3.1 常见类的概述和使用

3.1.1 API的使用和常用包的概述

1. java.lang包
    - java语言的核心包，该包的所有内容由java虚拟机自动导入。
    - 如System类，String类...
2. java.util包
    - java语言的工具包，里面提供了大量的工具类以及集合类等。
    - 如Scanner类，Random类，List集合...
3. java.io包
    - java语言的输入输出包，里面提供了大量读写文件相关的类。
4. java.net包
    - java语言的网络包，里面提供了大量网络编程相关的类。
    - 如ServerSocket类...
5. java.sql包
    - java语言的数据包，里面提供了大量操作数据库的类和接口等。
    - 如DriverManager类，Connection类

3.1.2 Object类的概念和构造方法

Object类是所有类的父类

Object类中的方法：

1. Boolean equals(Object obj)
2. int hashcode()
3. String toString()
4. getClass()

3.1.3 equals方法默认功能的使用

Boolean equals(Object obj)

1. 用于判断调用对象是否与参数对象相等
2. 比较对象地址

3.1.4 equals方法的重写实现

因为equals方法比较内存地址是否相等，所以通常我们会重写equals方法去比较对象的内容。

```
@Override
public boolean equals(Object obj){
  //先判断obj的对象类型
  if(obj instanceof Student){
    //类型转换
    Student st = (Student)obj
    return this.getId() == st.getId();
  }else{
    ...
  }
}
```

3.1.5 equals方法的重写优化

```
@Override
public boolean equals(Object obj){
  //先判断obj是否为空
  if(obj == null){
    return false;
  }

  //当调用对象等于参数对象
  if(this == obj){
    return true;
  }

  //先判断obj的对象类型
  if(obj instanceof Student){
    //类型转换
    Student st = (Student)obj
    return this.getId() == st.getId();
  }else{
    return false;
  }
}
```

3.1.6 hashCode方法的重写实现

重写equals方法后，通常还需要重写hashCode方法。保持equals方法和hashCode方法一致性。

3.1.7 toString方法的重写实现

包名.类名@哈希码的十六进制

当使用`print`或`println`打印引用或字符串拼接引用时自动调用`toString`方法。

3.1.11 Integer类的概念和构造方法

java5之前：

Integer.valueOf(int num): 将int转换成Integer类型

Integer.valueOf(String num): 将String转换成Integer类型

3.1.12 Integer类的装箱和拆箱机制

装箱：将int转换成Integer类型

拆箱：将Integer转换成int类型

从java5开始增加了自动装箱和自动拆箱

注意⚠️：

```
Integer num1 = 127;
Integer num2 = 127;
System.out.println(num1 == num2); //true
System.out.println(num1.equals(num2));//true
```

自动装箱池：将-128到127之间的整数已经填装完毕，当程序中使用该范围之间的整数时，无需装箱直接使用自动装箱池中的对象即可。

3.1.17 包装类的使用总结

1. 装箱：调用包装类的静态方法，即**包装类.valueOf()**
2. 拆箱：调用包装类的xxxValue()方法
3. 字符串转基本数据类型：调用包装类的parseXxx()方法

3.1.19 BigDecimal类的概念和使用

用于精确计算，商业开发

3.1.20 BigInteger类的概念和使用

希望表示比long还大的数据

3.2 String类的概述和使用

3.2.1 String类和常量池的概念

1. java.lang.String
2. String类由final关键字修饰，不能被继承
3. 从jdk1.9开始，该类的底层不使用char[]来储存数据，而是改成byte[]加编码标记，从而节约了一些空间。
4. 该类描述的字符串内容是个常量不可改变

常量池：由于String类型描述的字符串内容是不可变的，因此java虚拟机会将第一次出现的字符串放入常量池中，若后续代码中用到相同的字符串则直接使用常量池中已有的字符串，无需申请内存及创建对象，从而提高了性能

3.2.2 String类常用构造方法的使用

3.2.3 String类的笔试考点

1. 请问下面的代码会创建几个对象？分别存放在哪里？

```
String str1 = "hello" //一个对象，存在常量池
String str2 = new String("hello"); //两个对象，一个在常量池，一个在堆区
```

2. 常量优化

```
String str1 = "abcd";
String str2 = "ab"+"cd"; //常量优化机制
System.out.println(str1 == str2); //true
```

```
String str1 = "ab";
String str2 = str1 +"cd";
System.out.println(str1 == str2); //false
```

3.2.5 String类中字符串的获取和使用

1. char charAt(int index)
2. int length()
3. boolean isEmpty()

