第三章 Stream流及常用API

课前回顾

1. 如何使用方法引用

因为方法有静态方法、成员方法和构造方法之分，因此方法引用也分为静态方法引用、成员方法引用以及构造方法引用。

```
//静态方法引用
类名::方法名;

//成员方法引用
对象名::方法名;
this::方法名;
super::方法名;

//构造方法引用
类名::new
```

2. 描述 Consumer、Predicate、Function接口的作用

Consumer是消费者意思，主要用来消费数据。Predicate接口是条件的意思，主要用于检测数据是否满足条件。Function是功能的意思，主要用来转换数据类型

章节内容

- `Stream`流 **重点**
- 自动装箱和拆箱 **重点**
- Date类 **熟悉**
- Calendar类 **重点**

章节目标

- 掌握`Stream`流的使用
- 掌握自动装箱和拆箱
- 熟悉Date类
- 掌握Calendar类的使用

第一节 Stream流

1. 管道

管道

```
A pipeline is a sequence of aggregate operations.
管道就是一系列的聚合操作。
A pipeline contains the following components:
管道包含以下组件：
A source: This could be a collection, an array, a generator function, or an I/O channel.
源：可以是集合，数组，生成器函数或I / O通道。
Zero or more intermediate operations. An intermediate operation, such as filter, produces a new stream.
零个或多个中间操作。诸如过滤器之类的中间操作产生新的流。
A terminal operation. A terminal operation, such as forEach, produces a non-stream result, such as a primitive value (like a double value), a collection, or in the case of forEach, no value at all.
终结操作。终端操作（例如forEach）会产生非流结果，例如原始值（如双精度值），集合，或者在forEach的情况下根本没有任何值。
```

流

```
A stream is a sequence of elements. Unlike a collection, it is not a data structure that stores elements. Instead, a stream carries values from a source through a pipeline.
流是一系列元素。 与集合不同，它不是存储元素的数据结构。 取而代之的是，流通过管道携带来自源的值。
The filter operation returns a new stream that contains elements that match its predicate (this operation's parameter).
筛选器操作返回一个新流，该流包含与筛选条件（此操作的参数）匹配的元素。
```

2.如何获取`Stream`流

`Collection`接口

```
default Stream<E> stream();
```

`Stream`接口

```
//获取流
public static<T> Stream<T> of(T... values);
//将两个流拼接形成一个新的流
public static <T> Stream<T> concat(Stream<? extends T> a, Stream<? extends T> b);
```

示例

```
package com.cyx.stream;


import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.stream.Stream;

public class StreamTest {

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

```
        List<Integer> numbers1 = Arrays.asList(1, 2, 3, 4, 5);
        Stream<Integer> s1 = numbers1.stream();

        Stream<Integer> s2 = Stream.of(6, 7, 8, 9, 10);

        Stream<Integer> s3 = Stream.concat(s1, s2);

        Map<String,Integer> map = new HashMap<>();
        map.put("a", 1);
        map.put("b", 2);
        Stream<Map.Entry<String,Integer>> s4 = map.entrySet().stream();
    }
}
```

3. `Stream` 流中间聚合操作

`Stream`接口

```
Stream<T> filter(Predicate<? super T> predicate);//根据给定的条件过滤流中的元素
<R> Stream<R> map(Function<? super T, ? extends R> mapper);//将流中元素进行类型转换
Stream<T> distinct();//去重
Stream<T> sorted();//排序，如果存储元素没有实现Comparable或者相关集合没有提供Comparator将抛出异常
Stream<T> limit(long maxSize);//根据给定的上限，获取流中的元素
Stream<T> skip(long n);//跳过给定数量的元素

IntStream mapToInt(ToIntFunction<? super T> mapper);//将流中元素全部转为整数
LongStream mapToLong(ToLongFunction<? super T> mapper);//将流中元素全部转为长整数
DoubleStream mapToDouble(ToDoubleFunction<? super T> mapper);//将流中元素全部转为双精度浮点数
```

示例

