**1. Java自带工具方法**

**1.1 List集合拼接成以逗号分隔的字符串**

*// 如何把list集合拼接成以逗号分隔的字符串 a,b,c*

List<String> list = Arrays.asList("a", "b", "c");

*// 第一种方法，可以用stream流*

String join = list.stream().collect(Collectors.joining(","));

System.out.println(join); *// 输出 a,b,c*

*// 第二种方法，其实String也有join方法可以实现这个功能*

String join = String.join(",", list);

System.out.println(join); *// 输出 a,b,c*

**1.2 比较两个字符串是否相等，忽略大小写**

**if** (strA.equalsIgnoreCase(strB)) {

  System.out.println("相等");

}

**1.3 比较两个对象是否相等**

当我们用equals比较两个对象是否相等的时候，还需要对左边的对象进行判空，不然可能会报空指针异常，我们可以用java.util包下Objects封装好的比较是否相等的方法

Objects.equals(strA, strB);

源码是这样的

**public** **static** **boolean** **equals**(Object a, Object b) {

    **return** (a == b) || (a != **null** && a.equals(b));

}

**1.4 两个List集合取交集**

List<String> list1 = **new** ArrayList<>();

list1.add("a");

list1.add("b");

list1.add("c");

List<String> list2 = **new** ArrayList<>();

list2.add("a");

list2.add("b");

list2.add("d");

list1.retainAll(list2);

System.out.println(list1); *// 输出[a, b]*

**2. apache commons工具类库**

apache commons是最强大的，也是使用最广泛的工具类库，里面的子库非常多，下面介绍几个最常用的

**2.1 commons-lang，java.lang的增强版**

建议使用commons-lang3，优化了一些api，原来的commons-lang已停止更新

Maven依赖是：

<dependency>

    <groupId>org.apache.commons</groupId>

    <artifactId>commons-lang3</artifactId>

    <version>3.12.0</version>

</dependency>

**2.1.1 字符串判空**

传参CharSequence类型是String、StringBuilder、StringBuffer的父类，都可以直接下面方法判空，以下是源码：

**public** **static** **boolean** **isEmpty**(**final** CharSequence cs) {

    **return** cs == **null** || cs.length() == 0;

}

**public** **static** **boolean** **isNotEmpty**(**final** CharSequence cs) {

    **return** !isEmpty(cs);

}

*// 判空的时候，会去除字符串中的空白字符，比如空格、换行、制表符*

**public** **static** **boolean** **isBlank**(**final** CharSequence cs) {

    **final** **int** strLen = length(cs);

    **if** (strLen == 0) {

        **return** **true**;

    }

    **for** (**int** i = 0; i < strLen; i++) {

        **if** (!Character.isWhitespace(cs.charAt(i))) {

            **return** **false**;

        }

    }

    **return** **true**;

}

**public** **static** **boolean** **isNotBlank**(**final** CharSequence cs) {

    **return** !isBlank(cs);

}

**2.1.2 首字母转成大写**

String str = "yideng";

String capitalize = StringUtils.capitalize(str);

System.out.println(capitalize); // 输出Yideng

**2.1.3 重复拼接字符串**

String str = StringUtils.repeat("ab", 2);

System.out.println(str); // 输出abab

**2.1.4 格式化日期**

再也不用手写SimpleDateFormat格式化了

*// Date类型转String类型*

String date = DateFormatUtils.format(**new** Date(), "yyyy-MM-dd HH:mm:ss");

System.out.println(date); *// 输出 2021-05-01 01:01:01*

*// String类型转Date类型*

Date date = DateUtils.parseDate("2021-05-01 01:01:01", "yyyy-MM-dd HH:mm:ss");

*// 计算一个小时后的日期*

Date date = DateUtils.addHours(**new** Date(), 1);

**2.1.5 包装临时对象**

当一个方法需要返回两个及以上字段时，我们一般会封装成一个临时对象返回，现在有了Pair和Triple就不需要了

*// 返回两个字段*

ImmutablePair<Integer, String> pair = ImmutablePair.of(1, "yideng");

System.out.println(pair.getLeft() + "," + pair.getRight()); *// 输出 1,yideng*

*// 返回三个字段*

ImmutableTriple<Integer, String, Date> triple = ImmutableTriple.of(1, "yideng", **new** Date());

System.out.println(triple.getLeft() + "," + triple.getMiddle() + "," + triple.getRight()); *// 输出 1,yideng,Wed Apr 07 23:30:00 CST 2021*

**2.2 commons-collections 集合工具类**

Maven依赖是：

<dependency>

    <groupId>org.apache.commons</groupId>

    <artifactId>commons-collections4</artifactId>

    <version>4.4</version>

</dependency>

**2.2.1 集合判空**

封装了集合判空的方法，以下是源码：

**public** **static** **boolean** **isEmpty**(**final** Collection<?> coll) {

    **return** coll == **null** || coll.isEmpty();

}

**public** **static** **boolean** **isNotEmpty**(**final** Collection<?> coll) {

    **return** !isEmpty(coll);

}

*// 两个集合取交集*

Collection<String> collection = CollectionUtils.retainAll(listA, listB);

