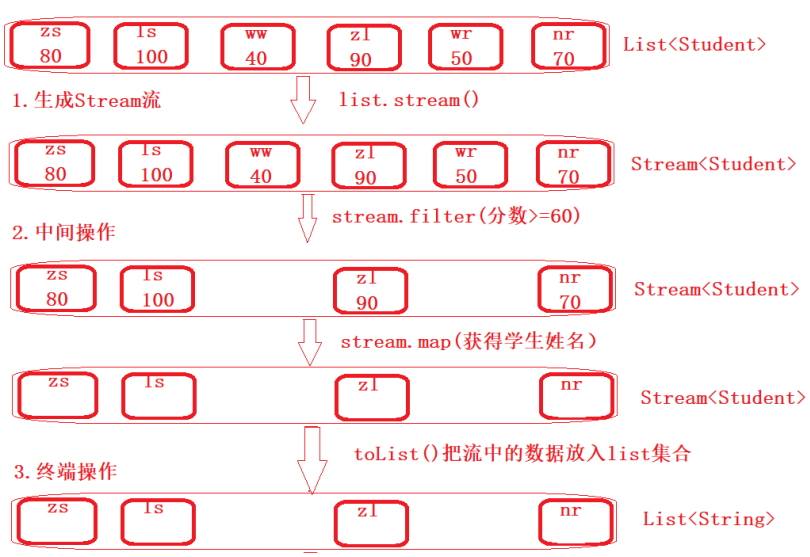
# 第二十章 Stream流

## 1. Stream流的概述

### 1.1 什么是Stream流

1. 简单的说：就是操作数组和集合的一种方式
2. Stream流基于数组和集合生成的元素序列，可以把Stream流看成一个高级的迭代器

### 1.2 Steam流的操作步骤




1. Stream是在数组或者集合等数据源上生成的元素序列
   可以把Stream看出一个高级迭代器
2. Stream操作步骤：
   1. 生成Stream
   2. 中间操作
   3. 终端操作
3. 说明：
   1. Stream流不存储数据，数据保存在数据源中，Stream强调的是对数据的操作
   2. Stream流的核心是延迟计算，只有对Stream流做终端操作，才会产生数据流动
   3. 可以把多个基本操作连接为一个流水线
   4. Stream流只要进行了终端操作，这个流就关闭了，不能再做中间操作了

### 1.3 Stream流的总结

1. 创建Stream
   - 通过数据源（数组|集合）创建一个Stream流
2. 中间操作
   - 对数据源中的数据进行处理，该操作会返回一个Stream流对象，因此可以使用链式调用
3. 终端操作
   - 执行终端操作，才会真正的执行中间操作，返回一个操作后的结果集

## 2. 创建Stream流

### 2.1 创建Stream（三种方式）

代码块

```
package com.powernode.stream01;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Stream;

public class Test01 {
    public static void main(String[] args) {
        //1.通过集合创建Stream流
        List<String> list = new ArrayList<>();
        Stream<String> stream = list.stream();
        //2.通过数组创建Stream流
        String[] str = {"aa", "bb", "cc"};
        Stream<String> stream1 = Arrays.stream(str);
        //3.通过静态方法创建Stream流
        Stream<String> aa = Stream.of("aa", "bb", "cc");
    }
}
```