3.2.7 String类实现字符串之间大小的比较

int compareTo(String str)

3.2.8 String类实现各种方法的使用

1. boolean contains(CharSequence chars) 区分大小写
2. String toUpperCase()
3. String toLowerCase()
4. String trim() 去除字符串两端的空格

3.2.10 String类实现字符和字符串的正向查找

int indexOf(String str) 返回第一次出现的索引位置

int indexOf(String str, int fromIndex)

int indexOf(int char)

int indexOf(int char, int fromIndex)

3.2.11 String类实现字符和字符串的反向查找

int lastIndexOf(String str) 返回最后一次出现的索引位置

int lastIndexOf(String str, int fromIndex)

int lastIndexOf(int char)

int lastIndexOf(int char, int fromIndex)

3.2.12 String类中子字符串的获取

String substring(int beginIndex)

String substring(int beginIndex, int endIndex)

3.2.13 正则表达式的概念和规则

^开头，$结尾

正则表达式	说明
[abc]	可要出现a，b，c中任意一个字符
[^abc]	可以出现任意一个字符，除了a，b，c
[a-zA-Z0-9]	可以出现a～z，A～Z，0～9中任意一个字符
.	任意一个字符（通常不包括换行符）
\d	任意一个数字字符，相当于[0-9]
\D	任意一个非数字字符
\s	空白字符
\S	非空白字符
\w	任意一个单词字符，相当于[a-zA-Z_0-9]
\W	任意一个非单词字符
X?	表示X可以出现一次或一次也没有，也就是0～1次
X*	表示X可以出现零次或多次，也就是0～n次
X+	表示X可以出现一次或多次，也就是1～n次
X{n}	表示X可以出现恰好n次
X{n,}	表示X可以出现至少n次，也就是>=n次
X{n,m}	表示X可以出现至少n次，但不超过m次，也就是$n <= X <= m$

Boolean matches(String regex)

3.2.16 正则表达式相关的方法使用

1. String[] split(String regex)
2. String replace(char oldChar, char newChar) 替换所有匹配字符
3. String replaceFirst(String regex, char newChar)
4. String replaceAll(String regex, char newChar)

3.3 可变字符串类和日期相关类

3.3.1 可变字符串类

java.lang.StringBuilder类和java.lang.StringBuffer类来**描述字符序列可以改变的字符串**

1. StringBuffer类是从jdk1.0开始存在，属于线程安全的类，因此效率比较低
2. StringBuilder类是从jdk1.5开始存在，属于非线程安全的类，因此效率比较高

3.3.2 StringBuilder类的常用构造方法

方法声明	功能介绍
StringBuilder()	使用无参方式构造对象，默认容量为16，可以放16个字符
StringBuilder(int capacity)	根据参数指定的容量来构造对象，容量为参数指定大小
StringBuilder(String str)	根据参数指定的字符串来构造对象，容量为16+字符串长度

方法声明	功能介绍
int capacity()	返回调用对象的容量
int length()	返回字符串的长度，也就是字符的个数

3.3.3 StringBuilder类实现插入操作

方法声明	功能介绍
StringBuilder insert(int offset, String str)	插入字符，返回调用对象自身的引用，会改变原有字符串
StringBuilder append(String str)	末尾添加字符

3.3.4 StringBuilder类扩容算法的源码解析

原始容量*2 + 2

底层采用byte数组来存储所有的字符

3.3.5 StringBuilder类实现字符和字符串的删除

方法声明	功能介绍
StringBuilder deleteCharAt(int index)	删除字符
StringBuilder delte(int start, int end)	删除字符串

3.3.6 StringBuilder类实现字符和字符串的改查

方法声明	功能介绍
StringBuilder setChatAt(int index, char ch)	
StringBuilder repleace(int start, int end, String str)	
int indexOf(String str)	
int lastIndexOf(String str)	
StringBuilder reverse()	

3.3.7 字符串类的笔试考点 ⚠️

1. 既然StringBuilder类的对象本身可以修改，为什么还要返回值呢?

   为了连续调用

2. 如何实现StringBuilder类型和String类型之间的转换?