*// 两个集合取并集*

Collection<String> collection = CollectionUtils.union(listA, listB);

*// 两个集合取差集*

Collection<String> collection = CollectionUtils.subtract(listA, listB);

**2.3 common-beanutils 操作对象**

Maven依赖：

<dependency>

    <groupId>commons-beanutils</groupId>

    <artifactId>commons-beanutils</artifactId>

    <version>1.9.4</version>

</dependency>

**public** **class** **User** {

    **private** Integer id;

    **private** String name;

}

设置对象属性

User user = **new** User();

BeanUtils.setProperty(user, "id", 1);

BeanUtils.setProperty(user, "name", "yideng");

System.out.println(BeanUtils.getProperty(user, "name")); *// 输出 yideng*

System.out.println(user); *// 输出 {"id":1,"name":"yideng"}*

对象和map互转

*// 对象转map*

Map<String, String> map = BeanUtils.describe(user);

System.out.println(map); *// 输出 {"id":"1","name":"yideng"}*

*// map转对象*

User newUser = **new** User();

BeanUtils.populate(newUser, map);

System.out.println(newUser); *// 输出 {"id":1,"name":"yideng"}*

**2.4 commons-io 文件流处理**

Maven依赖：

<dependency>

    <groupId>commons-io</groupId>

    <artifactId>commons-io</artifactId>

    <version>2.8.0</version>

</dependency>

文件处理

File file = **new** File("demo1.txt");

*// 读取文件*

List<String> lines = FileUtils.readLines(file, Charset.defaultCharset());

*// 写入文件*

FileUtils.writeLines(**new** File("demo2.txt"), lines);

*// 复制文件*

FileUtils.copyFile(srcFile, destFile);

**3. Google Guava 工具类库**

Maven依赖：

<dependency>

    <groupId>com.google.guava</groupId>

    <artifactId>guava</artifactId>

    <version>30.1.1-jre</version>

</dependency>

**3.1 创建集合**

List<String> list = Lists.newArrayList();

List<Integer> list = Lists.newArrayList(1, 2, 3);

*// 反转list*

List<Integer> reverse = Lists.reverse(list);

System.out.println(reverse); *// 输出 [3, 2, 1]*

*// list集合元素太多，可以分成若干个集合，每个集合10个元素*

List<List<Integer>> partition = Lists.partition(list, 10);

Map<String, String> map = Maps.newHashMap();

Set<String> set = Sets.newHashSet();

**3.2 黑科技集合**

**3.2.1 Multimap 一个key可以映射多个value的HashMap**

Multimap<String, Integer> map = ArrayListMultimap.create();

map.put("key", 1);

map.put("key", 2);

Collection<Integer> values = map.get("key");

System.out.println(map); *// 输出 {"key":[1,2]}*

*// 还能返回你以前使用的臃肿的Map*

Map<String, Collection<Integer>> collectionMap = map.asMap();

多省事，多简洁，省得你再创建 Map<String, List<Integer>>

**3.2.1 BiMap 一种连value也不能重复的HashMap**

BiMap<String, String> biMap = HashBiMap.create();

*// 如果value重复，put方法会抛异常，除非用forcePut方法*

biMap.put("key","value");

System.out.println(biMap); *// 输出 {"key":"value"}*

*// 既然value不能重复，何不实现个翻转key/value的方法，已经有了*

BiMap<String, String> inverse = biMap.inverse();

System.out.println(inverse); *// 输出 {"value":"key"}*

这其实是双向映射，在某些场景还是很实用的。

**3.2.3 Table 一种有两个key的HashMap**

*// 一批用户，同时按年龄和性别分组*

Table<Integer, String, String> table = HashBasedTable.create();

table.put(18, "男", "yideng");

table.put(18, "女", "Lily");

System.out.println(table.get(18, "男")); *// 输出 yideng*

*// 这其实是一个二维的Map，可以查看行数据*

Map<String, String> row = table.row(18);

System.out.println(row); *// 输出 {"男":"yideng","女":"Lily"}*

*// 查看列数据*

Map<Integer, String> column = table.column("男");

System.out.println(column); *// 输出 {18:"yideng"}*

**3.2.4 Multiset 一种用来计数的Set**

Multiset<String> multiset = HashMultiset.create();

multiset.add("apple");

multiset.add("apple");

multiset.add("orange");

System.out.println(multiset.count("apple")); *// 输出 2*

*// 查看去重的元素*

Set<String> set = multiset.elementSet();

System.out.println(set); *// 输出 ["orange","apple"]*

*// 还能查看没有去重的元素*

Iterator<String> iterator = multiset.iterator();

**while** (iterator.hasNext()) {

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

}

*// 还能手动设置某个元素出现的次数*

multiset.setCount("apple", 5);