2020年12月7日 16:09

Day01

1.JDK,JRE,JVM三者之间的关系，以及JDK,JRE包含的主要结构有哪些?

JDK=JRE+java的开发工具（javac.exe,java.exe,javadoc.exe)

JRE=JVM+Java核心类库

2.为什么要配置path环境变量？如何配置?

我们希望在任何一个文件路径下都可以都可以去执行Java的开发工具。

Day02

应用程序=算法+数据结构。

Java语言的应用领域：>java web 开发：后台开发； >大数据开发； >Android应用程序开发（客户端的开发）。

Java语言的特点：

A. 面向对象性：

两个要素：类。对象

三个特征：封装，继承，多态

B. 健壮性：去除了c语言中的指针。 自动的垃圾回收机制—>仍然会出现内存溢出，内存泄漏。

C. 跨平台性：write once,run anywhere:一次编译，到处运行。

归功于JYM（Java虚拟机）

Java API:application programing interface;习惯上：将语言提供的类库都称为API。

API文档：针对于提供的类库如何使用，给的一个说明书。类似于《新华字典》。

/**********/

自动类型提升：

结论：当容量小的数据类型的变量与容量大的shu据类型的变量做运算时，结果自动提升为容量大的数据类型。

Byte、char、short-->int-->long-->float-->double

特别的：当byte，short，char三种类型的变量做运算时，结果为int型。

强制类型转换：自动提升运算的逆运算。

1. 需要使用强转符： ()
2. 注意点：强制类型转换，可能导致精度损失。

对于整数，有四种表示方式:

>二进制（binary）：0--1 满二进一。以0B或0b开头。

>十进制（decimal）：0--9，满十进一。

>八进制（octal）：0—7满八进一。以数字0开头表示。

>十六进制（hex）：0-9及A-F，满16进一。以0X或0x开头表示。此处的A-F不区分大小写。 如：0x21AF+1=0x21B0

Day03

区分&和&&:

//相同点1：&和&&的运算结果相同

//相同点2：当符号左边是true时，二者都会执行符号右边的运算。

//不同点：但符号左边是false时，&会继续实行符号右边的运算，&&不再执行符号右边的运算。

开发中推荐使用&&

区分|和||的区别:

//相同点1：|和||的运算结果相同

//相同点2：当符号左边是false时，二者都会执行符号右边的操作。

//不同点：当符号左边是true时，| 会继续执行符号右边的操作，||不再执行符号右边的操作。

开发中推荐使用||

位运算符：结论：

1. 位运算符操作的都是整型的数据
2. <<(左移):在一定范围内，每向左移一位，相当于乘以2;
3. >>(右移)：在一定范围内，每向右移一位，相当于除以2

Day04

产生随机数的闭区间的一个计算区间公式：【a,b】:(int)(Math.random()*(b-a+1)+a)

分支结构1：if-else

分支结构2：Switch-case:

1. 格式:

```
switch(表达式){
    case 常量1:
            执行语句1;
            Break;
    case 常量2:
            执行语句2;
            Break;
    .
    .
    .
    Dafault：（.....类似于else)
        执行语句n;
}
```

1.说明：根据switch表达式中的值，，一次匹配各个case中的常量，。一旦匹配成功，则进入相应的case结构中，调用其执行语句。当调用完成执行语句以后，则仍然继续向下执行其他case结构中的执行语句，直到遇到break关键字或者swit-case末尾结束为止。