   ```
   String str = "this is a string";
   StringBuilder sb = new StringBuilder(str); // String -> StringBuilder

   StringBuilder sb = new StringBuilder("this is a string");
   String str = sb.toString(); // StringBuilder -> String
   ```

3. String, StringBuilder, StringBuffer的效率

   StringBuilder > StringBuffer > String

3.3.8 System类的概念和使用

1. java.lang.System
2. System.out/System.in
3. static long currentTimeMillis(): 返回当前时间与1970年1月1日0时0分0秒时间以毫秒为单位的时间差。可以用来测试代码的执行效率

3.3.9 Date类的概念和使用

java.util.Date类主要用于描述特定的时间，年月日时分秒

方法声明	功能介绍
Date()	当前系统时间
Date(long date)	参数是距离1970年1月1日0时0分0秒的毫秒数
long getTime()	获取调用对象距离1970年1月1日0时0分0秒的毫秒
void setTime()	

3.3.10 SimpleDateFormat类的概念和使用

java.text.SimpleDateFormat类主要用于实现日期和文本之间的转换

方法声明	功能介绍
SimpleDateFormat()	
SimpleDateFormat(String pattern)	yyyy-MM-dd HH:mm:ss
Final String format(Date date)	
Date parse(String source)	

3.3.11 Calendar类的概念和使用

1. java.util.Calender类主要描述特定的时间，取代Date类中的过时方法实现全球化。
2. 该类是个抽象类，因此不能被实例化，其具体子类针对不同国家的日历系统。

方法声明	功能介绍
static Calender getInstance()	拿到Calendar类的引用
void set(int year, int month, int date, int hourOfDay, int minute, int second)	设置时间, 注意月份是从0开始，即0～11
Date getTime()	转换为Date类型

3.3.12 Calendar类的方法和多态的使用方法

笔试考点⚠️：

1. Calender类是抽象类，不能创建对象。为什么可以获取Calendar类型的引用?

由源码可知，返回的并不是Calendar类型的对象，而是Calendar类的子类GregorianCalendar等对象，形成多态。

2. 多态的使用场合
    - 在方法体中直接使用多态的语法格式

```
Shape shape = new Rectangle()
```

    - 通过方法的参数传递形成多态

```
public static void draw(Shape shape){
  shape.show()
}

draw(new Rectangle(1,2,3,4))
```

    - 通过方法的返回值类型形成多态

```
Calendar getInstance(){
  return new GregorianCalender(zone, aLocale)
}
```

3.3.14 日期时间对象的创建和特征获取

1. Java.time.LocalDate类主要用于描述年-月-日的日期信息，该类不表示时间和时区信息

方法声明	功能介绍
static LocalTime now()	从默认时区的系统中获取时间
static LocalTime now(ZoneId zone)	获取指定时区的当前时间

2. java.time.LocalTime类主要用于描述时间
3. java.time.LocalDateTime类主要用于描述年-月-日-时-分-秒信息

方法声明	功能介绍
static LocalDateTime now()	获取默认时区的系统时间
static LocalDateTime of(int year, int month, int date, int hourOfDay, int minute, int second)	设置时间

3.3.15 日期时间对象的特征操作

LocalDateTime类中有很多能对时间进行操作的方法。

3.3.16 Instant类的概念和使用（待看……）

3.3.17 DateTimeFormatter类的概念和使用

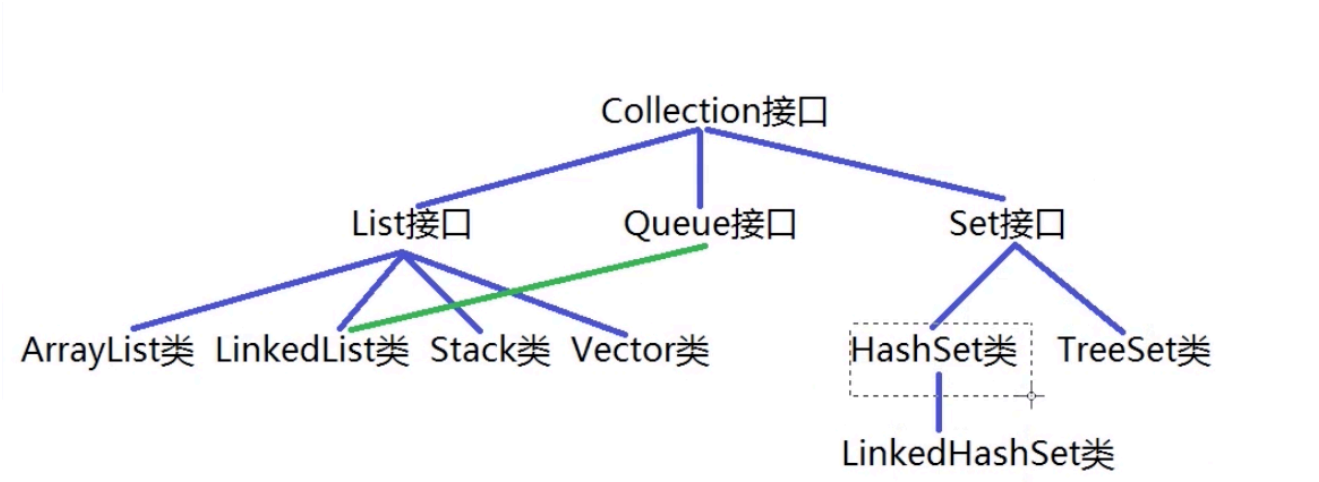
1. java.time.format.DateTimeFormatter类主要用于格式化和解析日期