## 2.2 体验一下Stream流

代码块

```
package com.powernode.stream01;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.function.Consumer;
import java.util.function.Predicate;
import java.util.stream.Stream;

public class Test02 {
    public static void main(String[] args) {
        List<Integer> list = Arrays.asList(60, 80, 30, 90, 50);
        //1.常规写法：求及格的分数
        for (Integer score : list) {
            if (score >= 60) {
                System.out.println(score);
            }
        }
        System.out.println("==============");
        //2.Stream流写法
        //2.1创建Stream流
        Stream<Integer> stream = list.stream();
```

```
23          //2.2中间操作
24          Stream<Integer> integerStream = stream.filter(new Predicate<Integer>()
    {
25              @Override
26              public boolean test(Integer score) {
27                  return score >= 60;//把符合条件的保留下来
28              }
29          });
30          //2.3终端操作
31          integerStream.forEach(new Consumer<Integer>() {
32              @Override
33              public void accept(Integer score) {
34                  System.out.println(score);
35              }
36          });
37          System.out.println("==============");
38         /* stream.filter(new Predicate<Integer>() {
39              @Override
40              public boolean test(Integer integer) {
41                  return false;
42              }
43          });*/  //.IllegalStateException  stream has already been operated upon
    or closed
44          //2.2中间操作
45          /*list.stream().filter(new Predicate<Integer>() {
46              @Override
47              public boolean test(Integer score) {
48                  return score >= 60;//把符合条件的保留下来
49              }
50          }).forEach(new Consumer<Integer>() {
51              @Override
52              public void accept(Integer score) {
53                  System.out.println(score);
54              }
55          });*/
56          list.stream().filter( score -> score >=
    60).forEach(System.out::println);
57      }
58  }
```

## 2.3 顺序流和并行流

### 2.3.1 顺序流和并行流概述

1. 前面讲解的三种获得流的方式，都获得是顺序流，顺序流对Stream元素的操作都是单线程，处理效率比较低

2. Stream流还可以使用多线程，使用多线程必须是并行流，提高效率
3. 顺序流可以转换为并行流

### 2.3.2 顺序流与并行流互相转换

代码块

```
package com.powernode.stream01;

import java.util.Arrays;
import java.util.List;
import java.util.function.Consumer;
import java.util.function.Predicate;
import java.util.stream.Stream;

public class Test03 {
    public static void main(String[] args) {
        //1.创建一个顺序流
        Stream<String> stream = Stream.of("aa", "bb", "cc");
        //2.判断当前流是否为并行流
        System.out.println("stream.isParallel() = " + stream.isParallel());
        //3.把顺序流转换为并行流
        Stream<String> parallel = stream.parallel();
        System.out.println("parallel.isParallel() = " + parallel.isParallel());
        //4.把并行流转换为顺序流
        Stream<String> sequential = parallel.sequential();
        System.out.println("sequential.isParallel() = " +
sequential.isParallel());

        //5.直接创建并行流
        List<String> list = Arrays.asList("aa", "bb", "cc");
        Stream<String> stream1 = list.parallelStream();
        System.out.println("stream1.isParallel() = " + stream1.isParallel());


    }
}
```

# 3. Stream流中间操作

## 3.1 中间操作的概述

1. 中间操作是对流中数据的处理，比如学生集合，拿到分数>=60，其实就是对集合中数据的处理
2. 这个筛选似乎预处理，就是规则写入，实际上并没有执行操作，这种方式称为预处理

3. Stream流的中操作返回的还是一个Stream流对象，方法链式调用
4. 数据准备，后面案例使用

代码块

```
package com.powernode.stream02;

import java.util.ArrayList;
import java.util.List;

class Student{
    private String name;
    private int age;
    private char sex;
    private String city;

    public Student(String name, int age, char sex, String city) {
        this.name = name;
        this.age = age;
        this.sex = sex;
        this.city = city;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public char getSex() {
        return sex;
    }

    public void setSex(char sex) {
        this.sex = sex;
    }

```

```
43      public String getCity() {
44          return city;
45      }
46
47      public void setCity(String city) {
48          this.city = city;
49      }
50
51      @Override
52      public String toString() {
53          return "Student{" +
54                  "name='" + name + '\'' +
55                  ", age=" + age +
56                  ", sex=" + sex +
57                  ", city='" + city + '\'' +
58                  '}';
59      }
60  }
61  public class StudentData {
62      public static List<Student> getStudentList(){
63          List<Student> list = new ArrayList<>();
64          list.add(new Student("zs", 23, '男', "北京"));
65          list.add(new Student("ls", 19, '女', "上海"));
66          list.add(new Student("ww", 26, '男', "广州"));
67          list.add(new Student("zl", 16, '女', "深圳"));
68          return list;
69      }
70  }
```