```
package com.cyx.stream;

import java.util.Arrays;
import java.util.List;
import java.util.Set;
import java.util.function.Function;
import java.util.function.IntFunction;
import java.util.function.Predicate;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class AggregateOperation {

    public static void main(String[] args) {
//        List<Integer> numbers = Arrays.asList(30,21,58,67,90,72);
//        for(Integer number: numbers){
//            if(number % 2 == 0){
//                System.out.println(number);
//            }
//        }
```

```
21
22          Stream<Integer> s1 = Stream.of(30,21,58,67,90,72,58,30,80);
23  //        Stream<Integer> s2 = s1.filter(new Predicate<Integer>() {
24  //            @Override
25  //            public boolean test(Integer integer) {
26  //                return integer % 2 == 0;
27  //            }
28  //        });
29          Stream<String> s2 = s1.filter(number -> number % 2 ==
    0).distinct().skip(1).sorted().limit(2).map(integer-> "字符串"+ integer);
30          s2.forEach(System.out::println);
31          //将流中的元素收集为一个List集合，此时流就已经消亡了
32  //        List<Integer> existNumbers = s2.collect(Collectors.toList());
33
34          //再使用流进行收集操作时就将报错
35  //        Set<Integer> set = s2.collect(Collectors.toSet());
36
37  //        Integer[] arr = s2.toArray(new IntFunction<Integer[]>() {
38  //            @Override
39  //            public Integer[] apply(int value) {
40  //                return new Integer[value];
41  //            }
42  //        });
43          //再使用流进行收集操作时就将报错
44  //        Integer[] arr = s2.toArray(Integer[]::new);
45  //        for(Integer number: arr){
46  //            System.out.println(number);
47  //        }
48      }
49  }
```

```
1   package com.cyx.stream;
2
3   import java.util.Arrays;
4   import java.util.stream.DoubleStream;
5   import java.util.stream.IntStream;
6   import java.util.stream.LongStream;
7   import java.util.stream.Stream;
8
9   public class NumberStream {
10
11      public static void main(String[] args) {
12          Stream<String> s1 = Stream.of("1","2","3","4");
13  //        IntStream s2 = s1.mapToInt(new ToIntFunction<String>() {
14  //            @Override
15  //            public int applyAsInt(String value) {
16  //                return Integer.parseInt(value);
17  //            }
18  //        });
19  //        IntStream s2 = s1.mapToInt(value->Integer.parseInt(value));
20          IntStream s2 = s1.mapToInt(Integer::parseInt);
21          int[] arr = s2.toArray();
22          System.out.println(Arrays.toString(arr));
23
24          LongStream s3 =
    Stream.of("1","2","3","4").mapToLong(Long::parseLong);
25          long[] arr2 = s3.toArray();
```

```
        System.out.println(Arrays.toString(arr2));

        DoubleStream s4 =
Stream.of("1","2","3","4").mapToDouble(Double::parseDouble);
        double[] arr3 = s4.toArray();
        System.out.println(Arrays.toString(arr3));
    }
}
```

4. `Stream` 流终结操作

```
void forEach(Consumer<? super T> action);//遍历操作流中元素
<A> A[] toArray(IntFunction<A[]> generator);//将流中元素按照给定的转换方式转换为数组
<R, A> R collect(Collector<? super T, A, R> collector);//将流中的元素按照给定的方式搜集起来

Optional<T> min(Comparator<? super T> comparator);//根据给定的排序方式获取流中最小元素
Optional<T> max(Comparator<? super T> comparator);//根据给定的排序方式获取流中最大元素
Optional<T> findFirst(); //获取流中第一个元素

long count();//获取流中元素数量
boolean anyMatch(Predicate<? super T> predicate);//检测流中是否存在给定条件的元素
boolean allMatch(Predicate<? super T> predicate);//检测流中元素是否全部满足给定条件
boolean noneMatch(Predicate<? super T> predicate);//检测流中元素是否全部不满足给定条件
```

示例

```
package com.cyx.stream;

import java.util.Arrays;
import java.util.List;
import java.util.Optional;
import java.util.Set;
import java.util.function.Function;
import java.util.function.IntFunction;
import java.util.function.Predicate;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class AggregateOperation {

    public static void main(String[] args) {
        List<String> numbers =
Arrays.asList("30","21","58","67","90","72");
//        Stream<String> s = numbers.stream();
//        Optional<String> first = s.findFirst();
//        System.out.println(first.get());
//        Optional<String> optional = s.max(String::compareTo);
```