方法声明	功能介绍
static DateTimeFormatter ofPattern(String pattern)	根据参数指定的格式来获取对象
String format(TemporalAccessor tempotal)	将参数指定日期时间转换为字符串
TemporalAccessor parese(CharSequence text)	将参数指定字符串转换为字符串日期时间

3.4 集合类库上

3.4.1 集合的概述

1. java.util.Collection集合和java.util.Map集合
2. 可以储存不同类型的多个元素



3.4.2 Collection集合的准备和元素添加

java.util.Collection接口是List接口，Queue接口以及Set接口的父接口。

方法声明	功能介绍
boolean add(E e)	
boolean addAll(Collection<? extends E> c)	

3.4.3 Collection集合判断单个元素是否存在

方法声明	功能介绍
boolean contains(Object o)	

contains(Object o)方法的工作原理：调用Objects.equals(o, e)，其中o代表contains方法的形式参数，e代表集合中的每个元素。而equals方法的工作原理：

```
public static boolean equals(Object a, Object b){
  return (a == b) || (a != null && a.equals(b))
}
```

3.4.4 Collection集合判断所有元素是否存在

方法声明	功能介绍
boolean containsAll(Collection<? extends E> c)	

3.4.5 Collection集合实现交集的计算

方法声明	功能介绍
boolean retainAll(Collection<? extends E> c)	发生改变返回true，否则返回false

3.4.6 Collection集合实现元素的删除

方法声明	功能介绍
boolean remove(Object o)	
boolean removeAll(Collection<? extends E> c)	

remove方法的工作原理与contains方法相似。

3.4.7 Collection集合实现其他方法的测试

方法声明	功能介绍
void clear()	
int size()	
boolean isEmpty()	
boolean equals(Object o)	
int hashCode()	
Object[] toArray()	
Iterator iterator()	

3.4.9 Collection集合实现迭代器的使用

方法声明	功能介绍
boolean hashNext()	判断集合中是否有可以访问的元素
E next()	指向下一个元素并取出当前元素
void remove()	

3.4.13 List集合的概念和ArrayList类的源码解析（重点）

1. java.util.List集合是Collection集合的子集合，该集合中允许出现有重复的元素并且有先后放入次序。
2. 该集合的主要实现类有：ArrayList类，LinkedList类，Stack类，Vector类（过时）。
3. ArrayList类的底层是采用动态数组进行数据管理的，支持下标访问，增删元素不方便。

源码分析：

3.4.14 LinkedList类的概念和源码解析（重点）

1. LinkedList类的底层采用双向链表进行数据管理，访问不方便，增删元素方便。

源码解析：

3.4.16 List集合中增加和查找方法的使用

方法声明	功能介绍
void add(int index, E element)	
E get(int index)	
int indexOf(E element)	
int lastIndexOf(E element)	

3.4.17 List集合中修改和删除以及子集合获取的使用

方法声明	功能介绍
E set(int index, E element)	
E remove(int index)	
List subList(int fromIndex, int toIndex)	

3.4.19 Queue集合的概念和使用

1. java.util.Queue集合是Collection集合的子集合，与List集合属于平级关系
2. 该集合的主要实现类是LinkedList类
3. 先进先出

3.5 集合类库下

3.5.1 泛型机制的基本概念

1. 通常情况下合集可以存放不同类型的对象，是因为将所有对象都看做Object类型放入的，因此从集合中取出元素时也是Object类型，为了表达该元素真实的数据类型，则需要强制强制类型转换，而强制类型转换可能会引发**类型转换异常**。
2. 为了避免类型转换异常，从java5开始增加泛型机制。
3. 泛型只在编译时期有效，在运行时期不区分时什么类型。

3.5.3 泛型机制的底层原理

泛型的本质就是参数化类型，也就是让数据类型作为参数传递，其中E相当于形式参数负责占位，而使用集合时<>中的数据类型相当于实际参数，用于给形势参数E进行初始化，从而使得集合中所有的E被实际参数替换，由于实际参数可以传递各种各样广泛的数据类型，因此得名泛型。