## 3.2 筛选（filter）

- 筛选：按照一定的规则，将符合规则的元素，提取到新的流中

### 3.2.1 *筛选出年龄>20的学生*

代码块

```
1   package com.powernode.stream02;
2
3   import java.util.function.Predicate;
4   import java.util.stream.Stream;
5
6   public class Test01 {
7       public static void main(String[] args) {
8           //需求1：筛选出年龄>20的学生
9           Stream<Student> stream = StudentData.getStudentList().stream();
```

```
        Stream<Student> studentStream = stream.filter(new Predicate<Student>()
{
            @Override
            public boolean test(Student student) {
                return student.getAge() > 20;//符合条件的元素，提取到流中
            }
        });
        //创建了新的流
        System.out.println(stream == studentStream);
        studentStream.forEach(System.out::println);
        //studentStream.forEach(System.out::println);

    }
}
```

### 3.2.2 筛选出名称长度大于2的字符串

代码块

```
package com.powernode.stream02;

import java.util.function.Consumer;
import java.util.function.Predicate;
import java.util.stream.Stream;

public class Test02 {
    public static void main(String[] args) {
        //需求1：筛选出名称长度大于2的字符串
        Stream<String> stream = Stream.of("张三丰", "赵敏", "周芷若", "张无忌",
"谢逊", "杨过");
        /*stream.filter(new Predicate<String>() {
            @Override
            public boolean test(String name) {
                return name.length() > 2;
            }
        }).forEach(System.out::println);*/
        stream.filter( name -> name.length() > 2).forEach(System.out::println);
    }
}
```

## 3.3 映射 （map）

### 3.3.1 将集合中数据按照一定的规则提取到新的流中

```java
package com.powernode.stream02;

import java.util.List;
import java.util.function.Function;
import java.util.function.Predicate;
import java.util.stream.Stream;

public class Test03 {
    public static void main(String[] args) {
        Stream<String> stream = Stream.of("hello", "word", "java", "too");
        //需求1：把流中字母转换为大写
        /* stream.map(new Function<String, String>() {
            @Override
            public String apply(String s) {
                return s.toUpperCase();
            }
        }).forEach(System.out::println);*/
        stream.map(String::toUpperCase).forEach(System.out::println);
        System.out.println("需求2：获得集合中所有学生的姓名");
        List<Student> studentList = StudentData.getStudentList();
        /*studentList.stream().map(new Function<Student, String>() {
            @Override
            public String apply(Student student) {
                return student.getName();
            }
        }).forEach(System.out::println);*/
        studentList.stream().map(
Student::getName).forEach(System.out::println);
        System.out.println("需求3：获得集合中性别为男的学生姓名");
        /**
         * 1.筛选：性别为男的学生对象
         * 2.映射：通过学生对象获得学生姓名
         */
        Stream<Student> stream1 = StudentData.getStudentList().stream();
        /* stream1.filter(new Predicate<Student>() {
            @Override
            public boolean test(Student student) {
                return student.getSex() == '男';
            }
        }).map(new Function<Student, String>() {
            @Override
            public String apply(Student student) {
                return student.getName();
            }
        }).forEach(System.out::println);*/
        stream1.filter( student-> student.getSex() == '男')
                .map(Student::getName)
```

```
47                  .forEach(System.out::println);
48      }
49  }
```

### 3.3.2 将多个集合中的元素映射到一个流中

代码块

```
1   package com.powernode.stream02;
2
3   import java.util.Arrays;
4   import java.util.List;
5   import java.util.function.Function;
6   import java.util.stream.Stream;
7
8   public class Test04 {
9       public static void main(String[] args) {
10          //将多个集合中的元素映射到一个流中
11          Stream<List<Integer>> stream = Stream.of(Arrays.asList(1, 2, 3),
    Arrays.asList(4, 5, 6), Arrays.asList(7, 8, 9));
12          //把流中集合取出放入流中（上面的流放的是集合，我们要放Integer对象）
13        /*  Stream<Integer> integerStream = stream.flatMap(new
    Function<List<Integer>, Stream<Integer>>() {
14              @Override
15              public Stream<Integer> apply(List<Integer> integers) {
16                  return integers.stream();
17              }
18          });
19          integerStream.forEach(System.out::println);*/
20        /* stream.flatMap(new Function<List<Integer>, Stream<Integer>>() {
21              @Override
22              public Stream<Integer> apply(List<Integer> integers) {
23                  return integers.stream();
24              }
25          }).forEach(System.out::println);*/
26          stream.flatMap(List<Integer>::stream).forEach(System.out::println);
27
28      }
29  }
```