```
//        String max = optional.get();
//        System.out.println(max);
//        System.out.println(s.count());
//        numbers.stream().map(new Function<String, Integer>() {
//            @Override
//            public Integer apply(String s) {
//                return Integer.parseInt(s);
//            }
//        }).anyMatch(new Predicate<Integer>() {
//            @Override
//            public boolean test(Integer integer) {
//                return integer % 2 == 1;
//            }
//        });
        boolean exist1 = numbers.stream().map(Integer::parseInt).anyMatch(number->number % 2 ==1);
        System.out.println(exist1);

        boolean exist2 = numbers.stream().map(Integer::parseInt).allMatch(number->number%2 ==1);
        System.out.println(exist2);

        boolean exist3 = numbers.stream().map(Integer::parseInt).noneMatch(number->number%2 ==1);
        System.out.println(exist3);
    }
}
```

5. `Stream`流聚合操作与迭代器的区别

Aggregate operations, like forEach, appear to be like iterators. However, they have several fundamental differences:
聚合操作（如forEach）似乎像迭代器。但是，它们有几个基本差异：

They use internal iteration: Aggregate operations do not contain a method like next to instruct them to process the next element of the collection. With internal delegation, your application determines what collection it iterates, but the JDK determines how to iterate the collection. With external iteration, your application determines both what collection it iterates and how it iterates it. However, external iteration can only iterate over the elements of a collection sequentially. Internal iteration does not have this limitation. It can more easily take advantage of parallel computing, which involves dividing a problem into subproblems, solving those problems simultaneously, and then combining the results of the solutions to the subproblems.
它们使用内部迭代：聚合操作不包含诸如next的方法来指示它们处理集合的下一个元素。使用内部委托，你的应用程序确定要迭代的集合，而JDK确定如何迭代该集合。通过外部迭代，你的应用程序既可以确定要迭代的集合，又可以确定迭代的方式。但是，外部迭代只能顺序地迭代集合的元素。内部迭代没有此限制。它可以更轻松地利用并行计算的优势，这涉及将问题分为子问题，同时解决这些问题，然后将解决方案的结果组合到子问题中。

They process elements from a stream: Aggregate operations process elements from a stream, not directly from a collection. Consequently, they are also called stream operations.
它们处理流中的元素：聚合操作从流中而不是直接从集合中处理元素。因此，它们也称为流操作。

They support behavior as parameters: You can specify lambda expressions as parameters for most aggregate operations. This enables you to customize the behavior of a particular aggregate operation.
它们支持将行为作为参数：你可以将lambda表达式指定为大多数聚合操作的参数。这使你可以自定义特定聚合操作的行为。

面试题

使用Stream流将一个基本数据类型的数组转换为包装类型的数组，再将包装类型的数组转换基本数据类型数组。

```
int[] intArr1 = {1, 2, 3, 4, 5};
//        Integer[] intArr2 = Arrays.stream(intArr1).mapToObj(new IntFunction<Integer>() {
//            @Override
//            public Integer apply(int value) {
//                return value;
//            }
//        }).toArray(new IntFunction<Integer[]>() {
//            @Override
//            public Integer[] apply(int value) {
//                return new Integer[value];
//            }
//        });
//        Integer[] intArr2 = Arrays.stream(intArr1).mapToObj(value -> value).toArray(Integer[]::new);
Integer[] intArr2 = Arrays.stream(intArr1).boxed().toArray(Integer[]::new);

Integer[] intArr3 = {1,2,3,4,5};
int[] intArr4 = Arrays.stream(intArr3).mapToInt(Integer::intValue).toArray();
```

第二节 包装类

1. 什么是包装类

```
There are, however, reasons to use objects in place of primitives, and the Java platform provides wrapper classes for each of the primitive data types. These classes "wrap" the primitive in an object.

