JDK10新特性

课程介绍

1. 了解Java SE的发展历史
2. 了解Open JDK 和 Oracle JDK
3. JDK10新特性
    - **局部变量类型推断**
    - 将 JDK 的多个代码仓库合并到一个储存库中
    - 垃圾收集器接口
    - G1 引入并行 Full GC
    - 应用程序类数据共享
    - 线程-局部变量管控
    - 删除JNI本地头文件生成工具（javah）
    - 额外的 Unicode 语言标签扩展
    - 在备用存储装置上的堆分配
    - 基于Java的实验性JIT编译器
    - 根证书
    - 基于时间的发行版本控制
    - **新增API**

适合人群

在校大学生，学习过Java，渴望了解Java前沿技术的同学。

正在从事Java编程工作。

JDK10新特性1-局部变量类型推断

目标

了解JDK10之前定义变量存在的问题

掌握JDK10局部变量类型推断的使用

掌握局部变量类型推断使用场景

了解局部变量类型推断不能使用场景

了解局部变量类型推断注意事项

JDK10之前定义变量存在的问题

很多人抱怨Java是一种强类型，需要引入大量的样板代码。很明显类型声明往往被认为是不必要的。JDK10之前的Java代码中，声明一个变量是非常繁琐的：

```
String str = "abc";
long l = 10L;
boolean b = true;
ArrayList<String> list = new ArrayList();
Stream<String> stream = list.stream();
```

许多流行的编程语言都已经支持某种形式的局部变量类型推断，如JS(var)。

JDK10局部变量类型推断的使用

JDK10 可以使用 **var** 进行 **局部变量类型推断**。

```
var str = "abc"; // 推断为 字符串类型
var l = 10L; // 推断为long 类型
var flag = true; // 推断为 boolean 类型
var list = new ArrayList<String>();  // 推断为 ArrayList<String>
var stream = list.stream(); // 推断为 Stream<String>
```

局部变量类型推断使用场景

1. 局部变量
2. 循环内

```
public class demo01var {
    // var x = 10; // 成员变量不能使用var
    public static void main(String[] args) {
        var a = 1;
        var str = "abc";
        var flag = true;
        var list = new ArrayList<String>();
        list.add("aa");
        var stream = list.stream();

        for (var s : list) {
            System.out.println(s);
        }

        for (var i = 0; i < 10; i++) {
            var x = 5;
        }
    }
}
```

局部变量类型推断不能使用场景

1. 成员变量
2. 方法参数
3. 方法返回类型

```
public class demo01var {
    // var x = 10; // 成员变量不能使用var
    public static void main(String[] args) {
    }

    // 参数不能使用var
    // public static void test01(var a) {}

    /* 方法返回类型不能使用var
    public static var test02() {
        return true;
    }*/
}
```

局部变量类型推断注意事项

1. `var` 并不是一个关键字，可以作为标识符，这意味着可以将一个变量、方法、包名写成 `var`。不过一般情况下不会有人这么写的，因为这本身就违反了普遍的命名规范。

```
// var 并不是一个关键字，而是一个保留的类型名称，这意味着可以将一个变量、方法、包名写成 `var`。
public static void test03() {
    var var = 10;
    System.out.println("var = " + var);
}
```

2. `var`声明变量的时候必须赋值、不能用于声明多个变量的情况。

```
// var 不能用来声明没有赋值的变量、不能用于声明多个变量的情况。
public static void test04() {
    // var x = null; // 不行,推断不出到底是什么类型
    int x = 1, y = 2; // 可以
    // var m = 1, n = 2; // 不行
}
```

小结

我们了解了JDK10之前定义变量是比较繁琐的，通过JDK10的局部变量类型推断定义变量更加的简单。

如何进行局部变量类型推断?

```
将变量名左边的类型替换成var即可
var i = 3;
```

什么情况下可以使用局部变量类型推断?

