常用API、Lambda、 常见算法

黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌

今天同学们需要学会什么

如何处理时间	认识包装类	正则表达式	经常需要操作数组元素	Lambda表达式
开发中经常与时间打交道，如何获取时间，进行时间计算等。	Java认为一切皆对象，集合、泛型也不支持基本数据类型，都需要应用包装类。	程序中需要对各种信息进行格式合法性的校验：如手机号码、邮箱等	如何快捷操作数组元素，如输出数组内容、排序、元素搜索等。排序和搜索算法是什么样的？	匿名内部类的代码是可以进一步的简化的，具体的简化规则是什么样的？

目录

Contents

Date 类概述

- Date类代表当前所在系统的日期时间信息。

Date的构造器

名称	说明
`public Date()`	创建一个Date对象，代表的是系统当前此刻日期时间。

Date的常用方法

名称	说明
public long getTime()	返回从1970年1月1日　00:00:00走到此刻的总的毫秒数

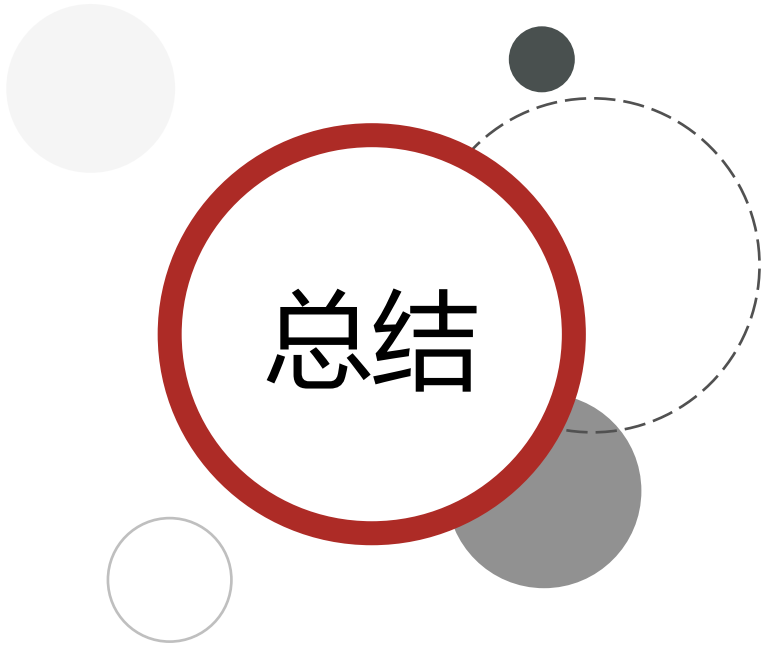
案例

- 请计算出当前时间往后走1小时121秒之后的时间是多少。

时间毫秒值 -> 日期对象

构造器	说明
public Date(long time)	把时间毫秒值转换成Date日期对象。

Date方法	说明
public void setTime(long time)	设置日期对象的时间为当前时间毫秒值对应的时间



1、日期对象如何创建，如何获取时间毫秒值？

- **public Date();**
- **public long getTime();**

2、时间毫秒值怎么恢复成日期对象

- **public Date(long time);**
- **public void setTime(long time);**

目录

Contents

当前获取时间的方式

```
public static void main(String[] args) {
        // 1、创建当前日期时间对象
        Date d = new Date();
        System.out.println(d);

        // 2、得到当前时间的毫秒值
        long time = d.getTime();
        System.out.println(time);
}
```

控制台输出：

```
Tue Jan 04 12:39:17 CST 2022
1641271157743
```

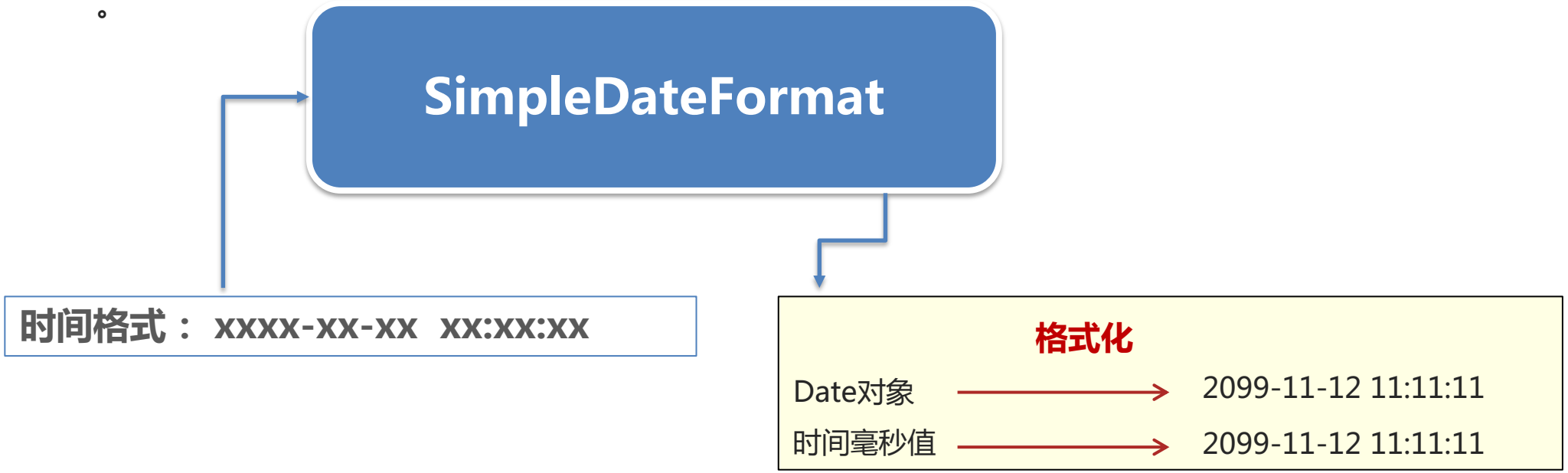
？？？

```
2022-01-04 12:39:17
```

SimpleDateFormat

- 代表简单日期格式化，可以用来把日期时间格式化成为我们想要的形式

。



SimpleDateFormat

- 代表简单日期格式化，可以用来把日期时间格式化成为我们想要的形式

。

构造器

构造器	说明
public SimpleDateFormat(String pattern)	创建简单日期格式化对象，并封装格式化的形式信息

格式化方法

格式化方法	说明
public final String **format(Date date)**	将日期格式化成日期/时间字符串
public final String **format(Object time)**	将时间毫秒值式化成日期/时间字符串