不论怎样，总有理由使用对象代替原始数据类型，并且Java平台为每种原始数据类型提供了包装类。 这些类将“原始数据类型”包装在对象中。
```

基本数据类型	包装类	基本数据类型	包装类
int	**Integer**	char	**Character**
byte	Byte	boolean	Boolean
short	Short	float	Float
long	Long	double	Double

2. 自动装箱和拆箱

自动装箱

```
Often, the wrapping is done by the compiler—if you use a primitive where an object is expected, the compiler boxes the primitive in its wrapper class for you.
通常，包装是由编译器完成的-如果你在期望一个对象的地方使用原始数据类型，则编译器会为你将原始数据类型放入其包装类中。
```

自动装箱方法

```
包装类名.valueOf(原始数据类型的值);
```

示例

```
//变量num期望获取一个整数对象，但赋值时给定的是一个基本数据类型int值，此时编译器将会将int值5进行包装
//调用的是Integer.valueOf(5)
Integer num = 5;
```

自动拆箱

```
Similarly, if you use a number object when a primitive is expected, the compiler unboxes the object for you.
类似地，如果在期望使用基本数据类型的情况下使用包装类型，则编译器会为你解包该对象。
```

自动拆箱方法

```
包装类对象.xxxValue();
```

示例

```
Integer num = new Integer(10);
//变量a期望获取一个基本数据类型的值，但赋值时给定的是一个引用数据类型的对象，此时编译器会将这个引用数据类型
//的对象中存储的数值取出来，然后赋值给变量a。调用的是num.intValue();
int a = num;
```

3.字符串转数字的方法

`Integer.parseInt("123")` 将字符串类型的数字转换为整数

`Long.parseLong("123")` 将字符串类型的数字转换为长整数

`Byte.parseByte("13")` 将字符串类型的数字转换为字节

`Short.parseShort("13")` 将字符串类型的数字转换为短整数

`Float.parseFloat("12.0f")` 将字符串类型的数字转换为单精度浮点数

`Double.parseDouble("123")` 将字符串类型的数字转换为双精度浮点数

`Boolean.parseBoolean("true")` 将字符串类型的布尔值转换为布尔值

示例

```
int num = Integer.parseInt("11");
float f = Float.parseFloat("12.5");
short s = Short.parseShort("5");
```

注意:如果字符串参数的内容无法正确转换为对应的基本类型，则会抛出 `java.lang.NumberFormatException` 异常。

第三节 日期时间

1. `Date` 类

常用方法

```
public Date(); //无参构造，表示计算机系统当前时间，精确到毫秒
public Date(long date);//带参构造，表示根据给定的时间数字来构建一个日期对象，精确到毫秒

public long getTime();//获取日期对象中的时间数字，精确到毫秒
public boolean before(Date when);//判断当前对象表示的日期是否在给定日期之前
public boolean after(Date when);//判断当前对象表示的日期是否在给定日期之后
```

示例

```
package com.cyx.date;

import java.util.Date;

public class DateTest {

    public static void main(String[] args) {
        Date now = new Date();
        System.out.println(now);
```

```
        //时间单位一般是毫秒，当然更精确的是纳秒
        long currentTime = System.currentTimeMillis() - 24 * 60 * 60 *
1000;
        Date date = new Date(currentTime);
        System.out.println(date);
        System.out.println(date.getTime());
        boolean before = now.before(date);//false
        boolean after = now.after(date);//true
        System.out.println(before);
        System.out.println(after);
    }
}
```

思考：打印的日期不好看懂，能否按照我们熟悉的方式来打印?

当然可以，首先我们需要将日期按照我们熟悉的日期格式转换为一个字符串日期，然后再打印。

2. `SimpleDateFormat` 类

常用方法

```
public SimpleDateFormat(String pattern); //根据给定的日期格式构建一个日期格式化对象

public final String format(Date date);//将给定日期对象进行格式化

public Date parse(String source) throws ParseException;//将给定的字符串格式日期解
析为日期对象
```

常用日期格式

字母	含义	说明
y	年year	不区分大小写，一般用小写
M	月month	区分大小写，只能使用大写
d	日day	区分大小写，只能使用小写
H	时hour	不区分大小写，一般用大写
m	分minute	区分大小写，只能使用小写
s	秒second	区分大小写，只能使用小写

示例

```
package com.cyx.date;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

public class DateFormatTest {

    public static void main(String[] args) {
        Date now = new Date();
        // yyyy-MM-dd HH:mm:ss  yyyy/MM/dd HH:mm:ss
```