## 3.4 去重（distinct）

### 3.4.1 包装类类型去除重复

```java
package com.powernode.stream03;

import java.util.stream.Stream;

public class Test01 {
    public static void main(String[] args) {
        Stream<Integer> stream = Stream.of(11, 22, 33, 11, 22, 44, 55);
        //去重
        stream.distinct().forEach(System.out::println);
    }
}
```

### 3.4.2 自定义类型去除重复

- *重写hashCode和equals*

代码块

```java
package com.powernode.stream03;



import java.util.function.Function;
import java.util.stream.Stream;

public class Test02 {
    public static void main(String[] args) {
        Stream<Student> stream = StudentData.getStudentList().stream();
        //需求1.去除重复的学生【注意：自定义对象，去除重复内容，重写hashCode和equals】
        stream.distinct().forEach(System.out::println);
        System.out.println("需求2：去除重复的学生，输出学生的姓名");
        /**
         * 1.distinct ： 去除重复学生
         * 2.map:映射学生年龄
         */
        StudentData.getStudentList().stream()
                .distinct()
                .map(new Function<Student, String>() {
                    @Override
                    public String apply(Student student) {
                        return student.getName();
                    }
                })
                .forEach(System.out::println);
        System.out.println("========优化========");
        StudentData.getStudentList().stream()
                .distinct()
```

```
                .map(Student::getName)
                .forEach(System.out::println);
    }
}
```

## 3.5 排序（sorted）

代码块

```
 package com.powernode.stream04;


import java.util.Comparator;
import java.util.function.Function;
import java.util.stream.Stream;

public class Test01 {
    public static void main(String[] args) {
        System.out.println("需求1：对流中的数据进行去重后升序排序");
        Stream.of(33, 11, 22, 44, 11, 22, 55)
                .distinct()
                .sorted()//默认升序
                .forEach(System.out::println);
        System.out.println("需求2：对流中的数据进行去重后降序排序");
        Stream.of(33, 11, 22, 44, 11, 22, 55)
                .distinct()
                .sorted(( o1,  o2) -> o2-o1)//默认升序
                .forEach(System.out::println);
        System.out.println("需求3：按照学生年龄升序排序");
        StudentData.getStudentList().stream()
                .sorted()//对自定义对象进行排序，必须让类实现Comparable接口，让类具有比较性
                .forEach(System.out::println);
        System.out.println("需求4：去除重复学生后，按照年龄进行降序排序，输出学生年龄");
        /*StudentData.getStudentList().stream()
                .distinct()
                .sorted(new Comparator<Student>() {
                    @Override
                    public int compare(Student o1, Student o2) {
                        return o2.getAge() - o1.getAge();
                    }
                })
                .map(new Function<Student, Integer>() {
                    @Override
                    public Integer apply(Student student) {
```

```
36                          return student.getAge();
37                      }
38                  })
39                  .forEach(System.out::println);*/
40          StudentData.getStudentList().stream()
41                  .distinct()
42                  .sorted(( o1,  o2)->
43                          o2.getAge() - o1.getAge()
44                  )
45                  .map(Student::getAge)
46                  .forEach(System.out::println);
47      }
48  }
```