1. 局部变量
2. 循环中的变量

JDK10新特性2-将 JDK 的多个代码仓库合并到一个储存库中

目标

了解这个新特性

它是一项对JDK源码进行管理的方案，开发人员无法直接使用

该新特性的动机

多年以来，JDK的完整代码库已分解为许多的仓库中。在 JDK9 中，JDK的源码被分成有 8 个仓库： root、corba、hotspot、jaxp、jaxws、jdk、langtools 和 nashorn 。JDK的源码分成多个仓库这种情况在支持各种所需的源代码管理操作方面做得很差。特别是，不可能在相互依赖的变更集的存储库之间执行原子提交。例如，如果今天用于单个错误修复或RFE的代码跨越了jdk和hotspot仓库，则无法在托管这两个不同仓库的目录林中原子地对两个仓库进行更改。跨越多个存储库的更改是很常见的。

将 JDK 的多个代码仓库合并到一个储存库中的好处

在 JDK10 中这些将被合并为一个，使得跨相互依赖的变更集的存储库运行 atomic commit （原子提交）成为可能。

JDK10新特性3-垃圾回收器接口

目标

了解这个新特性

这不是让开发者用来控制垃圾回收的接口，而是方便JDK的开发人员在JVM 源代码中快速的集成和移除垃圾回收器。

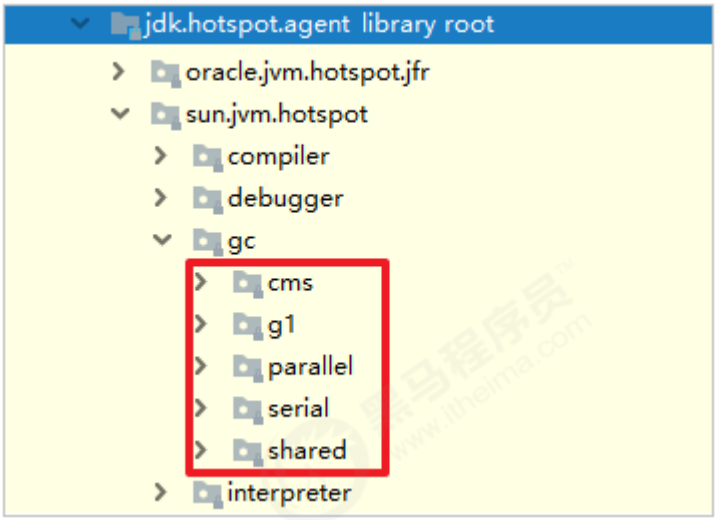
该新特性的动机

当前，垃圾回收器的代码分散，不方便新增新的垃圾回收器也不利于移除现有垃圾回收器。

通过引入一个干净的垃圾回收器（GC）接口来改善不同垃圾回收器的源代码隔离。

该新特性的目标

- JVM内部GC代码的更好的模块化
- 使在不影响当前代码库的情况下向JVM添加新GC变得更加简单
- 使从JVM构建中排除GC更容易



JDK10新特性4-G1 引入并行 Full GC

目标

了解这个新特性

该新特性的动机

G1 是设计来作为一种低延时的垃圾回收器。G1收集器还可以进行非常精确地对停顿进行控制。从JDK7开始启用G1垃圾回收器，在JDK9中G1成为默认垃圾回收策略。截止到ava 9，G1的Full GC采用的是单线程算法。也就是说G1在发生Full GC时会严重影响性能。