```
        SimpleDateFormat format = new SimpleDateFormat("yyyy年MM月dd日HH时mm分ss秒");
        String dateStr = format.format(now);
        System.out.println(dateStr);

        String s = "2000-01-01 12:01:50";
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        try {
            Date date = sdf.parse(s);
            System.out.println(date);
        } catch (ParseException e) {
            e.printStackTrace();
        }
    }
}
```

3. `Calendar` 类

常用静态字段

字段值	含义
YEAR	年
MONTH	月，**从0开始**
DAY_OF_MONTH	月中第几天
HOUR	时（12小时制）
HOUR_OF_DAY	时（24小时制）
MINUTE	分
SECOND	秒
DAY_OF_WEEK	周中第几天，**一周的第一天是周日，因此周日是1**

常用方法

```
public static Calendar getInstance();//获取日历对象
public final Date getTime();//获取日历表示日期对象
public final void setTime(Date date);//设置日历表示的日期对象

public int get(int field);//获取给定的字段的值
public void set(int field, int value);//设置给定字段的值
public void roll(int field, int amount);//根据给定的字段和更改数量滚动日历
public int getActualMaximum(int field);//获取给定字段的实际最大数量
```

示例

```
package com.cyx.date;

import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;

```

```java
public class CalendarTest {

    public static void main(String[] args) {
        //获取一个日历对象，默认就是系统当前时间
        Calendar c = Calendar.getInstance();
        Date date = c.getTime();
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        System.out.println(sdf.format(date));

        long time = System.currentTimeMillis() - 3 * 24 * 60 * 60 * 1000;
        Date before = new Date(time);
        c.setTime(before);//设置时间到3天前

        c.set(Calendar.YEAR, 2019);//设置年份为2020年
        c.set(Calendar.MONTH, 1);//设置月份

        c.roll(Calendar.DAY_OF_MONTH, -1);//天数-1

        int year = c.get(Calendar.YEAR);//获取年份
        System.out.println(year);

        int month = c.get(Calendar.MONTH);//获取月份
        System.out.println(month);

        int day = c.get(Calendar.DAY_OF_MONTH);//获取一个月中的第几天
        System.out.println(day);

        int hour = c.get(Calendar.HOUR_OF_DAY);//获取小时(24)
        System.out.println(hour);

        int minute = c.get(Calendar.MINUTE);//获取分钟
        System.out.println(minute);

        int second = c.get(Calendar.SECOND);//获取秒
        System.out.println(second);

        int maxDays = c.getActualMaximum(Calendar.DAY_OF_MONTH);//获取月份的最大天数
        System.out.println(maxDays);
    }
}
```

综合练习

日历制作

```java
package com.cyx.date;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Calendar;
import java.util.List;

public class MyCalendar {
    /**
     * 日历星期
     */
```

```
    private static final String[] WEEKS ={
      "一","二","三","四","五","六","日"
    };
    /**
     * 日历总天数
     */
    private static final int TOTAL_DAYS = 42;

    private List<DayInfo> getDisplayDayInfos(int year, int month){
        List<DayInfo> days = new ArrayList<>(TOTAL_DAYS);
        Calendar c = Calendar.getInstance();//构建日历对象，默认是系统当前时间
        c.set(Calendar.YEAR, year);//修改年份为给定的年份
        c.set(Calendar.MONTH, month); //修改月份给定的月份
        //获取当前月显示的天数
        int currentDisplayDays =
c.getActualMaximum(Calendar.DAY_OF_MONTH);
        c.set(Calendar.DAY_OF_MONTH, 1);//设置天数为给定月份的第一天，方便获取星
期
        int week = c.get(Calendar.DAY_OF_WEEK);//获取给定月份第一天是周几
        int lastDisplayDays = week - 2; //计算上一个月显示天数
        if(lastDisplayDays < 0){//如果显示天数 <0，说明该月第一天是周日
            lastDisplayDays = 6;
        }
        c.roll(Calendar.MONTH, -1);//将月份-1，方便计算上一个月的最大天数
        //获取上一个月的最大天数
        int lastMonthDays = c.getActualMaximum(Calendar.DAY_OF_MONTH);
        //计算上一个月开始显示的天数
        int lastMonthStart = lastMonthDays - lastDisplayDays + 1;
        //将上一个月显示天数添加至集合中
        for(int i=lastMonthStart; i <= lastMonthDays; i++){
            days.add(new DayInfo(i, false));
        }
        //将当前月显示天数添加至集合中
        for(int i=1; i<=currentDisplayDays;i++){
            days.add(new DayInfo(i, true));
        }
        //计算下一个月显示天数
        int nextDisplayDays = TOTAL_DAYS - lastDisplayDays -
currentDisplayDays;
        //将下一个月显示天数添加至集合中
        for(int i=1; i<=nextDisplayDays; i++){
            days.add(new DayInfo(i, false));
        }
        return days;
    }