## 3.6 合并（concat）

代码块

```
1   package com.powernode.stream05;
2
3   import java.io.Serializable;
4   import java.util.stream.Stream;
5
6   public class Test01 {
7       public static void main(String[] args) {
8           Stream<Integer> stream1 = Stream.of(11, 22, 33);
9           Stream<String> stream2 = Stream.of("aa", "bb", "cc");
10          /**
11           * <? extends Serializable> :向上限定
12           *   Number implements java.io.Serializable
13           *   String implements java.io.Serializable
14           */
15          Stream<? extends Serializable> concat = Stream.concat(stream1,
    stream2);
16          concat.forEach(System.out::println);
17
18      }
19  }
```

## 3.7 跳过和截取

代码块

```
1   package com.powernode.stream05;
2
```

```
import java.io.Serializable;
import java.util.stream.Stream;

public class Test02 {
    public static void main(String[] args) {
        Stream.of(11,22,33,44,55,66)
                .skip(2)//跳过前两个元素
                .limit(3)//截取前3个元素
                .forEach(System.out::println);//33,44,55

    }
}
```

# 4. Stream流的终端操作

- 触发 了终端操作，中间操作才会执行
- 终端操作执行完毕，会返回一个结果，关闭流
- Stream流关闭后，不可以再使用

## 4.1 遍历（forEach）掌握

代码块

```
package com.powernode.stream05;

import java.util.function.Consumer;
import java.util.function.Predicate;
import java.util.stream.Stream;

public class Test03 {
    public static void main(String[] args) {
       //需求：输出 年龄 > 18 的学生对象
        StudentData.getStudentList().stream()
                .filter(new Predicate<Student>() {
                    @Override
                    public boolean test(Student student) {
                        return student.getAge() > 18;
                    }
                })
                .forEach(new Consumer<Student>() {
                    @Override
                    public void accept(Student student) {
                        System.out.println(student);
                    }
                });
```

```
        System.out.println("====================");
        StudentData.getStudentList().stream()
                .filter( student->
                        student.getAge() > 18
                )
                .forEach(System.out::println);

    }
}
```

## 4.2 匹配（match）

- *匹配：流中是否存在某个元素*

代码块

```
package com.powernode.stream05;

import java.util.function.Predicate;

public class Test04 {
    public static void main(String[] args) {
        //匹配：流中是否存在某个元素
        //需求1：流中Student对象是否都为男性
       /* boolean flag = StudentData.getStudentList().stream()
                .allMatch(new Predicate<Student>() {
                    @Override
                    public boolean test(Student student) {
                        return student.getSex() == '男';
                    }
                });
        System.out.println(flag);*/
        boolean allMatch = StudentData.getStudentList().stream().allMatch(
student -> student.getSex() == '男');
        System.out.println("流中Student对象是否都为男性：" + allMatch);
        boolean anyMatch =
StudentData.getStudentList().stream().anyMatch(student -> student.getSex() ==
'男');
        System.out.println("流中Student对象至少有一个为男性：" + anyMatch);
        boolean noneMatch =
StudentData.getStudentList().stream().noneMatch(student -> student.getSex() ==
'男');
        System.out.println("流中Student对象一个都不为男性：" + noneMatch);


    }
```

```
26  }
```