该新特性的效果

JDK10又对G1进行了提升，G1 引入并行 Full GC算法，在发生Full GC时可以使用多个线程进行并行回收。能为用户提供更好的体验。

JDK10新特性5-应用程序类数据共享

目标

了解这个新特性

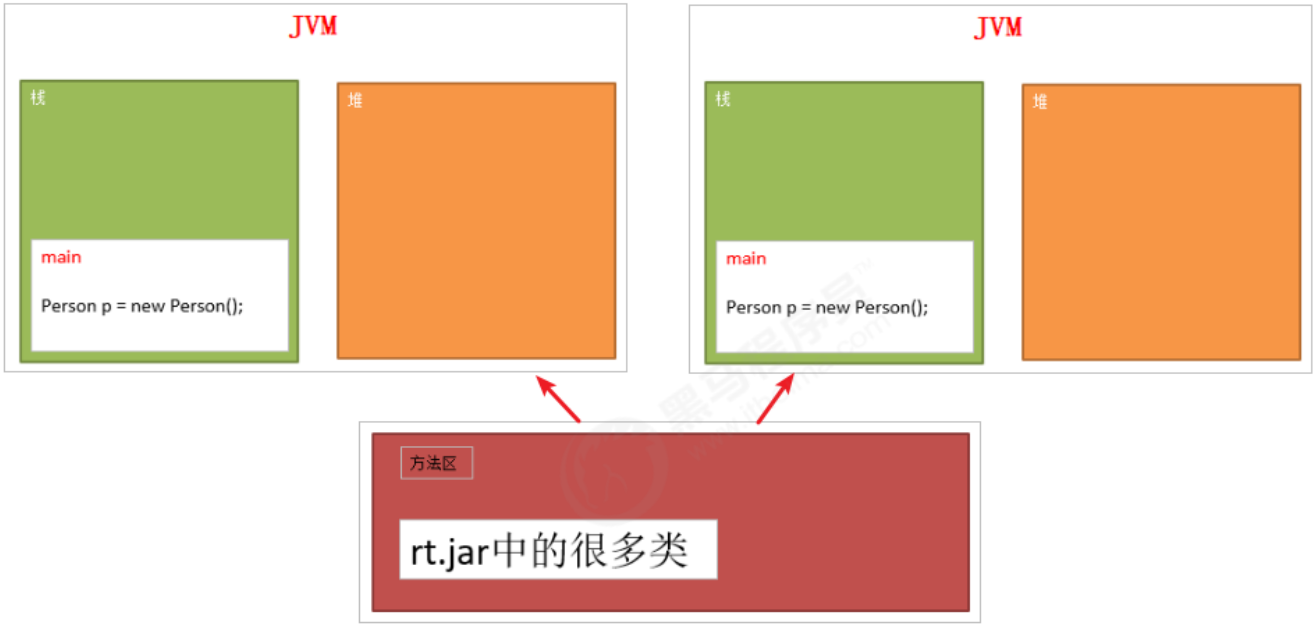
它是一项JVM性能增强功能，开发人员无法直接使用

该新特性的动机


Person.class 字节码文件
public class Person {
private String name;
private int age;
public Person() {
}
public Person(String name, int age) {
this.name = name;
this.age = age;
}
public void getup() {
System.out.println(name + "起床了");
}
}
栈
main
Person p = new Person();
堆
加载到
内存中
Person.class
public class Person {
private String name;
private int age;
public Person() {
}
public Person(String name, int age) {
this.name = name;
this.age = age;
}
public void getup() {
System.out.println(name + "起床了");
}
}


JVM
栈
main
Person p = new Person();
堆
方法区
rt.jar中的很多类


JVM
栈
main
Person p = new Person();
堆
方法区
rt.jar中的很多类



JDK 5中引入的类数据共享，将一组类预处理为共享的存档文件，然后可以在运行时对其进行内存映射以减少启动时间。当多个JVM共享同一个存档文件时，它还可以减少动态内存占用。

JDK 5仅允许引导类加载器加载归档的类。JDK10对应用程序类数据共享进行了扩展，允许“应用程序类加载器”，内置平台类加载器和自定义类加载器加载已归档的类。

该新特性的目标

- 通过在不同的Java进程之间共享通用的类元数据来减少占用空间。
- 缩短程序启动时间。

JDK10新特性6-线程本地握手

目标

了解这个新特性

它是一项JVM性能增强功能，开发人员无法直接使用

该新特性的动机

Safepoint是Hotspot JVM中一种让应用程序所有线程停止的机制。为了要做一些非常之安全的事情，需要让所有线程都停下来它才好做。比如菜市场，人来人往，有人忽然要清点人数，这时候，最好就是大家都原地不动，这样也好统计。Safepoint起到的就是这样一个作用。

JVM会设置一个全局的状态值。应用线程去观察这个值，一旦发现JVM把此值设置为了“大家都停下来”。此时每个应用线程就会得到这个通知，然后各自选择一个point（点）停了下来。这个点就叫Safepoint。待所有的应用线程都停下来了。JVM就开始做一些安全级别非常高的事情了。

比如下面这些事情：

垃圾清理暂停。类的热更新。偏向锁的取消。各种debug操作。

然而，让所有线程都到就近的safepoint停下来本是需要较长的时间。而且让所有线程都停下来显得有些粗暴。

为此Java10就引入了一种可以不用stop all threads的方式，就是Thread Local Handshake（线程本地握手）。

该新特性的效果

线程本地握手是在 JVM 内部相当低级别的更改，修改安全点机制，允许在不运行全局虚拟机安全点的情况下实现线程回调，使得部分回调操作只需要停掉单个线程，而不是停止所有线程。