2.break可以使用在switch-case结构中，表示一旦执行到此关键字，则跳出switch-case结构。

3.switch结构中的表达式，只能是如下的六种数据类型之一：byte、short、char、int、枚举类型、String类型。

Day05

Break和continue的区别和不同点：

Break用于switch-case，和循环结构，（结束当前循环），其后不可以声明执行语句。

Continue用于循环结构（结束当次循环），其后不可以声明执行语句。

break和continue关键字的使用

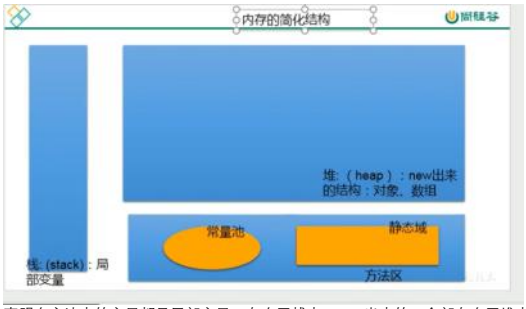
	使用范围	循环中使用的作用(不同点)	相同点
break:	switch-case		
	循环结构中	结束当前循环	关键字后面不能声明执行语句
continue:	循环结构中	结束当次循环	关键字后面不能声明执行语句

补充：带标签的break和continue的使用

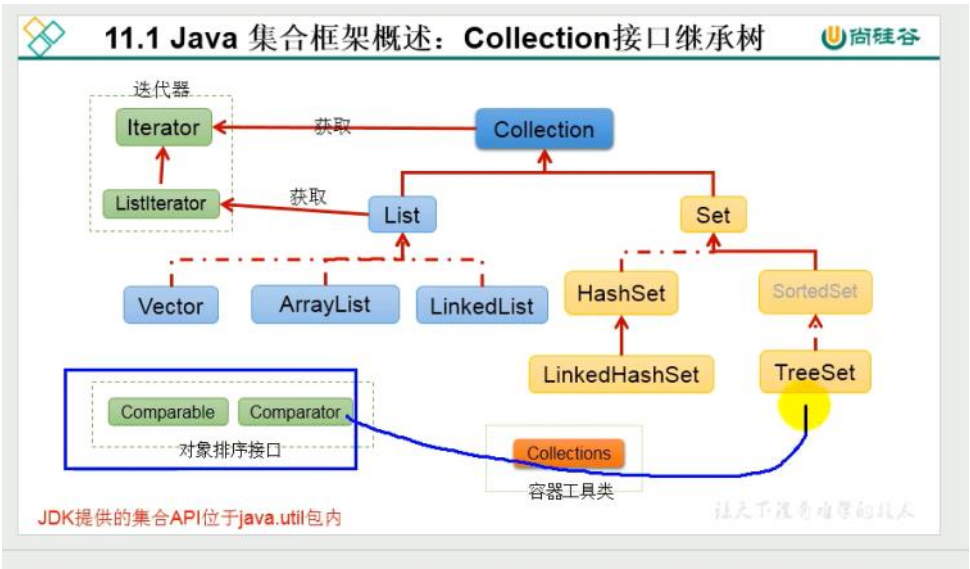
嵌套循环：

1. 内层循环遍历一遍，相当于外层循环执行了一次。
2. 假设外层循环执行了m次，内层循环执行了n次，则内层循环体一共执行了m*n次，
3. 外层循环执行行数，内层循环执行列数。

Day06



声明在方法中的变量都是局部变量，存在于栈中。new出来的，全部存在于堆中。



3. List 接口的常用方法有哪些？(增、删、改、查、插、长度、遍历)

add(Object obj)

