中的所有映射关系。
```

> PriorityQueue、TreeSet、TreeMap 默认由小到大升序排序。
>
> @Override
>
> public int compare(Integer o1, Integer o2) {
>
> *// TODO Auto-generated method stub*
>
> return o1 - o2; // 升序
>
> return o2 -o1; // 降序
>
> return o1 - o2 > 0 ? 1 : -1; // 升序
>
> return o1 - o2 > 0 ? -1 : 1; // 降序
>
> }

元素类型	排序方式
数值类型	按数值大小排序
Character类型	按字符的Unicode值来比较
Boolean类型	true大于false

元素类型	排序方式
String类型	按字符串中字符的unicode值进行比较
Date、Time类型	后边的时间、日期比前面的时间、日期大

PriorityQueue

```
public class Main {
    public static void main(String[] args) {
        Queue<User> q = new PriorityQueue<>((u1, u2) -> {
            if (u1.number.charAt(0) == u2.number.charAt(0)) {
                return u1.number.compareTo(u2.number);
            }
            if (u1.number.charAt(0) == "V") {
                return -1;
            } else {
                return 1;
            }
        });
        // 添加3个元素到队列:
        q.offer(new User("Bob", "A1"));
        q.offer(new User("Alice", "A2"));
        q.offer(new User("Boss", "V1"));
        System.out.println(q.poll()); // Boss/V1
```

```
        System.out.println(q.poll()); // Bob/A1
        System.out.println(q.poll()); // Alice/A2
        System.out.println(q.poll()); // null,因为队列为空
    }
}

class User {
    public final String name;
    public final String number;

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

    public String toString() {
        return name + "/" + number;
    }
}
```

LinkedHashSet

```
// LinkedHashSet 允许我们按照插入的顺序迭代元素。当使用迭代器循环LinkedHashSet时，元素将按照插入的顺序返回。

class GFG {
```

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

        Set<String> hs = new
LinkedHashSet<String>();
        // using add() method
        hs.add("Geek");
        hs.add("For");
        hs.add("Geeks");
        hs.add("A");
        hs.add("B");
        hs.add("Z");

        // using iterators
        Iterator itr = hs.iterator();
        while (itr.hasNext())
            System.out.print(itr.next() + ",
");
        System.out.println();

        // Using enhanced for loop for
iteration
        for (String s : hs)
            System.out.print(s + ", ");
        System.out.println();
    }
}

// 输出
// Geek, For, Geeks, A, B, Z,
// Geek, For, Geeks, A, B, Z,
```

TreeSet

```
public class Testdemo {
    public static void main(String[] args) {
        Set<Integer> t = new TreeSet<Integer>(new Comparator<Integer>() {
            @Override
            public int compare(Integer o1, Integer o2) {
                // TODO Auto-generated method stub
                return o2 - o1;
            }
        });
        t.add(55);
        t.add(45);
        t.add(50);
        Iterator<Integer> iterator = t.iterator();
        while(iterator.hasNext()){

System.out.println(iterator.next());
        }
    }
}

// 输出结果：[55,50,45]
```

LinkedHashMap

```
// 同 LinkedHashSet
```

TreeMap

```
1  public class Main {
2      public static void main(String[] args) {
3          Map<Person, Integer> map = new
   TreeMap<>(new Comparator<Person>() {
4              public int compare(Person p1,
   Person p2) {
5                  return
   p1.name.compareTo(p2.name);
6              }
7          });
8          map.put(new Person("Tom"), 1);
9          map.put(new Person("Bob"), 2);
10         map.put(new Person("Lily"), 3);
11         for (Person key : map.keySet()) {
12             System.out.println(key);
13         }
14         // {Person: Bob}, {Person: Lily},
   {Person: Tom}
15         System.out.println(map.get(new
   Person("Bob"))); // 2
16     }
17 }
18
19 class Person {
20     public String name;
21     Person(String name) {
22         this.name = name;
23     }
24     public String toString() {
25         return "{Person: " + name + "}";
```

```
    }
}

public class Main {
    public static void main(String[] args) {
        Map<Student, Integer> map = new
TreeMap<>(new Comparator<Student>() {
            public int compare(Student p1,
Student p2) {
                return p1.score > p2.score ?
-1 : 1;
            }
        });
        map.put(new Student("Tom", 77), 1);
        map.put(new Student("Bob", 66), 2);
        map.put(new Student("Lily", 99), 3);
        for (Student key : map.keySet()) {
            System.out.println(key);
        }
        System.out.println(map.get(new
Student("Bob", 66))); // null?
    }
}

class Student {
    public String name;
    public int score;
    Student(String name, int score) {
        this.name = name;
        this.score = score;
    }
    public String toString() {
```

```
54          return String.format("{%s:
    score=%d}", name, score);
55      }
56  }
```

1. 给一个字符串，给出出现最多次数的字符个数，注意不区分大小写

```
1  "AaA11" -> 3
2  "abcab" -> 2
```

2. 有一句话比如 "i am sorry"，它会在这句话前面和后面随机添加同一串或几串字符 "wub"，并且空格也会被替换成几个 "wub"。你需要将替换后的结果，还原成原来的句子。

```
1  public String temp(String str) {
2      String replaced = str.replace("wub", "
   ");
3      String[] words = replaced.split("\\s+");
4      StringBuilder result = new
   StringBuilder();
5      for (String word : words) {
6          if (!word.isEmpty()) {
7              if (result.length() > 0) {
8                  result.append(" ");
9              }
10             result.append(word);
11         }
12     }
```

```
13      return result.toString().trim();
14  }
```