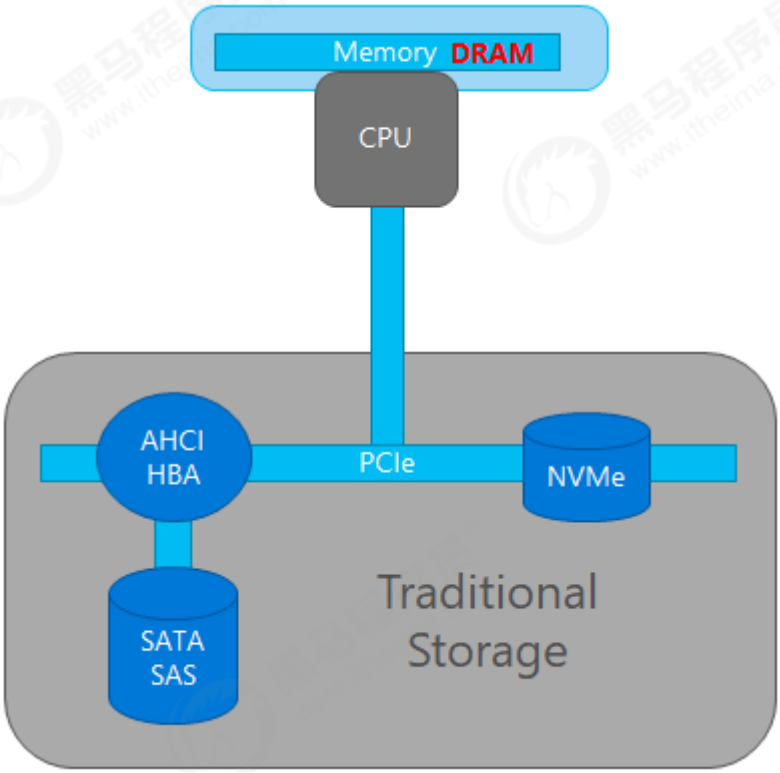
JDK10新特性7-在备用存储装置上进行堆内存分配

目标

了解这个新特性

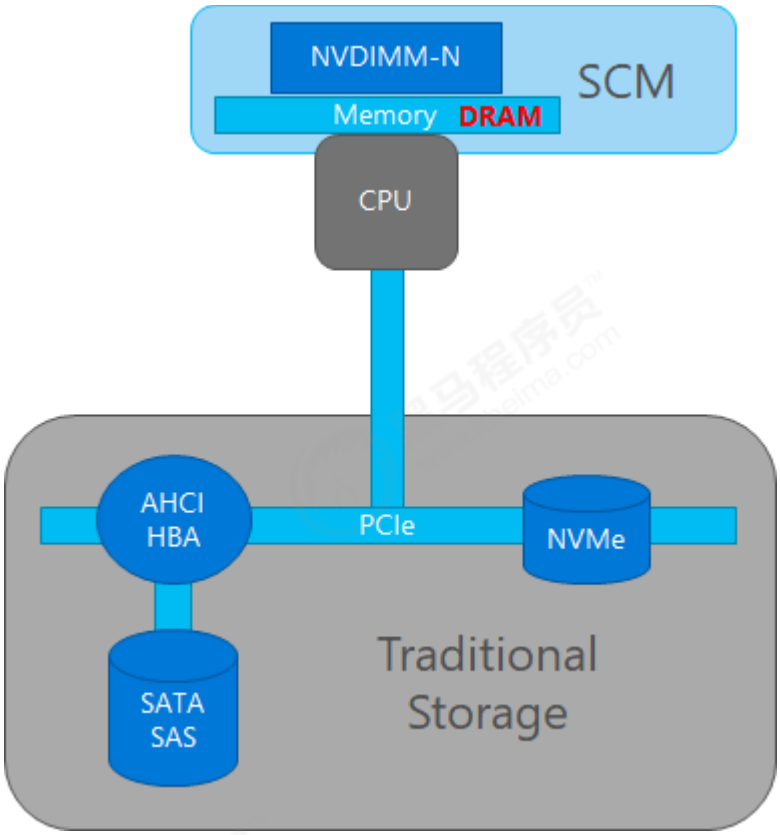
它是一项JVM性能增强功能，开发人员无法直接使用

该新特性的动机



NVDIMM-非易失性双列直插式内存模块（英语：non-volatile dual in-line memory module，缩写**NVDIMM**）特点：价格便宜，速度比DRAM慢，断电能保留数据。