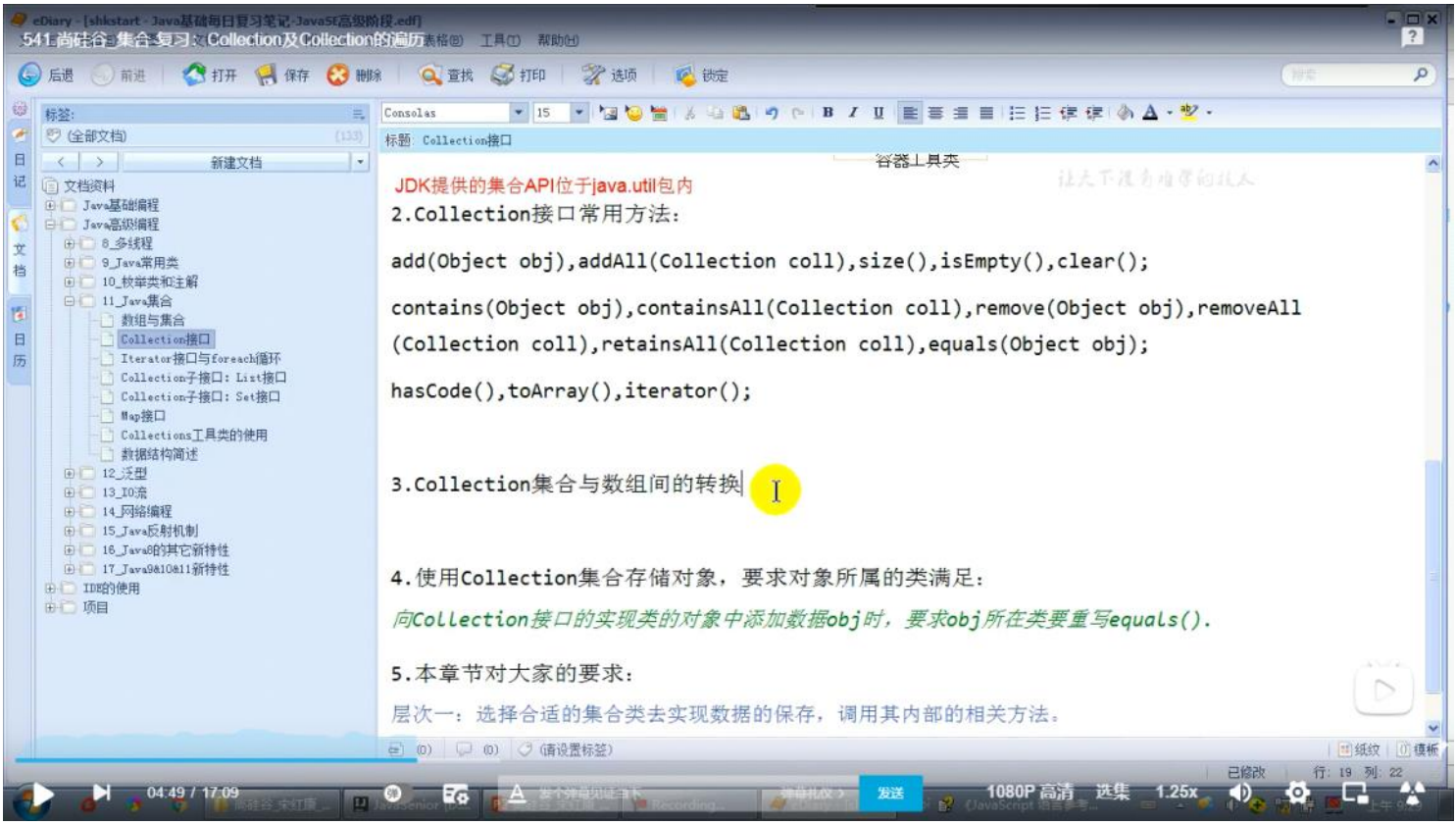
remove(Object obj)/remove(int index)