3. springboot + mybatis 实战题。Get 方法但是没有输入，改成 Get 方法可以输入三个参数，并且三个参数都可以为空。

```
1
```

1. 输入字符串aba,依次输出各个字符

```
1  public static void main(String[] args) {
2      Scanner sc = new Scanner(System.in);
3      String str = sc.next();
4      System.out.println("输入的字符串是: " +
   str);
5      for (char c : str.toCharArray()) {
6          System.out.println(c);
7      }
8  }
```

2. 字符串反转

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    String str = sc.next();
    System.out.println("输入的字符串是：" +
str);

    StringBuilder sb = new
StringBuilder(str);
    String reversed =
sb.reverse().toString();
    System.out.println("反转后的字符串是：" +
reversed);
}
```

3. 统计字符串次数

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    String str = sc.next();
    System.out.println("输入的字符串是：" +
str);

    int lowCon = 0, upCon = 0, numCon = 0;
    for (char c : str.toCharArray()) {
        if (c >= 'a' && c <= 'z') {
            lowCon++;
        } else if (c >= 'A' && c <= "Z") {
            upCon++;
        } else if (c >= '0' && c <= '9') {
            numCon++;
```

```
        }
    }
    System.out.println("大写字母有：" +
upCon);
    System.out.println("小写字母有：" +
lowCon);
    System.out.println("数字有：" + numCon);
}
```

4. 拼接字符串

```
public static void main(String[] args) {
    int[] arr= {1,3,5};
    if (arr == null) {
        return " ";
    } else if (arr.length() == 0) {
        return "[]";
    } else {
        StringBuilder str = new
StringBuilder("[");
        str.append(arr[0]);
        for(int i = 1; i < arr.length();
i++){
            str.append(", ").append(arr[i]);
        }
        str.append("]");
        return str.toString();
    }
}
```

5. 金额转换

```
1
```

6. 转换罗马数字

```
1  public static void main(String[] args) {
2      Scanner sc = new Scanner(System.in);
3      String str = "";
4      while(true) {
5          str = sc.next();
6          boolean flag = isValid(str);
7          if (flag) {
8              System.out.println("输入正确");
9              break;
10         } else {
11             System.out.println("输入错误，请重
   新输入");
12             continue;
13         }
14     }
15
16     StringBuilder sb = new StringBuilder();
17     for (char c : str.toCharArray()) {
18         sb.append(parse(c)).append(" ");
19     }
20
    System.out.println(sb.toString().trim(););
21 }
22
23 public static boolean isVaild(String str) {
```

```
    if (str.length() > 9) {
        return false;
    }
    for (char c : str.toCharArray()) {
        if (c < '0' || c > '9') {
            return false;
        }
    }
    return true;
}

public static String parse(char num) {
    String str = "";
    switch (num) {
        case '0' -> str = " ";
        case '1' -> str = "I";
        case '2' -> str = "II";
        case '3' -> str = "III";
        case '4' -> str = "IV";
        case '5' -> str = "V";
        case '6' -> str = "VI";
        case '7' -> str = "VII";
        case '8' -> str = "VX";
    }
    return str;
}
```

7. 输入任意字符串，打乱其顺序

```
public static void main(String[] args) {
    String str = "1234546";
    // 将字符串转换为字符数组
    char[] arr = str.toCharArray();
```

```
    // 打乱顺序
    arr = mix(arr);
    // 将字符数组转换为字符串
    String shuffledString = new String(arr);
    System.out.println(shuffledString);
    }


public static char[] mix(char[] arr) {
    Random rand = new Random();
    for (int i = arr.length() - 1; i > 0; i--) {
        int index = rand(i + 1);
        char temp = arr[index];
        arr[index] = arr[i];
        arr[i] = temp;
    }
    return arr;
}
```

8. 生成验证码

```
public static void main(String[] args) {
    Random rand = new Random();
    char[] arr = new char[52];
    arr = capital(arr);
    StringBuilder str = new StringBuilder();
    for (int i = 0; i < 4; i++) {
        int index = rand.nextInt(arr.length);
        str.append(arr[index]);
    }
    int num = rand.nextInt(10);
```

```
    int index = rand.nextInt(5);
    str.insert(index, num + "");
    System.out.println("随机访问的5个字符：" +
str.toString().trim());
}

// 将所有大写小写字符都存放在一个数组中
public static char[] capital(char[] arr) {
    int index = 0;
    for (char i = 'a'; i <= 'z'; i++) {
        arr[index++] = i;
    }
    for (char i = 'A'; i <= 'Z'; i++) {
        arr[index++] = i;
    }
    return arr;
}
```

9. 将字符串转换为数字

```
public static int parseArr(String str) {
    int i = 1;
    int sum = 0;
    for (char c : str.toCharArray()) {
        int num = (int) Math.pow(10,
str.length() - i);
        sum += (c - '0') * num;
        i++;
    }
    return sum;
}
```