格式化的时间形式的常用的模式对应关系如下：

- y 年
- M 月
- d 日
- H 时
- m 分
- s 秒

2020-11-11 13:27:06 ——— yyyy-MM-dd HH:mm:ss

2020年11月11日 13:27:06 ——— yyyy年MM月dd日 HH:mm:ss

格式化

Date对象 → 2099-11-12 11:11:11

时间毫秒值 → 2099-11-12 11:11:11

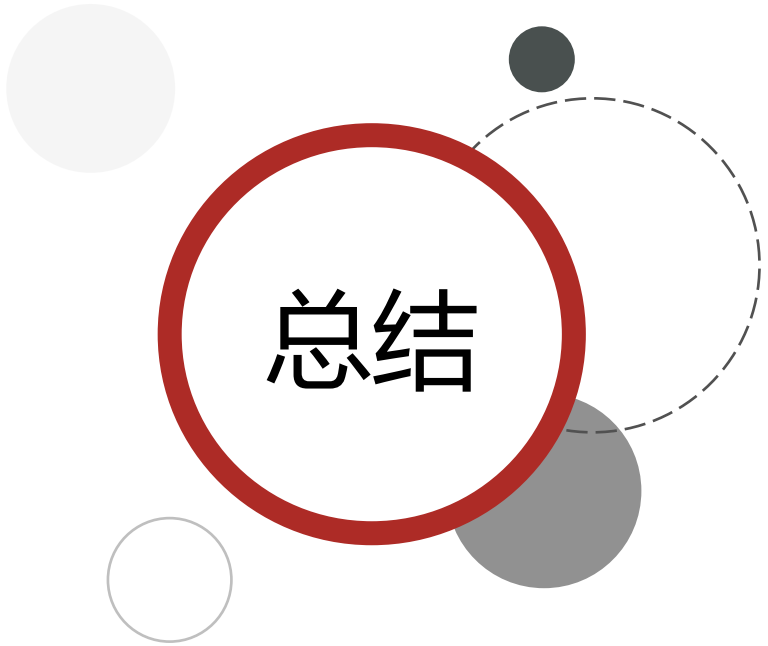
需求2

解析字符串时间成为日期对象 ？？？

2022-01-04 12:30:30 -> Date日期对象

SimpleDateFormat解析字符串时间成为日期对象

解析方法	说明
public Date **parse(String source)**	从给定字符串的开始解析文本以生成日期



1、SimpleDateFormat代表什么，有什么作用？

- **简单日期格式化对象**
- **可以把日期对象及时间毫秒值格式化成我们想要的字符串形式。**
- **可以把字符串的时间形式解析成Date日期对象。**

2、SimpleDateFormat的对象如何创建？

- **public SimpleDateFormat(String pattern)**

3、SimpleDateFormat格式化，以及解析时间的方法是怎么样的？

- **public final String format(Date d):格式化日期对象**
- **public final String format(Object time):格式化时间毫秒值**
- **public Date parse(String source)：解析字符串时间**

练习 秒杀活动

需求

- 某购物网站举办秒杀活动，开始时间和结束时间如左图所示，当前活动结束后，系统记录到2位用户的付款时间分别如下：
  - 小贾下单并付款的时间为：2020年11月11日 0:03:47
  - 小皮下单并付款的时间为：2020年11月11日 0:10:11
- 规则：顾客的付款时间必须在秒杀时间之内，请判断出两位顾客是否秒杀成功。

分析

- 把4个字符串形式的时间解析成日期对象。
- 判断小贾和小皮的时间是否在秒杀时间范围之内，并给出相应的提示。

目录

Contents

Calendar概述

- Calendar代表了系统此刻日期对应的日历对象。
- Calendar是一个抽象类，不能直接创建对象。

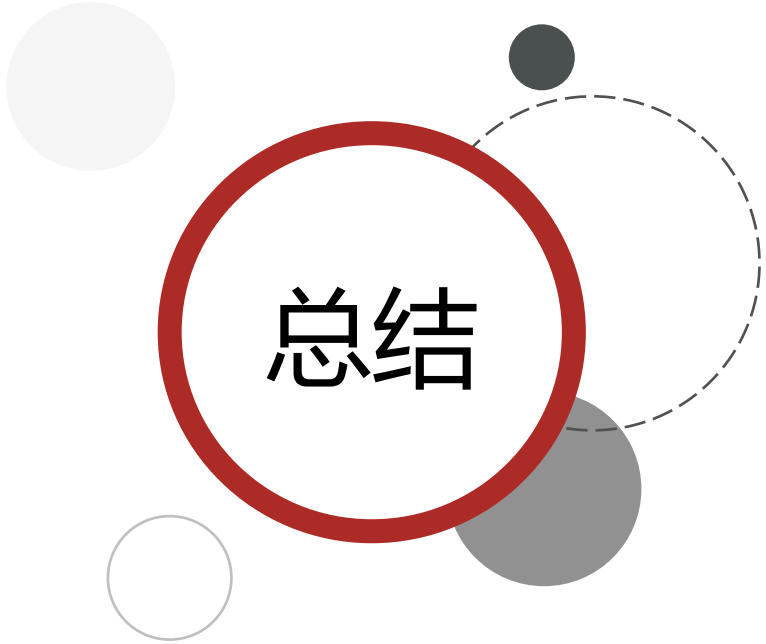
Calendar日历类创建日历对象的方法：

方法名	说明
`public static Calendar getInstance()`	获取当前日历对象

Calendar常用方法

方法名	说明
public int get(int field)	取日期中的某个字段信息。
public void set(int field,int value)	修改日历的某个字段信息。
public void add(int field,int amount)	为某个字段增加/减少指定的值
public final Date getTime()	拿到此刻日期对象。
public long getTimeInMillis()	拿到此刻时间毫秒值

注意：calendar是可变日期对象，一旦修改后其对象本身表示的时间将产生变化。



1、Calendar如何去得到日历对象的？

public static Calendar getInstance() ： 获取当前日历对象

目录

Contents

概述

- 从Java 8开始，java.time包提供了新的日期和时间API，主要涉及的类型有：

JDK8新增日期类

- **LocalDate**：不包含具体时间的日期。
- **LocalTime**：不含日期的时间。
- **LocalDateTime**：包含了日期及时间。
- **Instant**：代表的是时间戳。
- **DateTimeFormatter** 用于做时间的格式化和解析的
- **Duration**:用于计算两个“时间”间隔
- **Period**:用于计算两个“日期”间隔

- 新增的API严格区分了时刻、本地日期、本地时间，并且，对日期和时间进行运算更加方便。
- 其次，新API的类型几乎全部是不变类型（和String的使用类似），可以放心使用不必担心被修改。

LocalDate、LocalTime、LocalDateTime

- 他们 分别表示日期，时间，日期时间对象，他们的类的实例是不可变的对象。
- 他们三者构建对象和API都是通用的

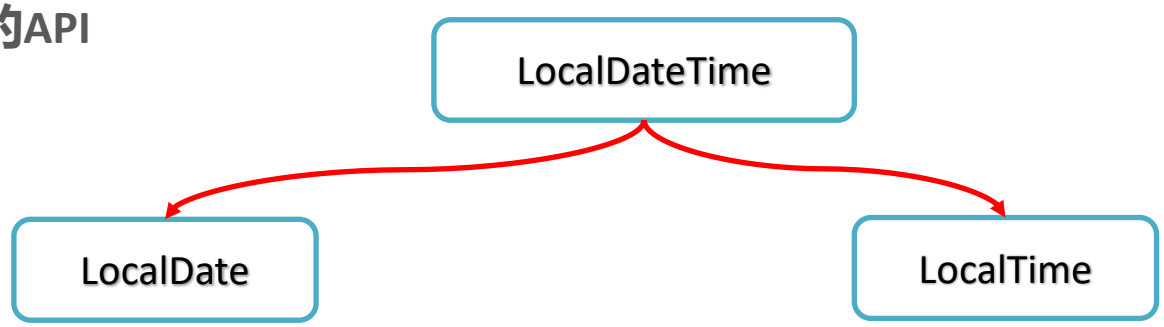
构建对象的方式如下：

方法名	说明	
public static Xxxx now();	静态方法，根据当前时间创建对象	LocaDate localDate = LocalDate.now(); LocalTime llocalTime = LocalTime.*now*(); LocalDateTime localDateTime = LocalDateTime.*now*();
public static Xxxx of(...);	静态方法，指定日期/时间创建对象	LocalDate localDate1 = LocalDate.*of*(2099 , 11,11); LocalTime localTime1 = LocalTime.*of*(11, 11, 11); LocalDateTime localDateTime1 = LocalDateTime.*of*(2020, 10, 6, 13, 23, 43);

LocalDate、LocalTime、LocalDateTime获取信息的API.