## 4.3 规约（reduce）

### 4.3.1 使用规约做运算

代码块

```
1   package com.powernode.stream06;
2
3   import java.util.Arrays;
4   import java.util.List;
5   import java.util.Optional;
6   import java.util.function.BinaryOperator;
7   import java.util.function.Function;
8   import java.util.stream.Stream;
9
10  public class Test01 {
11      public static void main(String[] args) {
12          List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
13          System.out.println("需求1：获得所有元素相加的结果");
14          Optional<Integer> reduce = list.stream().reduce(new
    BinaryOperator<Integer>() {
15              @Override
16              public Integer apply(Integer x, Integer y) {
17                  //return x + y;
18                  return Integer.sum(x, y);
19              }
20          });
21          System.out.println("获得所有元素相加的结果：  " + reduce.get());
22
23          Optional<Integer> reduce1 = list.stream().reduce(Integer::sum);
24          System.out.println("获得所有元素相加的结果：  " + reduce1.get());
25
26          System.out.println("需求2：获得所有元素相乘的结果");
27          Integer integer = list.stream().reduce(( x,  y)-> x * y).get();
28          System.out.println("需求3：获得最大长度的元素");
29          String s = Stream.of("aa", "bbb", "cccc", "ddddd", "ee")
30                  .reduce(new BinaryOperator<String>() {
31                      @Override
32                      public String apply(String s1, String s2) {
33                          return s1.length() > s2.length() ? s1 : s2;
34                      }
35                  }).get();
36          System.out.println(s);
37          System.out.println("需求4：获得Student对象所有年龄之和");
```

```
        //1.拿到年龄，2.求和
        Integer sumAge = StudentData.getStudentList().stream()
                .map(new Function<Student, Integer>() {
                    @Override
                    public Integer apply(Student student) {
                        return student.getAge();
                    }
                }).reduce(new BinaryOperator<Integer>() {
                    @Override
                    public Integer apply(Integer x, Integer y) {
                        return Integer.sum(x, y);
                    }
                }).get();
        System.out.println("sumAge = " + sumAge);
        System.out.println("-----------------");
        Integer sumAge1 = StudentData.getStudentList().stream()
                .map(Student::getAge)
                .reduce(Integer::sum)
                .get();
        System.out.println("sumAge1 = " + sumAge1);
        System.out.println("需求5：获得一个数和所有年龄之和相加");
        Integer reduce2 = list.stream().reduce(10, Integer::sum);
        System.out.println("reduce2 = " + reduce2);

    }
}
```

## 4.3.2 使用规约做统计

代码块

```
package com.powernode.stream06;


import java.util.Comparator;
import java.util.function.Function;

public class Test02 {
    public static void main(String[] args) {
        System.out.println("需求1：获得list中Student对象的个数");
        long count = StudentData.getStudentList().stream().count();
        System.out.println("count = " + count);
        System.out.println("需求2：获得年龄最大的学生");
        Student student = StudentData.getStudentList().stream()
                .max(new Comparator<Student>() {
                    @Override
```

```
                    public int compare(Student o1, Student o2) {
                        return o1.getAge() - o2.getAge();
                    }
                }).get();
        System.out.println(student);
        Student student1 = StudentData.getStudentList().stream()
                .max(( o1,  o2)-> o1.getAge() - o2.getAge()).get();
        System.out.println(student1);
        System.out.println("需求3：获得年龄最大的学生年龄");
        //1.获得所有年龄，2.求年龄最大值
        Integer maxAge = StudentData.getStudentList().stream()
                .map(new Function<Student, Integer>() {
                    @Override
                    public Integer apply(Student student) {
                        return student.getAge();
                    }
                })
                .max(new Comparator<Integer>() {
                    @Override
                    public int compare(Integer o1, Integer o2) {
                        return Integer.compare(o1, o2);
                    }
                })
                .get();

        Integer maxAge1 = StudentData.getStudentList().stream()
                .map(Student::getAge)
                .max(Integer::compare)
                .get();
        System.out.println("maxAge1 = " + maxAge1);


    }
}
```

## 4.4 收集（collect）

- 把流中的数据收集起来，最终形参一个流或者集合

### 4.4.1 归集

#### 4.4.1.1 归集List,Set和Map中

代码块

```
package com.powernode.stream07;
```

```
2
3  import java.util.*;
4  import java.util.function.Function;
5  import java.util.stream.Collectors;
6  import java.util.stream.Stream;
7
8  public class Test01 {
9      public static void main(String[] args) {
10         List<String> list = Arrays.asList("a", "bb", "ccc", "aaa", "bbb",
   "ee");
11         Stream<String> stream = list.stream();
12         //需求1：把流中的数据归集到List中
13         //List<String> collect = stream.collect(Collectors.toList());
14         List<String> collect = stream.toList();
15
16         //需求2：把流中的数据归集到Set中
17         Set<String> collect1 = list.stream().collect(Collectors.toSet());
18         Set<String> set = new HashSet<>(list);
19
20         //需求3：把流中的数据归集到Map中
21         List<String> list1 = Arrays.asList("张三:北京", "李四:南京", "王五:上海");
22         /*Map<String, String> collect2 =
   list1.stream().collect(Collectors.toMap(new Function<String, String>() {
23             @Override
24             public String apply(String s) {
25                 return s.substring(0, s.indexOf(":"));//截取key
26             }
27         }, new Function<String, String>() {
28             @Override
29             public String apply(String s) {
30                 return s.substring(s.indexOf(":") + 1);
31             }
32         }));*/
33         Map<String, String> collect2 = list1.stream().collect(Collectors.
34                 toMap( s-> s.substring(0, s.indexOf(":"))
35         ,  s -> s.substring(s.indexOf(":") + 1)));
36         System.out.println("collect2 = " + collect2);
37
38     }
39 }
```

