新特性

一、Lambda表达式（掌握）

- 简化接口实现类的写法

1.1 简化线程Runnable的写法

```
package com.qf.lambda;

/**
 * Lambda表达式简化线程的开启
 * @author Dushine2008
 *
 */
public class MyLambda01 {
    public static void main(String[] args) {
        Runnable r1 = new Runnable() {
            @Override
            public void run() {
                System.out.println(Thread.currentThread().getName() + "正在启动....");
            }
        };

        Thread t1 = new Thread(r1);
        t1.start();

        Runnable r2 = () -> {
            System.out.println(Thread.currentThread().getName() + "正在启动....");
        };
        Thread t2 = new Thread(r2);
        t2.start();

        Runnable r3 = () -> System.out.println(Thread.currentThread().getName() + "正在启动....");
        Thread t3 = new Thread(r3);
        t3.start();

        Thread t4 = new Thread(() -> System.out.println(Thread.currentThread().getName() + "正在启动...."));
        t4.start();

        new Thread(() -> System.out.println(Thread.currentThread().getName() + "正在启动....")).start();
    }

}
```

1.2 简化Comparator写法

```
package com.qf.lambda;
```

```java
import java.util.Comparator;
import java.util.TreeSet;

public class MyLambda02 {
    public static void main(String[] args) {
        Comparator<String> comparator01 = new Comparator<String>() {
            @Override
            public int compare(String s1, String s2) {
                return s1.compareTo(s2);
            }
        };
        TreeSet<String> set01 = new TreeSet<String>(comparator01);
        set01.add("hgf");
        set01.add("erty");
        set01.add("nbvfc");
        set01.add("xcvbss");
        set01.add("sadfsd");
        set01.add("jhgfdast");
        System.out.println(set01);

        Comparator<String> comparator02 = (String s1,String s2) -> {
            return s1.compareTo(s2);
        };
        TreeSet<String> set02 = new TreeSet<String>(comparator02);
        set02.add("hgf");
        set02.add("erty");
        set02.add("nbvfc");
        set02.add("xcvbss");
        set02.add("sadfsd");
        set02.add("jhgfdast");
        System.out.println(set02);

        Comparator<String> comparator03 = (s1,s2) -> {
            return s1.compareTo(s2);
        };

        TreeSet<String> set03 = new TreeSet<String>(comparator03);
        set03.add("hgf");
        set03.add("erty");
        set03.add("nbvfc");
        set03.add("xcvbss");
        set03.add("sadfsd");
        set03.add("jhgfdast");
        System.out.println(set03);

        // {} 和 return要么都在，要么都不在
        Comparator<String> comparator04 = (s1,s2) -> s1.compareTo(s2);

        TreeSet<String> set04 = new TreeSet<String>(comparator04);
        set04.add("hgf");
        set04.add("erty");
        set04.add("nbvfc");
        set04.add("xcvbss");
        set04.add("sadfsd");
        set04.add("jhgfdast");
        System.out.println(set04);
```

```
        TreeSet<String> set05 = new TreeSet<String>((s1,s2) -> s1.compareTo(s2));
        set05.add("hgf");
        set05.add("erty");
        set05.add("nbvfc");
        set05.add("xcvbss");
        set05.add("sadfsd");
        set05.add("jhgfdast");
        System.out.println(set05);

        TreeSet<String> set06 = new TreeSet<String>(String::compareTo);
        set06.add("hgf");
        set06.add("erty");
        set06.add("nbvfc");
        set06.add("xcvbss");
        set06.add("sadfsd");
        set06.add("jhgfdast");
        System.out.println(set06);

    }

}
```