方法名	说明
public int geYear()	获取年
public int getMonthValue()	获取月份（1-12）
Public int getDayOfMonth()	获取月中第几天乘法
Public int getDayOfYear()	获取年中第几天
Public DayOfWeek getDayOfWeek()	获取星期

转换相关的API



LocalDateTime的转换API

方法名	说明
public LocalDate toLocalDate()	转换成一个LocalDate对象
public LocalTime toLocalTime()	转换成一个LocalTime对象

修改相关的API

- LocalDateTime 综合了 LocalDate 和 LocalTime 里面的方法，所以下面只用 LocalDate 和 LocalTime 来举例。
- 这些方法返回的是一个新的实例引用，因为LocalDateTime 、LocalDate 、LocalTime 都是不可变的。

方法名	说明
plusDays, plusWeeks, plusMonths, plusYears	向当前 LocalDate 对象添加几天、 几周、几个月、几年
minusDays, minusWeeks, minusMonths, minusYears	从当前 LocalDate 对象减去几天、 几周、几个月、几年
withDayOfMonth, withDayOfYear, withMonth, withYear	将月份天数、年份天数、月份、年 份 修 改 为 指 定 的 值 并 返 回 新 的 LocalDate 对象
isBefore, isAfter	比较两个 LocalDate

目录

Contents

Instant时间戳

- JDK8获取时间戳特别简单，且功能更丰富。Instant类由一个静态的工厂方法now()可以返回当前时间戳。

```
Instant instant = Instant.now();
System.out.println("当前时间戳是: " + instant);

Date date = Date.from(instant);
System.out.println("当前时间戳是: " + date);

instant = date.toInstant();
System.out.println(instant);
```

- 时间戳是包含日期和时间的，与java.util.Date很类似，事实上Instant就是类似JDK8 以前的Date。
- Instant和Date这两个类可以进行转换。

目录

Contents

DateTimeFormatter

- 在JDK8中，引入了一个全新的日期与时间格式器DateTimeFormatter。
- 正反都能调用format方法。

```
LocalDateTime ldt = LocalDateTime.now();
System.out.println(ldt);//2021-03-01T15:09:17.444190900


DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
String ldtStr = ldt.format(dtf);
System.out.println(ldtStr);//2021-03-01 15:09:17


String ldtStr1 = dtf.format(ldt);
System.out.println(ldtStr1);//2021-03-01 15:09:17
```

目录

Contents

Period

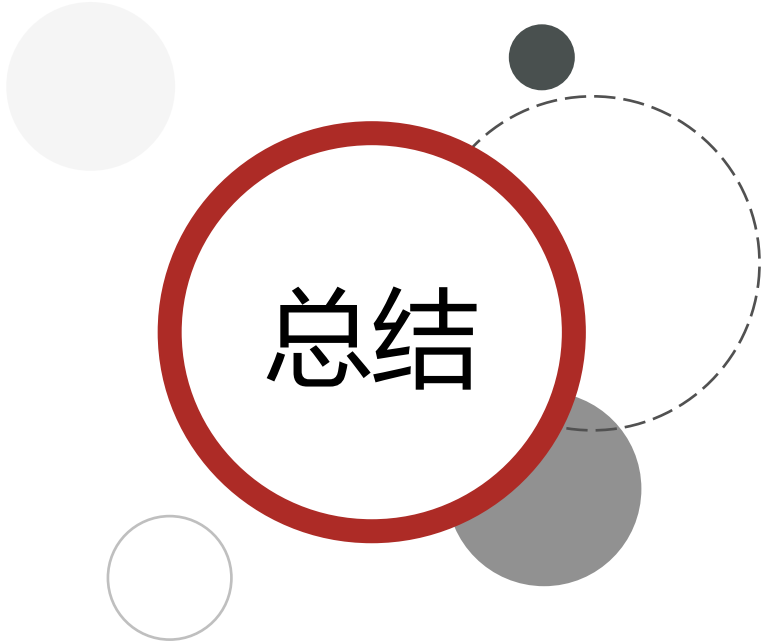
- 在Java8中，我们可以使用以下类来计算**日期间隔**差异：java.time.Period
- 主要是 Period 类方法 getYears()，getMonths() 和 getDays() 来计算,只能精确到年月日。
- 用于 LocalDate 之间的比较。

```
LocalDate today = LocalDate.now();

System.out.println(today);     // 2021-03-01

LocalDate birthDate = LocalDate.of(1995, 1, 11);

System.out.println(birthDate); // 1995-01-11

Period period = Period.between(birthDate, today);

System.out.printf("年龄 : %d 年 %d 月 %d 日", period.getYears(), period.getMonths(),

period.getDays());
```

Duration

- 在Java8中，我们可以使用以下类来计算时间间隔差异：java.time.Duration
- 提供了使用基于时间的值测量时间量的方法。
- 用于 LocalDateTime 之间的比较。也可用于 Instant 之间的比较。

```
LocalDateTime today = LocalDateTime.now();
System.out.println(today);
LocalDateTime birthDate = LocalDateTime.of(1990,10,1,10,50,30);
System.out.println(birthDate);

Duration duration = Duration.between(birthDate, today);//第二个参数减第一个参数
System.out.println(duration.toDays());//两个时间差的天数
System.out.println(duration.toHours());//两个时间差的小时数
System.out.println(duration.toMinutes());//两个时间差的分钟数
System.out.println(duration.toMillis());//两个时间差的毫秒数
System.out.println(duration.toNanos());//两个时间差的纳秒数
```



1、 Duration: 用于计算两个“时间”间隔。

2、 Period: 用于计算两个“日期”间隔。

目录

Contents

java.time.temporal.ChronoUnit

- ChronoUnit类可用于在单个时间单位内测量一段时间，这个工具类是最全的了，可以用于比较所有的时间单位

```
LocalDateTime today = LocalDateTime.now();
System.out.println(today);
LocalDateTime birthDate = LocalDateTime.of(1990,10,1,10,50,30);
System.out.println(birthDate);

System.out.println("相差的年数：" + ChronoUnit.YEARS.between(birthDate, today));
System.out.println("相差的月数：" + ChronoUnit.MONTHS.between(birthDate, today));
System.out.println("相差的周数：" + ChronoUnit.WEEKS.between(birthDate, today));
System.out.println("相差的天数：" + ChronoUnit.DAYS.between(birthDate, today));
System.out.println("相差的时数：" + ChronoUnit.HOURS.between(birthDate, today));
System.out.println("相差的分数：" + ChronoUnit.MINUTES.between(birthDate, today));
System.out.println("相差的秒数：" + ChronoUnit.SECONDS.between(birthDate, today));
System.out.println("相差的毫秒数：" + ChronoUnit.MILLIS.between(birthDate, today));
System.out.println("相差的微秒数：" + ChronoUnit.MICROS.between(birthDate, today));
System.out.println("相差的纳秒数：" + ChronoUnit.NANOS.between(birthDate, today));

System.out.println("相差的半天数：" + ChronoUnit.HALF_DAYS.between(birthDate, today));
System.out.println("相差的十年数：" + ChronoUnit.DECADES.between(birthDate, today));
System.out.println("相差的世纪（百年）数：" + ChronoUnit.CENTURIES.between(birthDate, today));
System.out.println("相差的千年数：" + ChronoUnit.MILLENNIA.between(birthDate, today));
System.out.println("相差的纪元数：" + ChronoUnit.ERAS.between(birthDate, today));
```

目录

Contents

包装类

- 其实就是8种基本数据类型对应的引用类型。

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

为什么提供包装类？

- Java为了实现一切皆对象，为8种基本类型提供了对应的引用类型。
- 后面的集合和泛型其实也只能支持包装类型，不支持基本数据类型。

自动装箱：基本类型的数据和变量可以直接赋值给包装类型的变量。

自动拆箱：包装类型的变量可以直接赋值给基本数据类型的变量。

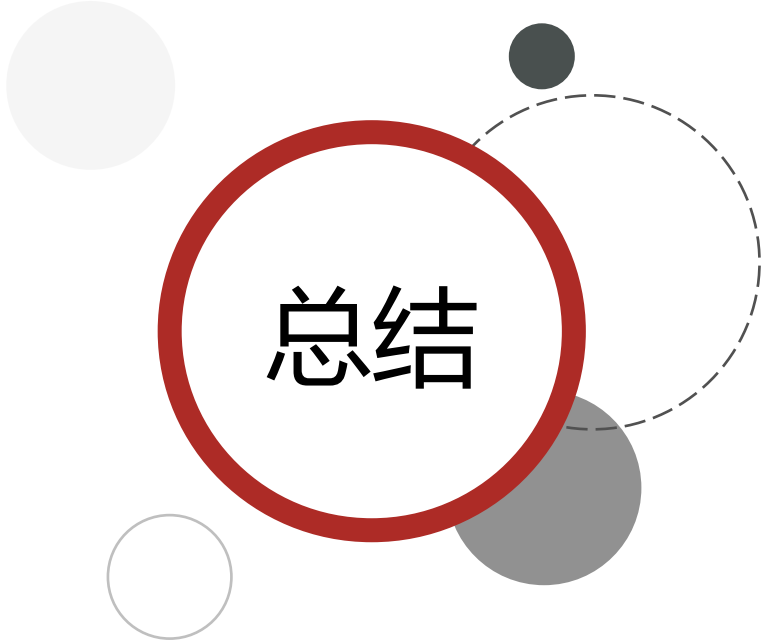
包装类的特有功能

- 包装类的变量的默认值可以是null，容错率更高。
- 可以把基本类型的数据转换成字符串类型(用处不大)