### 4.4.1.2 归集具体的集合

代码块

```
1  package com.powernode.stream07;
```

```
2
3  import java.util.*;
4  import java.util.function.Supplier;
5  import java.util.stream.Collectors;
6  import java.util.stream.Stream;
7
8  public class Test02 {
9      public static void main(String[] args) {
10         List<String> list = Arrays.asList("a", "bb", "ccc", "aaa", "bbb",
   "bb");
11         //需求1：把流中的数据归集到ArrayList中
12        /* ArrayList<String> collect =
   list.stream().collect(Collectors.toCollection(new Supplier<ArrayList<String>>
   () {
13
14             @Override
15             public ArrayList<String> get() {
16                 return new ArrayList<>();
17             }
18         }));*/
19         ArrayList<String> arrayList =
   list.stream().collect(Collectors.toCollection(ArrayList::new));
20         System.out.println("arrayList = " + arrayList);
21         //需求2：把流中的数据归集到LinkedList中
22         LinkedList<String> linkedList =
   list.stream().collect(Collectors.toCollection(LinkedList::new));
23         System.out.println("linkedList = " + linkedList);
24         //需求3：把流中的数据归集到HashSet中
25         HashSet<String> hashSet =
   list.stream().collect(Collectors.toCollection(HashSet::new));
26         System.out.println("hashSet = " + hashSet);
27         //需求4：把流中的数据归集到TreeSet中
28         TreeSet<String> treeSet =
   list.stream().collect(Collectors.toCollection(TreeSet::new));
29         System.out.println("treeSet = " + treeSet);
30     }
31 }
```

#### 4.4.1.3 将年龄>=18的女学生，按照年龄升序归集到ArrayList中

代码块

```
1   package com.powernode.stream07;
2
3  import java.util.ArrayList;
4  import java.util.Collection;
```

```java
import java.util.List;
import java.util.function.Predicate;
import java.util.function.Supplier;
import java.util.stream.Collectors;

public class Test03 {
    public static void main(String[] args) {
        /**
         * 将年龄>=18的女学生，按照年龄升序归集到ArrayList中
         *   1.过滤：年龄>=18的女学生
         *   2.排序：年龄排序
         *   3.归集到ArrayList
         */
        /*StudentData.getStudentList().stream()
                .filter(new Predicate<Student>() {
                    @Override
                    public boolean test(Student student) {
                        return student.getAge() >= 18 && student.getSex() == '女';
                    }
                })
                .sorted()
                .collect(Collectors.toCollection(new Supplier<ArrayList<Student>>() {
                    @Override
                    public ArrayList<Student> get() {
                        return new ArrayList<>();
                    }
                }))
                .forEach(System.out::println);*/
        StudentData.getStudentList().stream()
                .filter( student -> student.getAge() >= 18 && student.getSex() == '女')
                .sorted()
                .collect(Collectors.toCollection(ArrayList::new))
                .forEach(System.out::println);
    }
}
```

#### 4.4.1.4 把流中数据归集到数组

代码块

```java
package com.powernode.stream07;

import java.util.ArrayList;
```

```
import java.util.Arrays;
import java.util.function.IntFunction;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class Test04 {
    public static void main(String[] args) {
        Stream<String> stream = Stream.of("aa", "bb", "cc");
       /* String[] array = stream.toArray(new IntFunction<String[]>() {
            @Override
            public String[] apply(int length) {
                return new String[length];
            }
        });*/
        String[] array = stream.toArray(String[]::new);
        System.out.println(Arrays.toString(array));

    }
}
```