1.3 自定义函数式接口

```
package com.qf.lambda;

public class MyLambda03 {

    public static void main(String[] args) {
        GetSum getSum = new GetSum() {
            @Override
            public int add2num(int a, int b) {
                return a+b;
            }
        };
        int add2num = getSum.add2num(33, 88);
        System.out.println(add2num);

        GetSum getSum02 = (int a,int b) -> a+b;
        int add2num2 = getSum02.add2num(88, 121);
        System.out.println(add2num2);

        GetMul getMul = (int a,int b) -> System.out.println(a*b);
        getMul.mul2num(33, 55);
    }

}

/**
 *  获取相加结果的接口
 * @author Dushine2008
 *
 */
@FunctionalInterface
interface GetSum{
    int add2num(int a,int b);
```

```
}

/**
 *  获取相乘结果的接口
 * @author Dushine2008
 *
 */
@FunctionalInterface
interface GetMul{
    void mul2num(int a,int b);
}
```

二、常见函数型接口（熟悉）

```
package com.qf.lambda;

import java.util.Arrays;
import java.util.Random;
import java.util.function.Consumer;
import java.util.function.Supplier;

public class MyFunctionInter01 {

    public static void main(String[] args) {
        Consumer<Integer> consumer = new Consumer<Integer>() {
            @Override
            public void accept(Integer t) {
                System.out.println("这次团建花费了" + t);
            }
        };

        play(consumer, 168);

        Consumer<Integer> consumer02 = (t) -> System.out.println("这次团建花费了" + t);
        play(consumer02, 861);

        play((t) -> System.out.println("这次团建花费了" + t), 681);

        Supplier<Integer> supplier = new Supplier<Integer>() {

            @Override
            public Integer get() {
                return new Random().nextInt(1000);
            }
        };

        int[] arr = getArr(supplier, 10);
        System.out.println(Arrays.toString(arr));

        Supplier<Integer> supplier0 = () -> new Random().nextInt(1000);

        int[] arr2 = getArr(supplier0, 10);
        System.out.println(Arrays.toString(arr2));

        int[] arr3 = getArr(() -> new Random().nextInt(1000), 10);
```

```
        System.out.println(Arrays.toString(arr3));

    }

    /**
     *  消费型接口
     *      传入T类型的参数
     *      使用这个参数
     * @param consumer
     * @param money
     */
    public static void play(Consumer<Integer> consumer,Integer money) {
        consumer.accept(money);
    }


    /**
     *  供给型接口
     * @param supplier
     * @param count
     * @return
     */
    public static int[] getArr(Supplier<Integer> supplier,int count) {
        int[] arr = new int[count];
        for (int i = 0; i < arr.length; i++) {
            // 通过自定义的规则向数组中添加元素
            arr[i] = supplier.get();
        }

        return arr;
    }

}
```

```
package com.qf.lambda;

import java.util.ArrayList;
import java.util.List;
import java.util.function.Function;
import java.util.function.Predicate;

public class MyFunctionInter02 {
    public static void main(String[] args) {
        Function<String, Integer> function = new Function<String, Integer>() {
            @Override
            public Integer apply(String t) {
                return t.length();
            }
        };

        Integer len = getStrLen(function, "vohgbdfakdlfja[odjf");
        System.out.println(len);

        Function<String, Integer> function01 = (String t) ->  t.length();
        Integer len2 = getStrLen(function01, "hfisdhhdifhodsfowefjslfjoiwef");
        System.out.println(len2);
```

```
        Integer len3 = getStrLen((String t) ->  t.length(),
"hfisdhhdifhodsfowefjslfjoiwef");
        System.out.println(len3);

        Predicate<String> predicate = new Predicate<String>() {
            @Override
            public boolean test(String t) {
                return t.startsWith("zhang");
            }
        };

        ArrayList<String> list = new ArrayList<String>();
        list.add("zhangsan");
        list.add("lisi");
        list.add("wangwu");
        list.add("zhaoliu");
        list.add("zhangsansan");

        List<String> newList = getZhang(predicate, list);
        System.out.println(newList);
    }

    /**
     *  函数型接口
     *      操作T
     *      返回R
     * @param function
     * @param str
     * @return
     */
    public static Integer getStrLen(Function<String, Integer> function,String str) {
        return function.apply(str);
    }

    /**
     *  断言型接口
     * @param predicate
     * @param list
     * @return
     */
    public static List<String> getZhang(Predicate<String> predicate,List<String> list)
{
        ArrayList<String> newList = new ArrayList<String>();

        for (String name : list) {
            if (predicate.test(name)) {
                newList.add(name);
            }
        }

        return newList;
    }
}
```