① 调用toString()方法得到字符串结果。
② 调用Integer.toString(基本类型的数据)。

- 可以把字符串类型的数值转换成真实的数据类型（真的很有用）

① Integer.parseInt(“字符串类型的整数”)
② Double.parseDouble(“字符串类型的小数”)。



1、包装类是什么，作用是什么？

- **基本数据类型对应的引用类型，实现了一切皆对象。**
- **后期集合和泛型不支持基本类型，只能使用包装类。**

2、包装类有哪些特殊功能？

- **可以把基本类型的数据转换成字符串类型(用处不大)**
- **可以把字符串类型的数值转换成真实的数据类型（真的很有用）**

目录

Contents

正则表示

- 正则表达式可以用一些规定的字符来制定规则，并用来校验数据格式的合法性。

正则表达式初体验

- 需求：假如现在要求校验一个qq号码是否正确，6位及20位之内，必须全部是数字 。
- 先使用目前所学知识完成校验需求；然后体验一下正则表达式检验。

目录

Contents

字符串对象提供了匹配正则表达式的方法

public boolean matches (String regex)：判断是否匹配正则表达式，匹配返回true，不匹配返回false。

字符类(默认匹配一个字符)

[abc]	只能是a, b, 或c
[^abc]	除了a, b, c之外的任何字符
[a-zA-Z]	a到z A到Z，包括（范围）
[a-d[m-p]]	a到d，或m通过p：（[a-dm-p]联合）
[a-z&&[def]]	d, e, 或f(交集)
[a-z&&[^bc]]	a到z，除了b和c：（[ad-z]减法）
[a-z&&[^m-p]]	a到z，除了m到p：（[a-lq-z]减法）

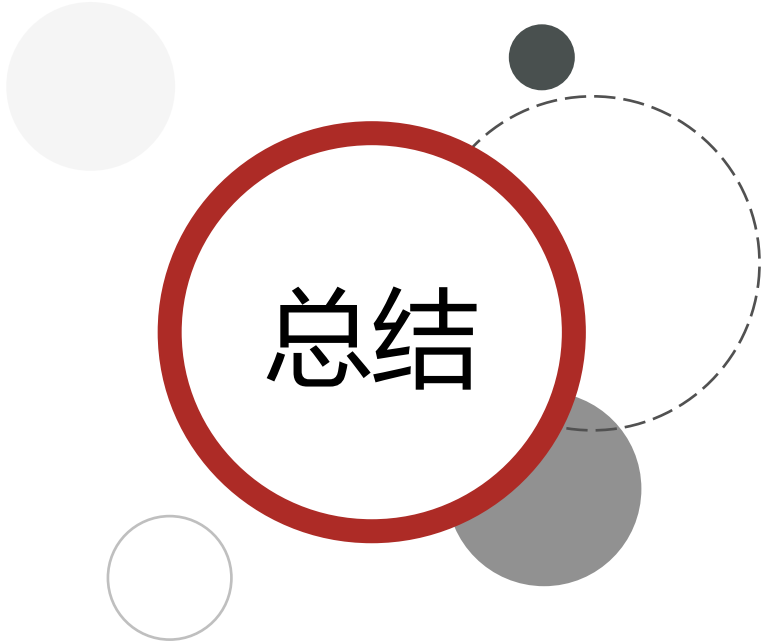
预定义的字符类(默认匹配一个字符)

.	任何字符
\d	一个数字：[0-9]
\D	非数字：[^0-9]
\s	一个空白字符：[\t\n\x0B\f\r]
\S	非空白字符：[^\s]
\w	[a-zA-Z_0-9] 英文、数字、下划线
\W	[^\w] 一个非单词字符

贪婪的量词（配合匹配多个字符）

X?	X，一次或根本不
X*	X，零次或多次
X+	X，一次或多次
X {n}	X，正好n次
X {n, }	X，至少n次
X {n,m}	X，至少n但不超过m次

```
System.out.println("a".matches("[abc]")); // true
System.out.println("z".matches("[abc]")); // false
System.out.println("ab".matches("[abc]")); // false
System.out.println("ab".matches("[abc]+")); //true
```



1、String类的哪个方法可以与正则表达式进行匹配。

public boolean matches(String regex):

判断是否匹配正则表达式，匹配返回true，不匹配返回false。

目录

Contents

案例 请使用正则表达式完成如下需求

需求

① 请编写程序模拟用户输入手机号码、验证格式正确，并给出提示，直到格式输入正确为止。

② 请编写程序模拟用户输入邮箱号码、验证格式正确，并给出提示，直到格式输入正确为止。

③ 请编写程序模拟用户输入电话号码、验证格式正确，并给出提示，直到格式输入正确为止。

分析

- 定义方法，接收用户输入的数据，使用正则表达式完成检验，并给出提示。

目录

Contents

正则表达式在字符串方法中的使用

方法名	说明
public String replaceAll(String regex,String newStr)	按照正则表达式匹配的内容进行替换
public String[] split(String regex)：	按照正则表达式匹配的内容进行分割字符串，反回一个字符串数组。

目录

Contents

正则表达式支持爬取信息

```
String rs = "来黑马程序学习Java,电话020-43422424，或者联系邮箱" +
        "itcast@itcast.cn,电话18762832633，0203232323" +
        "邮箱bozai@itcast.cn，400-100-3233 ，4001003232";
// 需求：从上面的内容中爬取出 电话号码和邮箱。
// 1.定义爬取规则
String regex = "(\\w{1,}@\\w{2,10}(\\.\\w{2,10}){1,2})|" +
        "(1[3-9]\\d{9})|(0\\d{2,5}-?\\d{5,15})|400-?\\d{3,8}-?\\d{3,8}";
// 2.编译正则表达式成为一个匹配规则对象
Pattern pattern = Pattern.compile(regex);
// 3.通过匹配规则对象得到一个匹配数据内容的匹配器对象
Matcher matcher = pattern.matcher(rs);
// 4.通过匹配器去内容中爬取出信息
while(matcher.find()){
    System.out.println(matcher.group());
}
```

```
020-43422424
itcast@itcast.cn
18762832633
0203232323
bozai@itcast.cn
400-100-3233
4001003232
```

目录

Contents

Arrays类概述

- 数组操作工具类，专门用于操作数组元素的。

Arrays类的常用API

方法名	说明
public static String toString(类型[] a)	返回数组的内容（字符串形式）
public static void sort(类型[] a)	对数组进行默认升序排序
public static <T> void sort(类型[] a, Comparator<? super T> c)	使用比较器对象自定义排序
public static int binarySearch(int[] a, int key)	二分搜索数组中的数据，存在返回索引，不存在返回-1

目录

Contents

Arrays类的排序方法

方法名	说明
public static void sort(类型[] a)	对数组进行默认升序排序
public static <T> void sort(类型[] a, Comparator<? super T> c)	使用比较器对象自定义排序