### 4.4.2 统计

代码块

```
package com.powernode.stream07;


import java.util.Comparator;
import java.util.IntSummaryStatistics;
import java.util.List;
import java.util.Optional;
import java.util.function.ToDoubleFunction;
import java.util.function.ToIntFunction;
import java.util.stream.Collectors;

public class Test05 {
    public static void main(String[] args) {
        List<Student> list = StudentData.getStudentList();
        //需求1：统计元素个数
        long count = list.stream().count();
        System.out.println(count);
        //需求2：获得年龄的平均值
       /* Double avgAge = list.stream().collect(Collectors.averagingDouble(new ToDoubleFunction<Student>() {
            @Override
            public double applyAsDouble(Student student) {
```

```
                return student.getAge();
            }
        }));
        */
        Double avgAge =
list.stream().collect(Collectors.averagingDouble(Student::getAge));
        System.out.println("avgAge = " + avgAge);
        //需求3：获得年龄最大的学生
        /*Optional<Student> collect =
list.stream().collect(Collectors.maxBy(new Comparator<Student>() {
            @Override
            public int compare(Student o1, Student o2) {
                return o1.getAge() - o2.getAge();
            }
        }));*/
        Student student = list.stream().collect(Collectors.maxBy((o1, o2) ->
                o1.getAge() - o2.getAge()
        )).get();
        System.out.println("student = " + student);
        //需求4：获得所有学生年龄之和
    /*    Integer sumAge = list.stream().collect(Collectors.summingInt(new
ToIntFunction<Student>() {
            @Override
            public int applyAsInt(Student student1) {
                return student1.getAge();
            }
        }));*/
        Integer sumAge =
list.stream().collect(Collectors.summingInt(Student::getAge));
        System.out.println("sumAge = " + sumAge);
        //需求5：统计汇总，年龄
       /* IntSummaryStatistics collect =
list.stream().collect(Collectors.summarizingInt(new ToIntFunction<Student>() {
            @Override
            public int applyAsInt(Student student1) {
                return student1.getAge();
            }
        }));*/
        IntSummaryStatistics collect =
list.stream().collect(Collectors.summarizingInt(Student::getAge));
        System.out.println(collect);

    }
}
```

### 4.4.3 分组

代码块

```
package com.powernode.stream07;

import java.util.List;
import java.util.Map;
import java.util.function.BiConsumer;
import java.util.function.Function;
import java.util.stream.Collectors;

public class Test06 {
    public static void main(String[] args) {
        List<Student> list = StudentData.getStudentList();
        //需求：按照学生的性别进行分组，性别为key，集合为value
        Map<Character, List<Student>> collect =
list.stream().collect(Collectors.groupingBy(new Function<Student, Character>()
{
            @Override
            public Character apply(Student student) {
                return student.getSex();
            }
        }));
        System.out.println(collect);
        collect.forEach(new BiConsumer<Character, List<Student>>() {
            @Override
            public void accept(Character k, List<Student> v) {
                System.out.println(k + ":" + v);
            }
        });
    }
}
```

### 4.4.4 结合

代码块

```
package com.powernode.stream07;

import java.util.List;
import java.util.Map;
import java.util.function.BiConsumer;
import java.util.function.Function;
import java.util.stream.Collectors;

public class Test07 {
    public static void main(String[] args) {
```

```
11          List<Student> list = StudentData.getStudentList();
12          //需求：将学生姓名连接成一个字符串，中间使用逗号隔开
13        /*  String collect = list.stream().map(new Function<Student, String>() {
14              @Override
15              public String apply(Student student) {
16                  return student.getName();
17              }
18          }).collect(Collectors.joining(","));
19          System.out.println(collect);*/
20          String collect =
    list.stream().map(Student::getName).collect(Collectors.joining(","));
21          System.out.println(collect);
22      }
23  }
```