三、方法引用

- Lambda的简写形式
- 实例::实例方法
- 实例::静态方法
- 类::静态方法
- 类::new

```
package com.qf.lambda;

import java.util.Comparator;
import java.util.function.Consumer;
import java.util.function.Supplier;

public class MyLambda01 {

    public static void main(String[] args) {
        Consumer<String> consumer01 = (t) -> System.out.println(t);
        consumer01.accept("我是消费型接口输出的内容");

        // 对象 :: 实例方法
        Consumer<String> consumer02 = System.out::println;
        consumer02.accept("我也是消费型接口输出的内容========");

        Comparator<Integer> comparator01 = (i1, i2) -> Integer.compare(i1, i2);
        int i01 = comparator01.compare(555, 666);
        System.out.println(i01);

        Comparator<Integer> comparator02 = Integer::compare;
        int i02 = comparator02.compare(888, 333);
        System.out.println(i02);

        // 类::静态方法
        Consumer<String> consumer03 = (t) -> Person.show(t);
        consumer03.accept("Person中的show000000");

        Consumer<String> consumer04 = Person::show;
        consumer04.accept("Person中的show11111111111111111");


        Supplier<Person> supplier01 = new Supplier<Person>() {
            @Override
            public Person get() {
                return new Person();
            }
        };

        Person person01 = supplier01.get();
        System.out.println(person01);

        // 类::new
        Supplier<Person> supplier02 = Person::new;
        Person person02 = supplier02.get();
        System.out.println(person02);

    }

}
```

四、Stream

4.1 定义

- 流
- 有创建、中间操作、终止操作

4.2 创建Stream对象

```
package com.qf.stream;

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

/**
 *  创建Stream对象
 * @author Dushine2008
 *
 */
public class MyStream01 {
    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<String>();

        list.add("曹操");
        list.add("大乔");
        list.add("貂蝉");
        list.add("黄月英");
        list.add("吕布");

        System.out.println(list);

        Stream<String> stream01 = list.stream();
        System.out.println(stream01);
        // 遍历stream中的数据
        stream01.forEach(System.out::println);

        list.stream().forEach(System.out::println);

        System.out.println("==================");

        int sum = Arrays.stream(new int[] {33,55,77,99}).sum();
        System.out.println(sum);

        IntStream stream02 = Arrays.stream(new int[] {33,55,77,99});
        stream02.forEach(System.out::println);
        System.out.println("==================");
        // stream02.forEach(System.out::println);

        IntStream stream03 = IntStream.of(11,22,33,44,55,66);
        stream03.forEach(System.out::println);


    }
}
```

4.3 中间操作

```
package com.qf.stream;

import java.util.ArrayList;

public class MyStream02 {

    public static void main(String[] args) {
        ArrayList<Employee> list = new ArrayList<Employee>();

        list.add(new Employee("songjiang", 13000));
        list.add(new Employee("chaogai", 15000));
        list.add(new Employee("wusong", 11000));
        list.add(new Employee("luzhishen", 8000));
        list.add(new Employee("likui", 18000));
        list.add(new Employee("linchong", 28000));
        list.add(new Employee("linchong", 28000));

        System.out.println(list);

        System.out.println("==============filter================");

        list.stream().filter((e) -> e.getMoney()>=12000).forEach(System.out::println);

        System.out.println("==============limit================");
        list.stream().limit(3).forEach(System.out::println);

        System.out.println("==============skip================");
        list.stream().skip(3).forEach(System.out::println);

        System.out.println("==============distinct================");
        list.stream().distinct().forEach(System.out::println);

        System.out.println("==============sorted================");
        list.stream().sorted((e1,e2) ->
e1.getName().compareTo(e2.getName())).forEach(System.out::println);

    }

}
```