自定义排序规则

- **设置Comparator接口对应的比较器对象，来定制比较规则。**

如果认为左边数据 大于 右边数据 返回正整数

如果认为左边数据 小于 右边数据 返回负整数

如果认为左边数据 等于 右边数据 返回0

目录

Contents

冒泡排序的思想

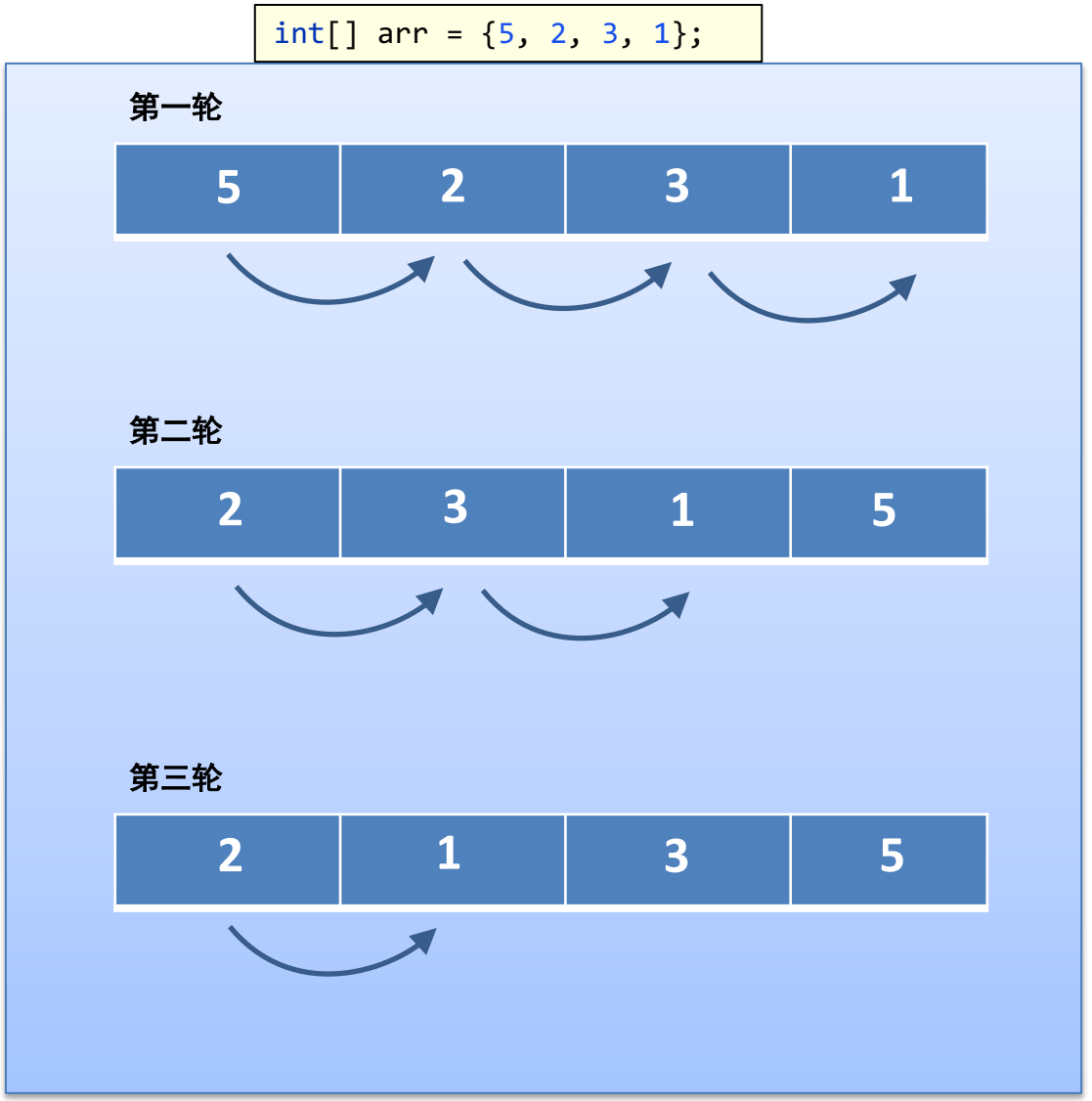
- 每次从数组中找出最大值放在数组的后面去。

实现冒泡排序的关键步骤分

- 确定总共需要做几轮： 数组的长度-1.
- 每轮比较几次:

i(轮数)	次数	**次数规律: 数组的长度 - i**
1	3	
2	2	
3	1	

- 当前位置大于后一个位置则交换数据



目录

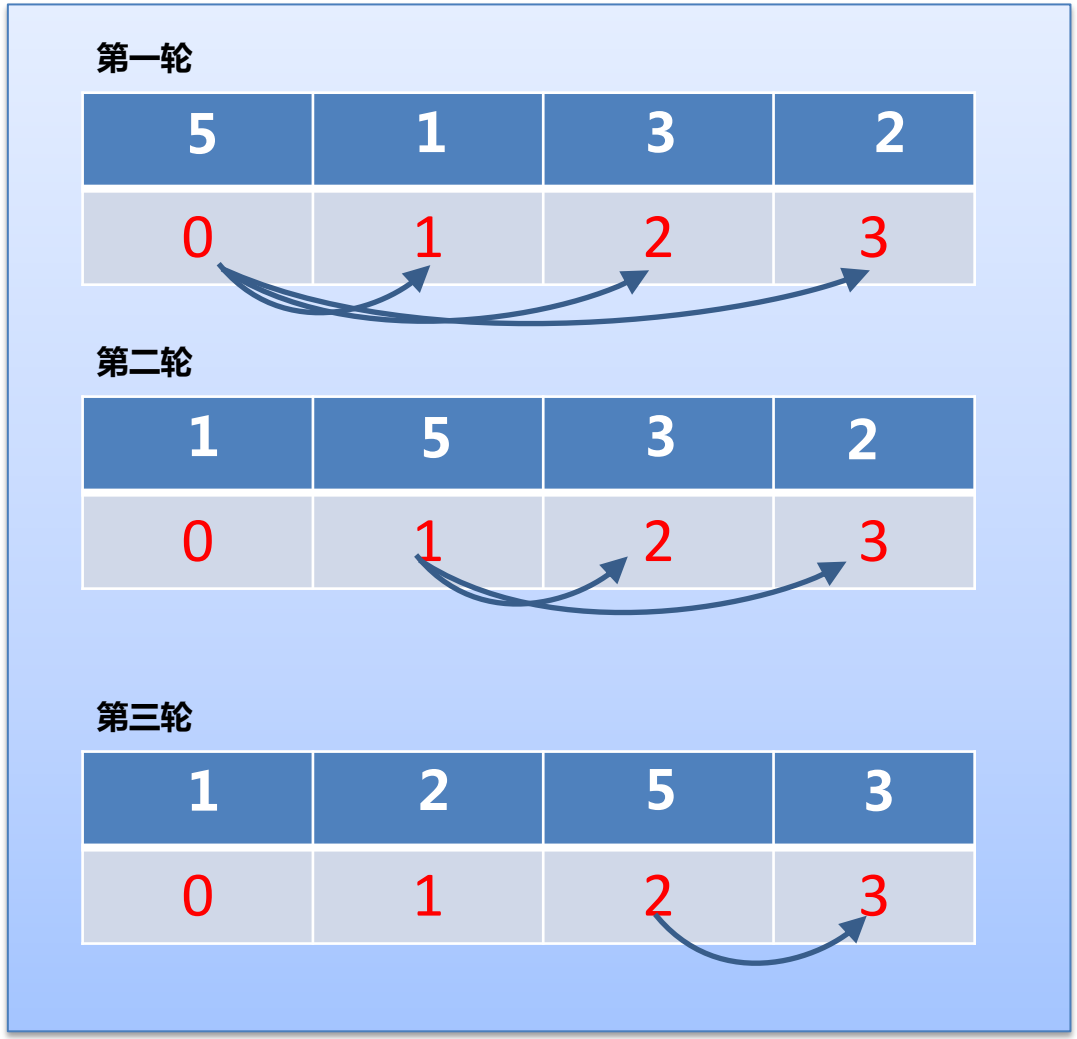
Contents

选择排序的思想

- 每轮选择当前位置，开始找出后面的较小值与该位置交换

选择排序的关键

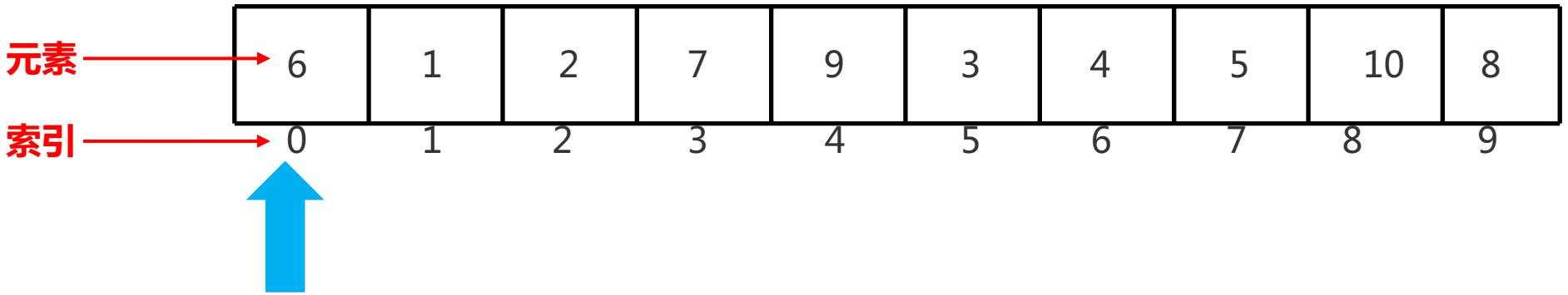
- 确定总共需要选择几轮： 数组的长度-1.
- 控制每轮从以前位置为基准，与后面元素选择几次。