    private List<Integer> getDisplayDays(int year, int month){
        List<Integer> days = new ArrayList<>(TOTAL_DAYS);
        Calendar c = Calendar.getInstance();//构建日历对象，默认是系统当前时间
        c.set(Calendar.YEAR, year);//修改年份为给定的年份
        c.set(Calendar.MONTH, month); //修改月份给定的月份
        //获取当前月显示的天数
        int currentDisplayDays =
c.getActualMaximum(Calendar.DAY_OF_MONTH);
        c.set(Calendar.DAY_OF_MONTH, 1);//设置天数为给定月份的第一天，方便获取星
期
        int week = c.get(Calendar.DAY_OF_WEEK);//获取给定月份第一天是周几
```

```
        int lastDisplayDays = week - 2; //计算上一个月显示天数
        if(lastDisplayDays < 0){//如果显示天数 <0，说明该月第一天是周日
            lastDisplayDays = 6;
        }
        c.roll(Calendar.MONTH, -1);//将月份-1，方便计算上一个月的最大天数
        //获取上一个月的最大天数
        int lastMonthDays = c.getActualMaximum(Calendar.DAY_OF_MONTH);
        //计算上一个月开始显示的天数
        int lastMonthStart = lastMonthDays - lastDisplayDays + 1;
        //将上一个月显示天数添加至集合中
        for(int i=lastMonthStart; i <= lastMonthDays; i++){
            days.add(i);
        }
        //将当前月显示天数添加至集合中
        for(int i=1; i<=currentDisplayDays;i++){
            days.add(i);
        }
        //计算下一个月显示天数
        int nextDisplayDays = TOTAL_DAYS - lastDisplayDays -
currentDisplayDays;
        //将下一个月显示天数添加至集合中
        for(int i=1; i<=nextDisplayDays; i++){
            days.add(i);
        }
        return days;
    }

    public void showCalendarDayInfo(int year, int month){
        Arrays.stream(WEEKS).forEach(week-> System.out.print(week +
"\t"));
        System.out.println();
        List<DayInfo> days = getDisplayDayInfos(year, month);
        for(int i=0; i<days.size(); i++){
            DayInfo dayInfo = days.get(i);
            dayInfo.show();
            if(i%7 == 6){
                System.out.println();
            }
        }
    }

    public void showCalendar(int year, int month){
        Arrays.stream(WEEKS).forEach(week-> System.out.print(week +
"\t"));
        System.out.println();
        List<Integer> days = getDisplayDays(year, month);
        for(int i=0; i<days.size(); i++){
            if(i%7 == 6){
                System.out.println(days.get(i));
            } else {
                System.out.print(days.get(i) + "\t");
            }
        }
    }
}

package com.cyx.date;

```

```java
public class DayInfo {

    private int day;//天数

    private boolean currentMonthDay;//是否是当前月的天数

    public DayInfo(int day, boolean currentMonthDay) {
        this.day = day;
        this.currentMonthDay = currentMonthDay;
    }

    public void show(){
        if(currentMonthDay){
            System.out.print(day + "\t");
        } else {
            System.err.print(day + "\t");
        }
        try {
            Thread.sleep(5L);
        } catch (InterruptedException e) {
        }
    }
}

package com.cyx.date;

public class MyCalendarTest {

    public static void main(String[] args) {
        MyCalendar calendar = new MyCalendar();
        calendar.showCalendar(2015, 2);
        calendar.showCalendarDayInfo(2015, 2);
    }
}
```