4.4 终止操作

```
package com.qf.stream;

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;

public class MyStream03 {
    public static void main(String[] args) {
        ArrayList<Employee> list = new ArrayList<Employee>();
```

```
        list.add(new Employee("songjiang", 13000));
        list.add(new Employee("chaogai", 15000));
        list.add(new Employee("wusong", 11000));
        list.add(new Employee("luzhishen", 8000));
        list.add(new Employee("likui", 18000));
        list.add(new Employee("linchong", 28000));
        list.add(new Employee("linchong", 28000));

        System.out.println(list);


        System.out.println("===============min================");
        Optional<Employee> min = list.stream().min((e1,e2) -> (int)(e1.getMoney()-
e2.getMoney()));
        System.out.println(min);

        System.out.println("===============max================");
        Optional<Employee> max = list.stream().min((e1,e2) -> (int)(e2.getMoney()-
e1.getMoney()));
        System.out.println(max);

        System.out.println("===============count================");
        long count = list.stream().count();
        System.out.println(count);

        System.out.println("===============reduce：规约================");
        Optional<Double> reduce = list.stream().map(e->e.getMoney()).reduce((x,y)->
(x+y));
        System.out.println(reduce);

        System.out.println("===============collect================");
        List<String> collect = list.stream().map(e-
>e.getName()).collect(Collectors.toList());
        System.out.println(collect);
    }
}
```

五、时间API

5.1 新的时间API

- LocalDate
- LocalTime
- LocalDateTime
- Instant
- ZoneId

```
package com.qf.newtime;

import java.time.Instant;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.ZoneId;
import java.util.Set;
```

```
public class MyLocalDate {

    public static void main(String[] args) {
        // 创建日期对象
        LocalDate date01 = LocalDate.now();
        System.out.println(date01);

        System.out.println(date01.getDayOfMonth());
        System.out.println(date01.getDayOfYear());
        System.out.println(date01.getDayOfWeek());

        // 创建时间对象
        LocalDateTime dateTime = LocalDateTime.now();
        System.out.println(dateTime);

        System.out.println(dateTime.getDayOfMonth());
        System.out.println(dateTime.getDayOfYear());
        System.out.println(dateTime.getDayOfWeek());
        System.out.println(dateTime.getHour());
        System.out.println(dateTime.getMinute());

        Instant instant = Instant.now();
        System.out.println(instant);
        // 获取1970-01-01到现在的秒
        System.out.println(instant.getEpochSecond());
        System.out.println(System.currentTimeMillis());

        // 获取所有的时区
        Set<String> ids = ZoneId.getAvailableZoneIds();
        for (String id : ids) {
            System.out.println(id);
        }
    }
}
```

3.2 新老时间转换

```
package com.qf.newtime;

import java.time.Instant;
import java.time.LocalDateTime;
import java.time.ZoneId;
import java.util.Date;

public class NewOldTime {

    public static void main(String[] args) {
        // 老时间===>新时间
        Date date01 = new Date();
        System.out.println(date01);

        Instant instant01 = date01.toInstant();

        LocalDateTime dateTime01 = LocalDateTime.ofInstant(instant01,
ZoneId.systemDefault());
        System.out.println(dateTime01);
```

```
        // 新时间===>老时间
        LocalDateTime dateTime02 = LocalDateTime.now();

        Instant instant02 = dateTime02.atZone(ZoneId.systemDefault()).toInstant();

        Date date02 = Date.from(instant02);
        System.out.println(date02);

    }
}
```