随着廉价的NV-DIMM内存的可用性，未来的系统可能会配备异构内存架构。除了DRAM之外，这种架构还将具有一种或多种具有不同特性的非DRAM存储器。



该JEP的目标是无需更改现有的应用程序代码可以代替DRAM用于对象堆。所有其他内存结构（例如代码堆，元空间，线程堆栈等）将继续驻留在DRAM中。

应用场景

1. 在多JVM部署中，某些JVM（例如守护程序，服务等）的优先级低于其他JVM。与DRAM相比，NV-DIMM具有更高的访问延迟。低优先级进程可以将NV-DIMM内存用于堆，从而允许高优先级进程使用更多的DRAM。
2. 大数据和内存数据库等应用程序对内存的需求不断增长。这样的应用程序可以将NV-DIMM用于堆，因为与DRAM相比，NV-DIMM可能具有更大的容量，且成本更低。

JDK10新特性8-基于Java的实验性JIT编译器

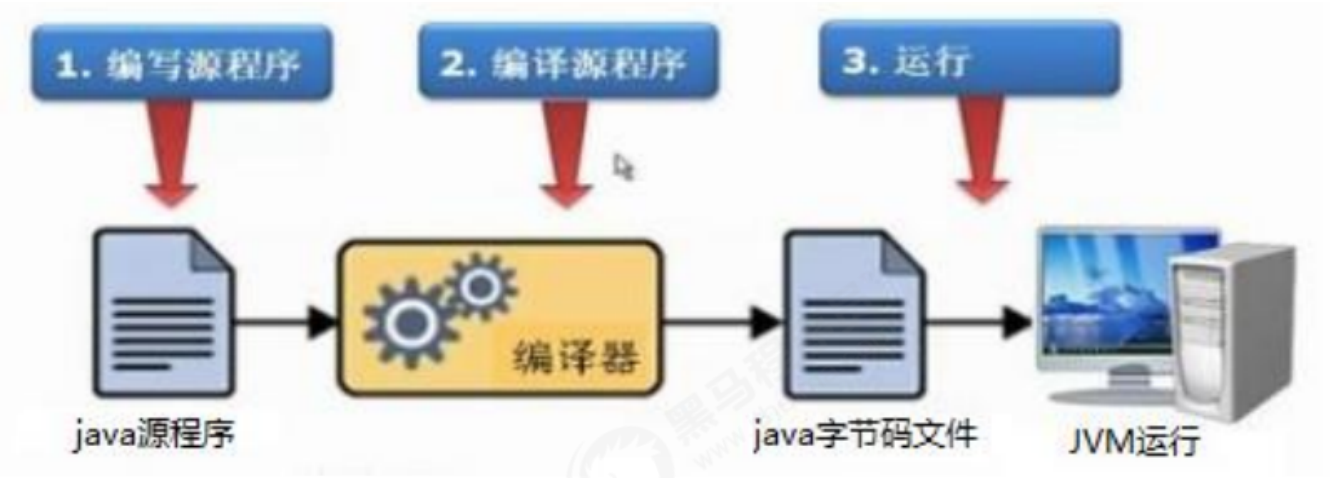
目标

了解这个新特性

它是一项JVM性能增强功能，开发人员无法直接使用

该新特性的动机

Java编译器指的是JDK自带的javac指令。这一指令可将Java源程序编译成.class字节码文件(bytecode)。字节码无法直接运行，但可以被不同平台JVM中的 interpreter(解释器) 解释执行。由于一个Java指令可能被转译成十几或几十个对等的微处理器指令，这种模式执行的速度相当缓慢。



该新特性的目的

由于interpreter效率低下，JVM中又增加JIT compiler（即时编译器，just in time）会在运行时有选择性地将运行次数较多的方法编译成二进制代码，直接运行在底层硬件上。花费少许的编译时间来节省稍后相当长的执行时间，JIT这种设计的确增加不少效率，但是它并未达到最顶尖的效能，因为某些极少执行到的Java指令在JIT编译时所额外花费的时间可能比interpreter解释器执行时的时间还长，针对这些指令而言，整体花费的时间并没有减少。

Graal是基于Java的JIT编译器，这项 JEP 将 Graal 编译器研究项目引入到 JDK 中。为了让 JVM 性能与当前 C++ 所写版本匹敌（或有幸超越）提供基础。

