Stream

判断是中间操作还是最终操作：

查看方法有没有返回值

map映射：

```
List<String> list = Arrays.asList("MySQL", "Java", "HTML", "CSS", "JavaScript");
list.stream().map(String::length).forEach(System.out::println);
list.stream().mapToInt(String::length).forEach(System.out::println);
list.stream().mapToDouble(x -> x.codePointAt(0)).forEach(System.out::println);
//list.stream().mapToLong()
```

flatmap和map的区别

flatmap会把每个元素映射成新的流

map会把每个元素映射成新的元素

```
list.stream().flatMap(x -> Stream.of(x.split(""))).forEach(System.out::println);
list.stream().flatMapToInt()
list.stream().flatMapToDouble()
list.stream().flatMapToLong()
```

测试类Girl：

```
public class Girl implements Comparable<Girl> {
 private String name;
 private String gender;
 private int age;
 private Dog dog;

 public Girl() {
 }

  public Girl(String name, String gender, int age) {
  this.name = name;
  this.gender = gender;
  this.age = age;
  }

  public String getName() {
```

```
17  return name;
18  }
19
20  public void setName(String name) {
21  this.name = name;
22  }
23
24  public String getGender() {
25  return gender;
26  }
27
28  public void setGender(String gender) {
29  this.gender = gender;
30  }
31
32  public int getAge() {
33  return age;
34  }
35
36  public void setAge(int age) {
37  this.age = age;
38  }
39
40  public Dog getDog() {
41  return dog;
42  }
43
44  public void setDog(Dog dog) {
45  this.dog = dog;
46  }
47
48  @Override
49  public String toString() {
50  return "com.cqw.Girl{" +
51  "name='" + name + '\'' +
52  ", gender='" + gender + '\'' +
53  ", age=" + age +
54  '}';
55  }
56
57  //实现comparable接口后重写排序规则
```

```
@Override
public int compareTo(Girl o) {
if (this.getAge() > o.getAge()){
return -1;
}else if (this.getAge() < o.getAge()){
return 1;
}else {
return 0;
}
}
//去重时重写的对比方法
@Override
public boolean equals(Object o) {
if (this == o) {
return true;
}
if (o == null || getClass() != o.getClass()) {
return false;
}
Girl girl = (Girl) o;
return age == girl.age &&
Objects.equals(name, girl.name) &&
Objects.equals(gender, girl.gender);
}
//去重时重写的对比方法
@Override
public int hashCode() {
return Objects.hash(name, gender, age);
}


public void eat(String thing){
if(thing != null) {
System.out.println("吃" + thing);
}
}


public void eatPro(String thing){
//把thing用Optional容器包装起来
```

```
98  // Optional<String> thing1 = Optional.ofNullable(thing);
99  //thing1.ifPresent(x -> System.out.println("吃"+x));
100  Optional.ofNullable(thing).ifPresent(x -> System.out.println("吃"+x));
101  }
102
103  //获取某个女孩的狗名
104  //过滤参数是否为空
105  private static String dogName(Girl girl){
106  if(girl != null){
107  Dog dog = girl.getDog();
108  if (dog != null){
109  String name = dog.getName();
110  if (name != null){
111  return name;
112  }
113  }
114  }
115  return null;
116  }
117  //过滤参数是否为空
118  private static String dogNamePro(Girl girl){
119  return Optional.ofNullable(girl)
120  .map(g -> g.getDog())
121  .map(d -> d.getName())
122  .orElse(null);
123  }
124 }
```

filter过滤：

```
1  ArrayList<Girl> arrayList = new ArrayList<>();
2  Collections.addAll(arrayList,new Girl("如花","女",16),new Girl("大美","女",36),new Girl("杉菜","女",66));
3  arrayList.stream().filter(x -> x.getAge() <= 20).forEach(System.out::println);
4  arrayList.stream().filter(x -> x.getAge() <= 20).limit(1).forEach(System.out::println);
```

distinct去重：

需要在类中重写equals和hashCode方法

```
1  ArrayList<Girl> arrayList = new ArrayList<>();
```

```
Collections.addAll(arrayList,new Girl("如花","女",16),new Girl("如花","女",16),new Girl("大美","女",36),new Girl("大美","女",36),new Girl("杉菜","女",66));
arrayList.stream().distinct().forEach(System.out::println);
```

sorted排序：

可以选择在类中重写排序方法

```
ArrayList<Girl> arrayList = new ArrayList<>();
Collections.addAll(arrayList,new Girl("如花","女",36),new Girl("大美","女",16),new Girl("杉菜","女",66));
//不重写排序方法
arrayList.stream().sorted((x1,x2) ->{
 if (x1.getAge() > x2.getAge()){
 return -1;
 }else if (x1.getAge() < x2.getAge()){
 return 1;
 }else{
  return 0;
  }
 }).forEach(System.out::println);
 //不重写排序方法的lambda格式
 arrayList.stream().sorted((x1,x2) -> -Integer.compare(x1.getAge(), x2.getAge())).forEach(System.out::println);
 //重写排序方法
 arrayList.stream().sorted().forEach(System.out::println);
```

limit限制：

```
Integer[] numbers ={12, 21, 9, 4, 30};
Arrays.stream(numbers).limit(3).forEach(System.out::println);
//获取三个整型随机数
Random random = new Random();
random.ints().limit(3).forEach(System.out::println);
//random.doubles()
//random.longs()

//在指定范围内获取随机数
 random.ints(10,20).limit(5).forEach(System.out::println);
```

peek在遍历执行前所执行的操作：

```
List<String> list = Arrays.asList("a1", "a2", "a3");
```

```
2 list.stream().peek(x -
>System.out.println("? ? ? ? ")).forEach(System.out::println);
```

max最大最小值：

```
1  List<Integer> list = Arrays.asList(22,33);
2  //Optional是个容器，存放了结果数据
3  Optional<Integer> max = list.stream().max(Integer::compareTo);
4 // 判断是否为空
5  max.ifPresent(System.out::println);
6  Integer integer = max.get();
7  System.out.println(integer);
```

collect接收stream处理过的数据

```
1 List<Integer> list = Arrays.asList(11,22,33);
2 list.stream().map(x -> x + 10).forEach(System.out::println);
3 System.out.println(list);
4 //转换为集合
5 List<Integer> list1 = list.stream().map(x -> x + 10).collect(Collectors.t
oList());
6 System.out.println(list1);
7 //转换为指定类型的集合
8 ArrayList<Integer> arraylist = list.stream().map(x -> x + 10).collect(Col
lectors.toCollection(ArrayList::new));
9 System.out.println(arraylist);
```

Optional容器

具体方法可观察Girl类