目录

Contents

基本查找

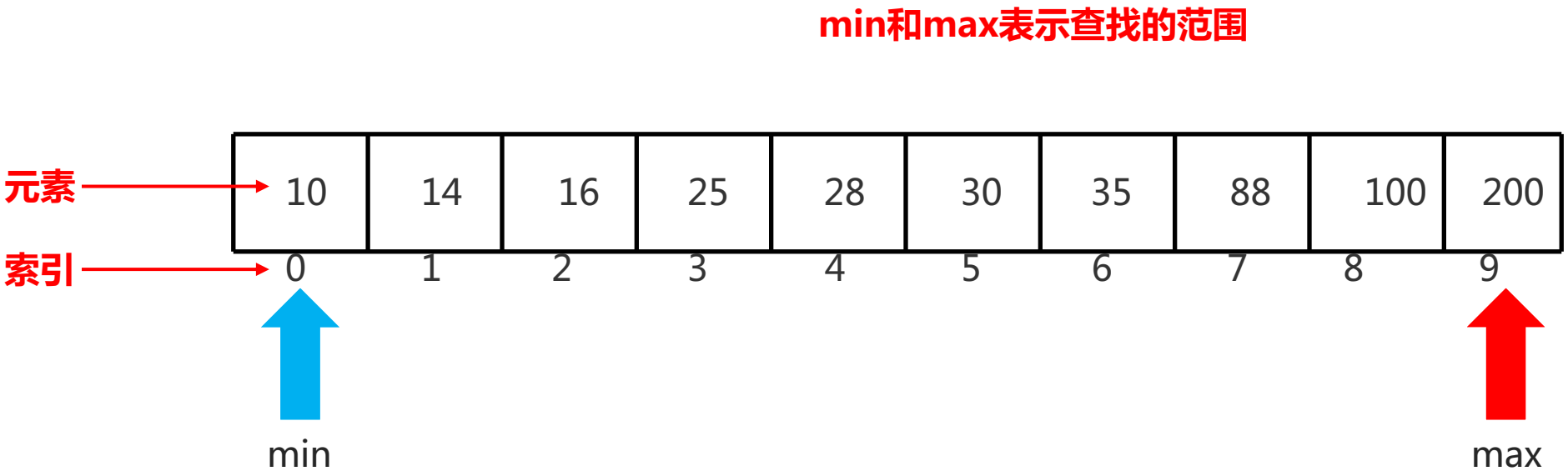


需求：我要查找数组中的3在哪个索引？

结论：在数据量特别大的时候，基本查找从前往后寻找的性能是很差的！

二分查找

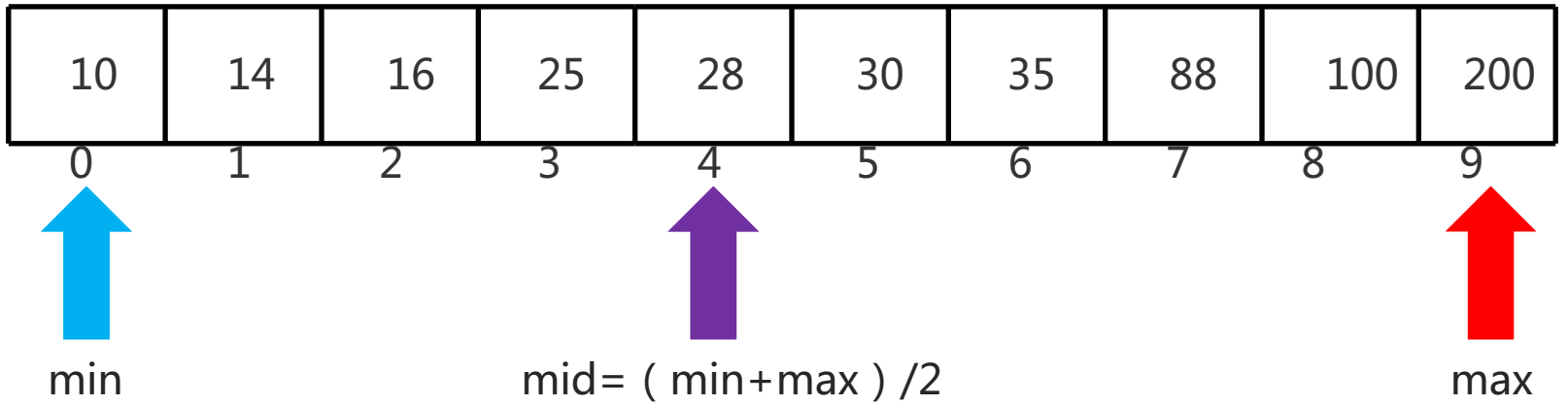
- 二分查询性能好，二分查找的前提是必须是排好序的数据。



需求：我要查找数组中的16在哪个索引？

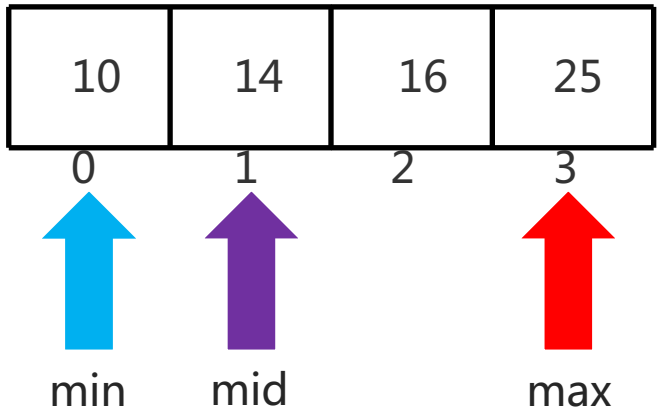
二分查找

- 二分查找性能好，二分查找的前提是必须是排好序的数据。



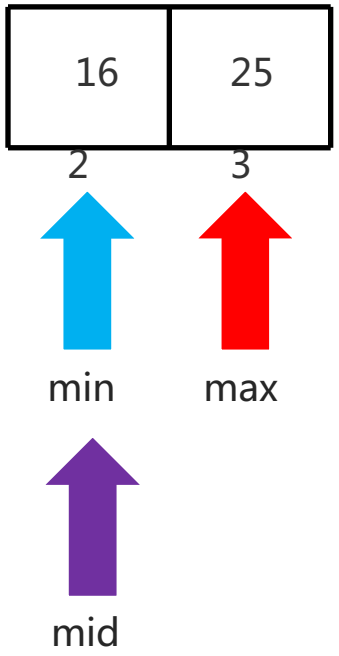
二分查找

- 二分查找性能好，二分查找的前提是必须是排好序的数据。
- **二分查找相当于每次去掉一半的查找范围**

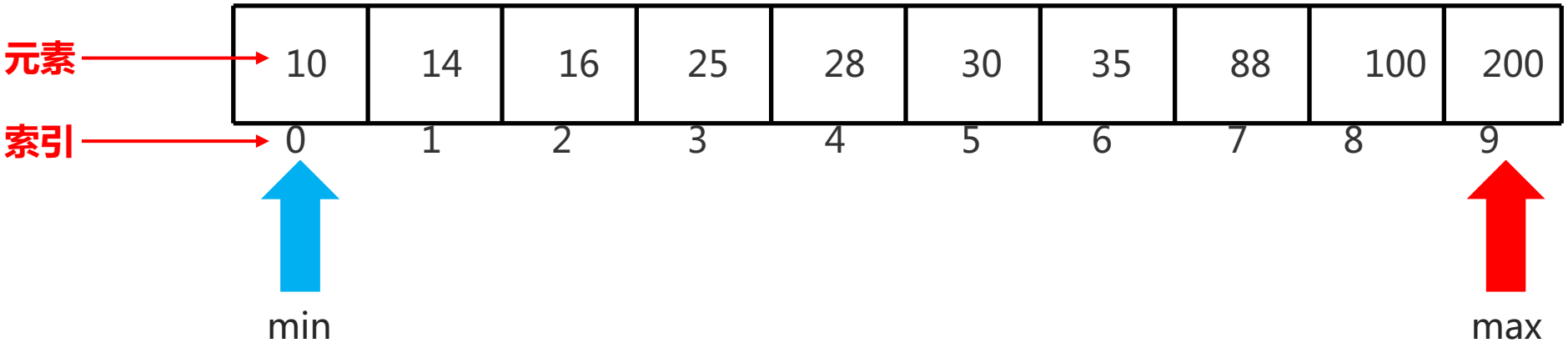


二分查找

- 二分查找性能好，二分查找的前提是必须是排好序的数据。
- **二分查找相当于每次去掉一半的查找范围**

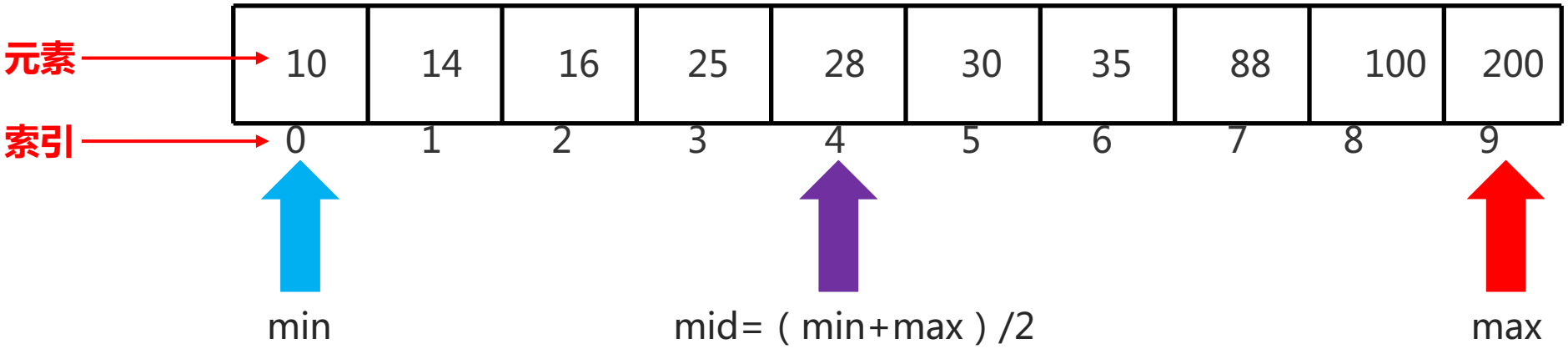


二分查找 --- 元素不存在

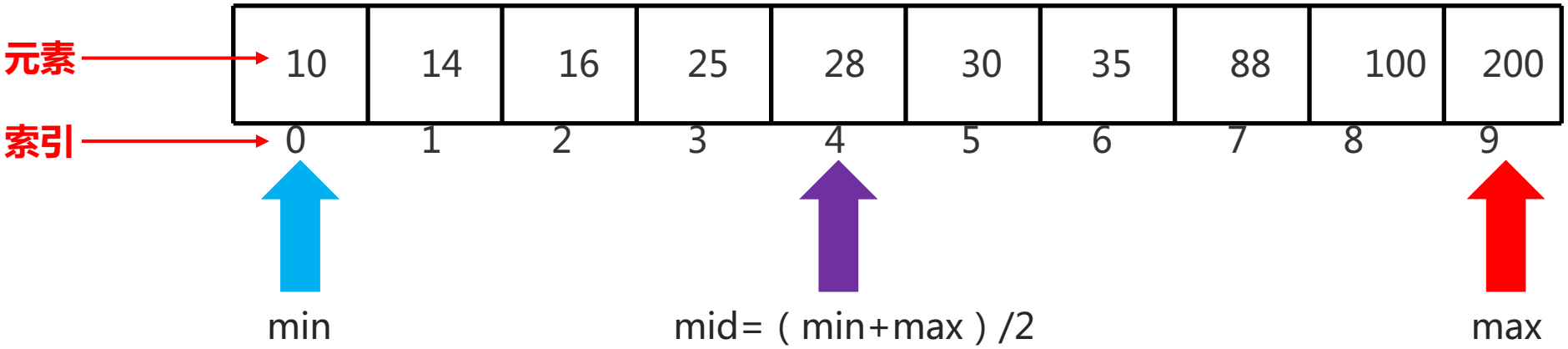


需求：我要查找数组中的300在哪个索引？

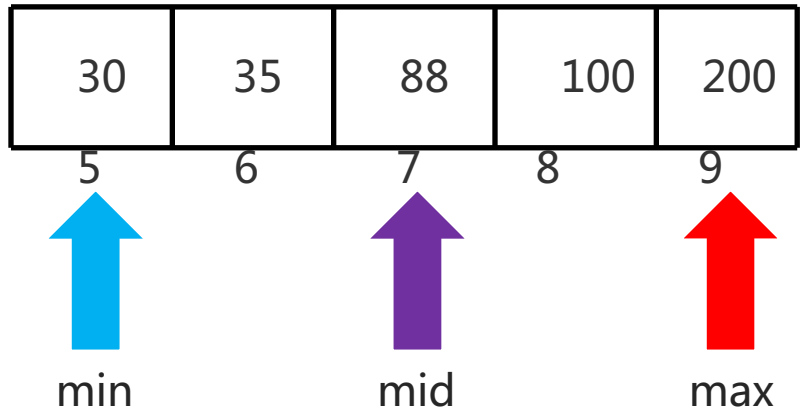
二分查找 --- 元素不存在


元素
索引
10
14
16
25
28
30
35
88
100
200
0
1
2
3
4
5
6
7
8
9
min
mid=（min+max）/2
max