JDK10新特性9-删除javah工具

目标

了解这个新特性

javah 用于生成C语言的头文件。

该新特性的动机

从JDK8开始，javah的功能已经集成到了javac中。去掉javah工具。

```
javac -h . 文件名.java
```

JDK10新特性10-额外的 Unicode 语言标签扩展

目标

了解这个新特性

该新特性的动机

之前对Unicode语言环境扩展仅限于日历和数字。该JEP在相关JDK类中实现最新规范中指定的更多扩展。

对以下附加扩展的支持：

- `cu` （货币类型）
- `fw` （一周的第一天）
- `rg` （区域覆盖）
- `tz` （时区）

为了支持这些附加扩展，将对以下API进行更改：

- `java.text.DateFormat::getInstance` 将根据扩展名返回实例 `ca`， `rg` 和/或 `tz`
- `java.text.DateFormatSymbols::getInstance` 将根据扩展名返回实例 `rg`
- `java.text.DecimalFormatSymbols::getInstance` 将根据扩展名返回实例 `rg`
- `java.text.NumberFormat::get*Instance` 将根据扩展名 `nu` 和/或返回实例 `rg`
- `java.time.format.DateTimeFormatter::localizedBy` 将返回 `DateTimeFormatter` 基于扩展情况下 `ca`，`rg` 和/或 `tz`
- `java.time.format.DateTimeFormatterBuilder::getLocalizedDateTimePattern` 将根据 `rg` 扩展名返回模式字符串。
- `java.time.format.DecimalStyle::of` 将 `DecimalStyle` 根据扩展名返回实例 `nu`，和/或 `rg`
- `java.time.temporal.WeekFields::of` 将 `WeekFields` 根据扩展名 `fw` 和/或返回实例 `rg`
- `java.util.Calendar::{getFirstDayOfWeek,getMinimalDaysInWeek}` 将根据扩展名 `fw` 和/或返回值 `rg`
- `java.util.Currency::getInstance` 将 `Currency` 根据扩展名 `cu` 和/或返回实例 `rg`
- `java.util.Locale::getDisplayName` 将返回一个字符串，其中包括这些U扩展名的显示名称
- `java.util.spi.LocaleNameProvider` 这些U扩展的键和类型将具有新的SPI

JDK10新特性11-根证书

目标

了解这个新特性

该新特性的动机

Open JDK源代码中的密钥库当前为空。因此，默认情况下，TLS等关键安全组件在Open JDK构建中不起作用。

JDK10开源Oracle Java SE Root CA程序中的根证书，减少这些构建与Oracle JDK构建之间的差异。

JDK10新特性12-基于时间的发行版本控制

目标

了解这个新特性

该新特性的动机

从JDK9以后每六个月内严格发布Java SE平台和JDK的新版本 。之前的命名方案不太适合将来。

重铸JEP 223引入的版本号方案，以使其更适合于基于时间的发布模型，这些模型定义功能发布（可以包含新功能）和更新发布（仅修复错误）。

使开发人员或最终用户易于确定版本的发布时间，以便他们可以判断是否将其升级到具有最新安全修复程序以及可能的附加功能的新版本。

JDK10新增API

1. 学习集合新增的copyof方法
2. 学习Reader新增transferTo方法
3. 学习IO流大家族添加Charset参数的方法
4. 学习ByteArrayOutputStream新增toString方法

目标

学习集合新增的copyof方法

学习步骤

1. 学习List、Set、Map新增加了一个静态方法copyOf方法
2. 对copyOf返回的集合进行修改
3. 查看copyOf源码

JDK10 给 java.util 包下的List、Set、Map新增加了一个静态方法 `copyOf` 。copyof方法将元素放到一个不可修改的集合并返回。

代码

```
package com.itheima.demo02copyof;

import java.util.*;

public class Demo02 {
    public static void main(String[] args) {
        var list = new ArrayList<String>();
        list.add("aa");
        list.add("bb");
        list.add("cc");
        list.add("dd");
```

```
        var list2 = List.copyOf(list);
        System.out.println(list2);
        System.out.println(list2.getClass());
        list2.add("ee");
    }
}
```