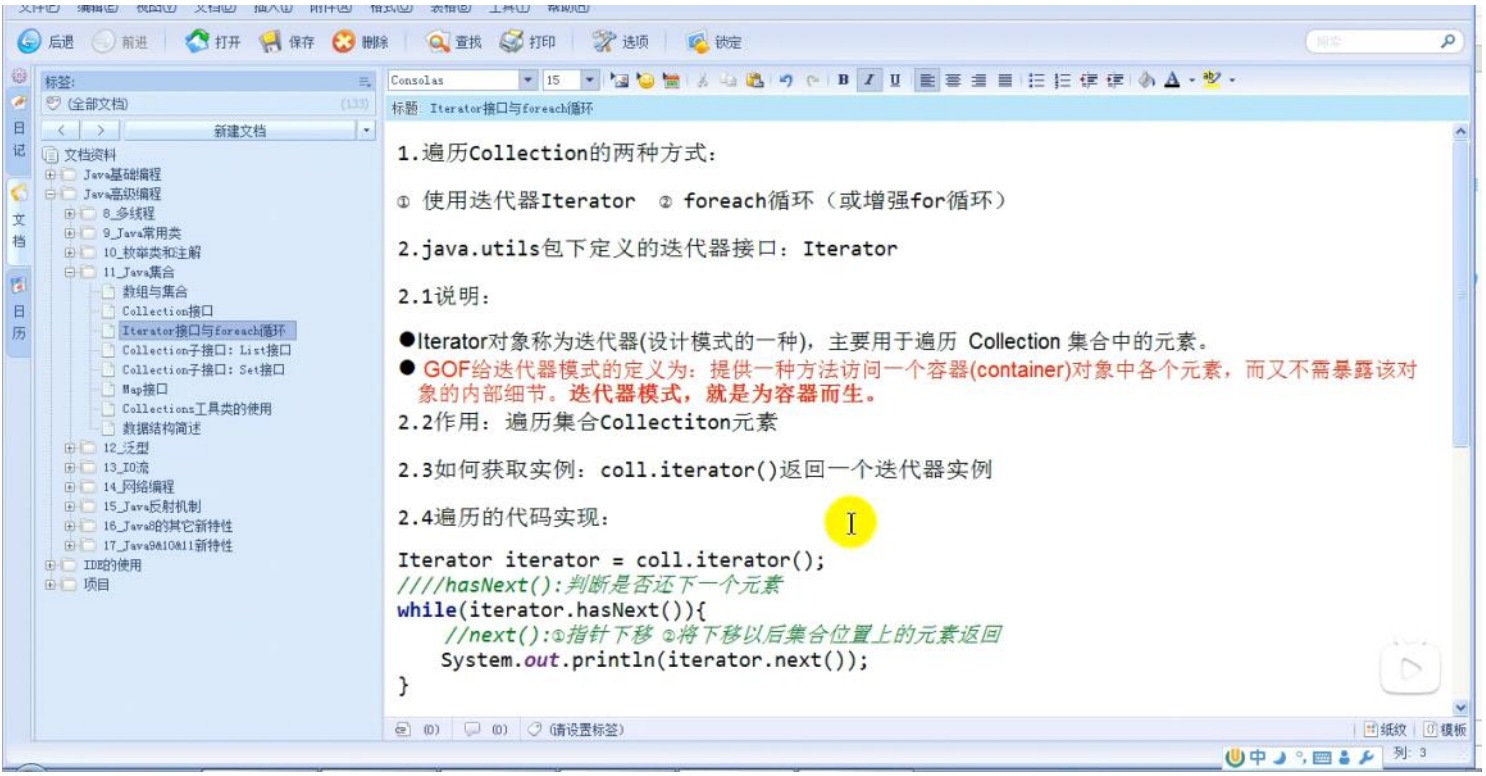
set(int index,Object obj)

get(int index)

add(int index,Object obj)

size()