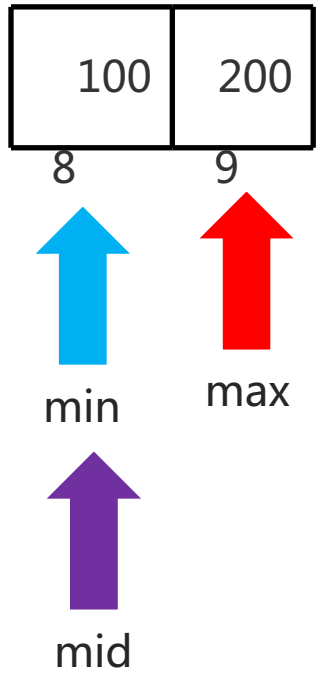
二分查找 --- 元素不存在


元素
索引
10
14
16
25
28
30
35
88
100
200
0
1
2
3
4
5
6
7
8
9
min
mid=（min+max）/2
max

二分查找 --- 元素不存在



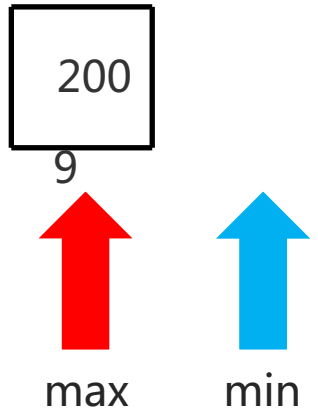
二分查找 --- 元素不存在

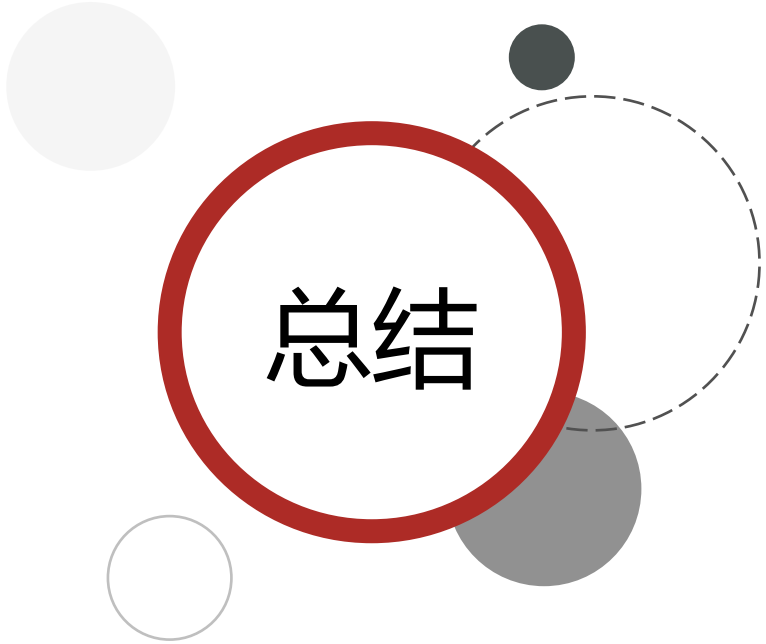


二分查找 --- 元素不存在

二分查找 --- 元素不存在

结论：二分查找正常的检索条件应该是开始位置min <= 结束位置max





1、数组的二分查找的实现步骤是什么样的?

- **定义变量记录左边和右边位置。**
- **使用while循环控制查询（条件是左边位置<=右边位置）**
- **循环内部获取中间元素索引**
- **判断当前要找的元素如果大于中间元素，左边位置=中间索引+1**
- **判断当前要找的元素如果小于中间元素，右边位置=中间索引-1**
- **判断当前要找的元素如果等于中间元素，返回当前中间元素索引。**

目录

Contents

Lambda概述

- Lambda表达式是JDK 8开始后的一种新语法形式。
- **作用：简化匿名内部类的代码写法。**

Lambda表达式的简化格式

```
(匿名内部类被重写方法的形参列表) -> {
   被重写方法的方法体代码。
}
注：-> 是语法形式，无实际含义
```

注意：Lambda表达式只能简化函数式接口的匿名内部类的写法形式

什么是函数式接口？

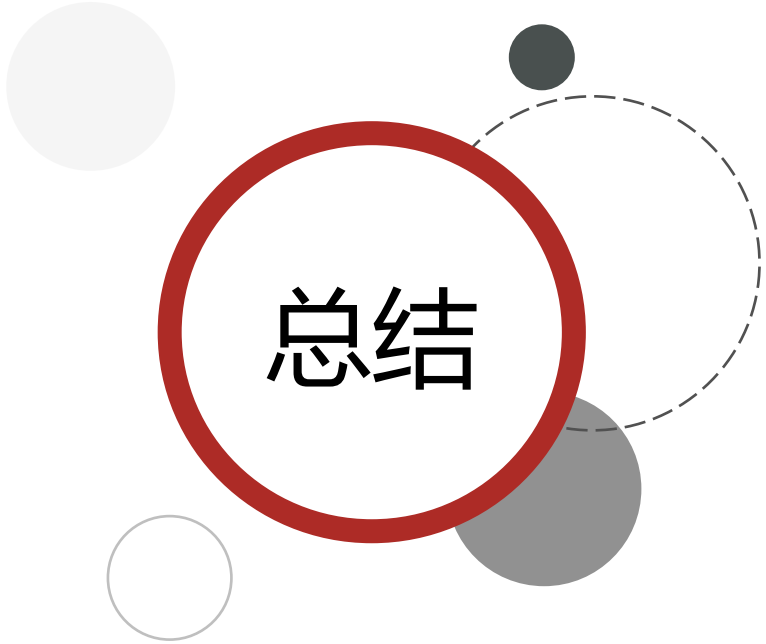
- 首先必须是接口、其次接口中有且仅有一个抽象方法的形式

体验Lambda表达式

```
public class LambdaDemo1 {
    public static void main(String[] args) {

        goSwimming( new Swimming() {
            @Override
            public void swim() {
                System.out.println("铁汁，我们去游泳吧~");
            }
        } );
    }

    public static void goSwimming(Swimming swimming) {
        swimming.swim();
    }
}
```

```
public class LambdaDemo1 {
    public static void main(String[] args) {

        goSwimming( () -> {
          System.out.println("铁汁，我们去游泳吧~") } );

    }

    public static void goSwimming(Swimming swimming) {
        swimming.swim();
    }
}
```

代码更少，关注点更加明确了



1、Lambda表达式的基本作用？

- **简化函数式接口的匿名内部类的写法。**

2、Lambda表达式有什么使用前提？

- **必须是接口的匿名内部类，接口中只能有一个抽象方法**

目录

Contents

Lambda表达式简化Comparator接口的匿名形式

```
public static void main(String[] args) {
    Integer[] ages = {66, 33, 22, 55, 88};

    Arrays.sort(ages, new Comparator<Integer>(){
        @Override
        public int compare(Integer o1, Integer o2) {
            return o2 - o1;
        }
    });

    System.out.println("内容:" + Arrays.toString(ages));
}
```

```
Arrays.sort(ages, (Integer o1, Integer o2) -> {
        return o2 - o1;
});
```

注意：通常我们见到的函数式接口上都有一个@FunctionalInterface注解，

标记该接口必须是满足函数式接口。

Lambda表达式简化按钮监听器ActionListener的匿名内部类形式

```
JButton btn = new JButton("登录");

// 给登录按钮绑定点击事件监听器
btn.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println("登录一下~~~");
    }
});
```

```
btn.addActionListener( (ActionEvent e) -> {
        System.out.println("登录一下~~~");
});
```

目录

Contents

Lambda表达式的省略写法（进一步在Lambda表达式的基础上继续简化）

- 参数类型可以省略不写。
- 如果只有一个参数，参数类型可以省略，同时()也可以省略。
- 如果Lambda表达式的方法体代码只有一行代码。可以省略大括号不写,同时要省略分号！
- 如果Lambda表达式的方法体代码只有一行代码。可以省略大括号不写。此时，如果这行代码是return语句，必须省略return不写，同时也必须省略"

传智教育旗下高端IT教育品牌