3.5.6 泛型方法的定义和使用

参数必须是泛型参数

举例：

```
public static <T> void printArray(T[] arr){

}
```

3.5.7 泛型通配符的使用和特点

<?> 可以传入任意类型的参数。不支持元素的添加，但支持元素的获取。

<? extends E> 表示类型的上界是E，只能是E或E的子类。不支持元素的添加，但支持元素的获取。

<? super E> 表示类型的下界是E，只能是E或E的父类。

3.5.9 Set集合的基本类型

1. java.util.Set集合是Collection集合的子集合，与List集合平级
2. 没有先后放入次序，且不允许重复
3. 该集合的主要实现类：HashSet类，TreeSet类，LinkedHashSet类

3.5.12 TreeSet集合的概念

二叉树是要指每个节点最多有两个子节点的树形结构。

有序二叉树：

1. 左子树中的任意节点都小于根节点
2. 右子树中的任意节点都大于根节点

TreeSet的底层数据结构是红黑树。红黑树是一种特殊的有序二叉树。

为了让插入的元素符合红黑树的规则，需要将新插入的元素与原有元素进行大小比较。有两种比较元素大小的方式：

1. 使用元素的自然排序规则行进比较并排序，让元素实现java.lang.Comparable接口。
2. 使用比较器规则进行比较并排序，构造TreeSet集合时传入java.util.Comparator接口。

3.5.14 TreeSet集合实现自然排序

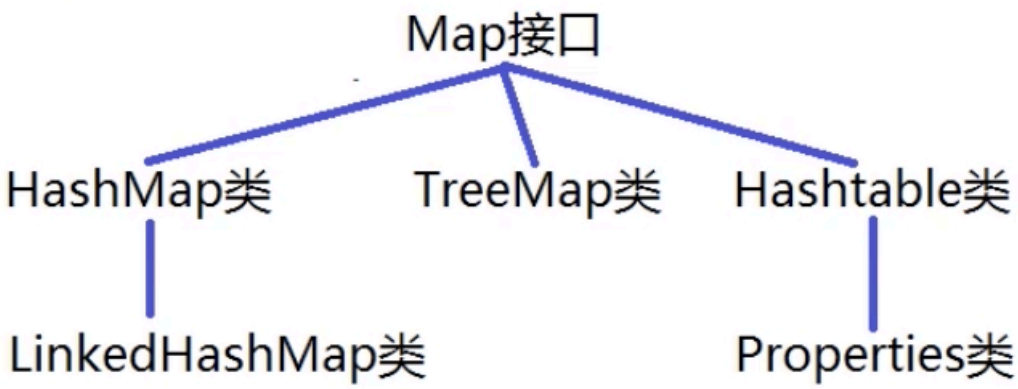
3.5.15 TreeSet集合实现比较器排序

匿名内部类：接口/父类类型 引用变量名 = new 接口/父类类型(){ 方法的重写 }

```
Comparator<Student> comparator = new Comparator<Student>(){
  @Override
  public int compare(Student o1, Student o2){
    return o1.getAge() - o2.getAge();
  }
}

Set<Student> s1 = new TreeSet<>(comparator);
```

3.5.16 Map集合的概念



1. java.util.Map<K, V> key-value pair 键值对
2. key不允许重复，而且一个key只能对应一value
3. 其中HashTable类是古老的Map实现类，与HashMap类相比属于线程安全的类，且不允许null作为key和value的值。

3.5.17 Map集合实现元素的增加和修改

方法声明	功能介绍
V put(K key， V value)	

3.5.18 元素放入HashMap集合的过程

1. DEFAULT_INITIAL_CAPACITY：HashMap的初始容量是16
2. DEFAULT_LOAD_FACTOR：HashMap的默认加载因子是0.75
3. threshold：扩容的临界值，该数值就是容量*加载因子，也就是12

3.5.19 Map集合实现元素的查找和删除操作

方法声明	功能介绍
V get(Object key)	
boolean containsKey(Object key)	
boolean containsValue(Object value)	
V remove(Object key)	

3.5.20 集合的三种遍历方式

方法声明	功能介绍
Set keySet()	返回包含键的Set视图
Collection values()	返回包含值的Set视图
Set<Map.Entry<K,V>> entrySet()	返回键值对

3.5.21 Collections类的编程使用

java.util.Collections类主要提供了对集合操作的静态方法

END