1. Map存储数据的特点是什么?并指明key, value, entry 存储数据的特点。

双列数据，存储key-value对数据。

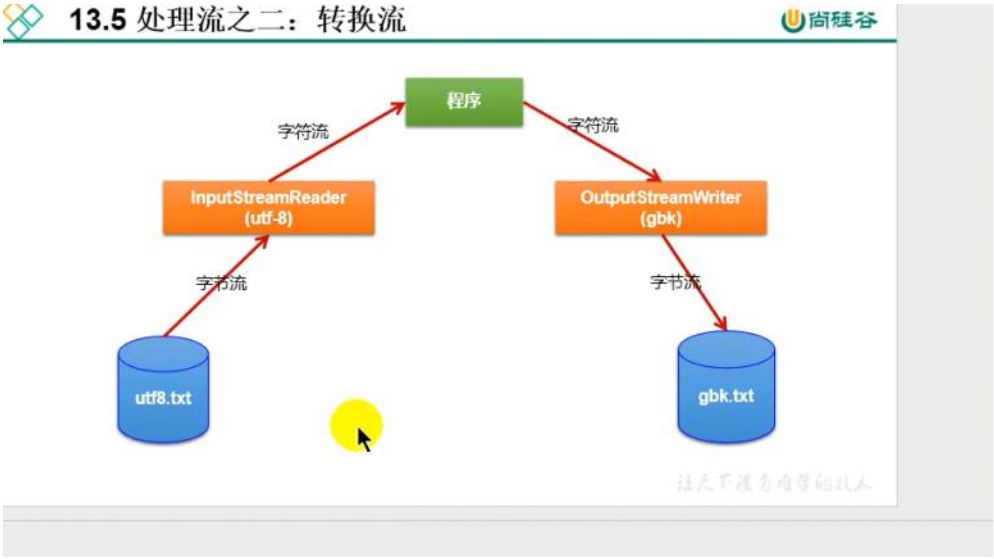
key:无序的、不可重复的-→Set存储

value:无序的、可重复的->Collection存储w

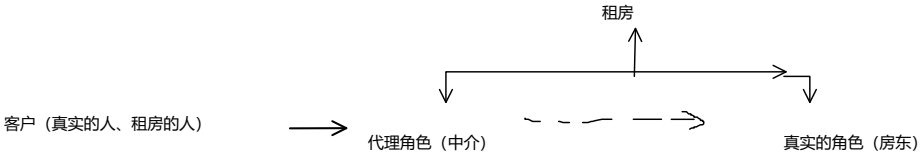
key-value:无序的、不可重复>Set 存储

2.描述HashMap的底层实现原理(jdk8版)

3. Map中常用实现类有哪些?各自有什么特点?



代理模式架构：



10.1 静态代理

角色分析：

抽象角色：一般会使用接口或者抽象类来解决

真实角色：被代理的角色

代理角色：代理真实的角色，代理真实的角色后，我们一般会做一些附属操作

客户：访问代理对象的人！

代码步骤：

1.接口

```java
/**
 * 租房
 */
public interface Rent {
    public void rent();
}
```

2.真实

```java
/**
 *房东
 */
public class Host implements Rent{
    @Override
    public void rent() {
        System.out.println("房东要出租房子");
    }
}
```

角色

```java
/**
 *  客户
 */
public class Client {
    @Test
    public  void test() {
        Host host = new Host();
        //代理
        Proxy proxy = new Proxy(host);
        proxy.rent();
    }
}
```

3.代理角色

```java
/**
 * 代理中介
 */
public class Proxy implements Rent{
    private Host host;

    public Proxy() {
    }

    public Proxy(Host host) {
        this.host = host;
    }

    @Override
    public void rent() {
        host.rent();
        seeHouse();
        Contract();
        fees();
    }

    //看房
    public void seeHouse(){
        System.out.println("中介带你看房");
    }
    //收中介费
    public void fees(){
        System.out.println("收中介费");
    }
    //合同
    public void Contract(){
        System.out.println("签租赁合同");
    }
}
```

4.客户端访问代理角色

```java
/**
 *  客户
 */
public class Client {
    @Test
    public  void test() {
        Host host = new Host();
        //代理
        Proxy proxy = new Proxy(host);
        proxy.rent();
    }
}
```

代理模式的好处：

可以使真实角色更加纯粹！不用去关注一些公共的业务

公共的业务交给代理角色！实现了业务的分工！

公共业务发生扩展的时候，方便集中管理！

缺点：

一个真实的角色就会产生一个代理的角色；代码量就会翻倍！开发效率就会变低~