运行出错

```
[aa, bb, cc, dd]
class java.util.ImmutableCollections$ListN
Exception in thread "main" java.lang.UnsupportedOperationException
    at java.base/java.util.ImmutableCollections.uoe(ImmutableCollections.java:71)
    at java.base/java.util.ImmutableCollections$AbstractImmutableCollection.add(ImmutableCollections.java:75)
    at com.itheima.demo02copyof.Demo02.main(Demo02.java:19)
```

不能往copyof返回的集合中添加元素，因为返回的是不可变的集合：class java.util.ImmutableCollections$ListN

查看java.util.ImmutableCollections$ListN源码：

```
static final class ListN<E> extends AbstractImmutableList<E> implements Serializable {
    ...
}
```

可以看到`ListN`继承了`AbstractImmutableList`。再看`AbstractImmutableList`的源码：

```
static abstract class AbstractImmutableList<E> extends AbstractImmutableCollection<E>
            implements List<E>, RandomAccess {

    // all mutating methods throw UnsupportedOperationException
    @Override public void    add(int index, E element) { throw uoe(); }
    @Override public boolean addAll(int index, Collection<? extends E> c) { throw uoe(); }
    @Override public E       remove(int index) { throw uoe(); }
    @Override public void    replaceAll(UnaryOperator<E> operator) { throw uoe(); }
    @Override public E       set(int index, E element) { throw uoe(); }
    @Override public void    sort(Comparator<? super E> c) { throw uoe(); }
}
```

可以看到在`AbstractImmutableList`中所有对元素增删改的方法都会抛出异常。由此可见copyof返回的是不可变集合。

List，Set，Map的copyof方法代码：

```
package com.itheima.demo02copyof;

import java.util.*;

public class Demo02 {
    public static void main(String[] args) {
        var list = new ArrayList<String>();
        list.add("aa");
        list.add("bb");
```

```
        list.add("cc");
        list.add("dd");

        var list2 = List.copyOf(list);
        System.out.println(list2);
        System.out.println(list2.getClass());
        list2.add("ee");

        System.out.println("---------------");
        var set = new HashSet<String>();
        set.add("b");
        set.add("a");
        set.add("c");
        set.add("d");

        var set2 = Set.copyOf(set);
        System.out.println(set2.getClass());
        for (String string : set2) {
            System.out.println(string);
        }

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

        var map = new HashMap<>();
        map.put("k1", "v1");
        map.put("k2", "v2");

        var map2 = Map.copyOf(map);
        System.out.println(map2.getClass());
        var keys = map2.keySet();
        for (Object object : keys) {
            System.out.println(map2.get(object));
        }
    }
}
```

小结

List,Set,Map新增的copyOf方法，返回的是一个新的集合，这个新的集合是不可变的集合，不能改变集合中的内容

目标

学习Reader新增transferTo方法

学习步骤

1. 回顾以前IO流复制文件
2. 学习JDK10使用transferTo方法复制文件

以前IO流复制文件

```
private static void test01() throws IOException {
    // 字符流复制文本文件
    FileReader fis = new FileReader("JDK10\\files\\a.txt");
    FileWriter fos = new FileWriter("JDK10\\files\\b.txt");
    char[] chs = new char[1024 * 8];
    int len;
    while ((len = fis.read(chs)) != -1) {
        fos.write(chs, 0, len);
    }

    fis.close();
    fos.close();
}
```

以上代码要自己定义数组，编写循环读取和写数据的代码，比较麻烦。使用JDK10的 `transferTo` 方法就很简单了。

JDK10使用transferTo方法复制文件

JDK10 给 InputStream 和 Reader 类中新增了 `transferTo` 方法，`transferTo` 方法的作用是将输入流读取的数据使用字符输出流写出。可用于复制文件等操作。

```
private static void test02() throws IOException {
    FileReader fis = new FileReader("JDK10\\files\\a.txt");
    FileWriter fos = new FileWriter("JDK10\\files\\c.txt");
    fis.transferTo(fos);

    fis.close();
    fos.close();
}
```

我们只需要调用 `transferTo` 方法就可以复制文件了。我们来看看 Reader 类的 transferTo 源码：

```
public long transferTo(Writer out) throws IOException {
    Objects.requireNonNull(out, "out");
    long transferred = 0;
    char[] buffer = new char[TRANSFER_BUFFER_SIZE];
    int nRead;
    while ((nRead = read(buffer, 0, TRANSFER_BUFFER_SIZE)) >= 0) {
        out.write(buffer, 0, nRead);
        transferred += nRead;
    }
    return transferred;
}
```

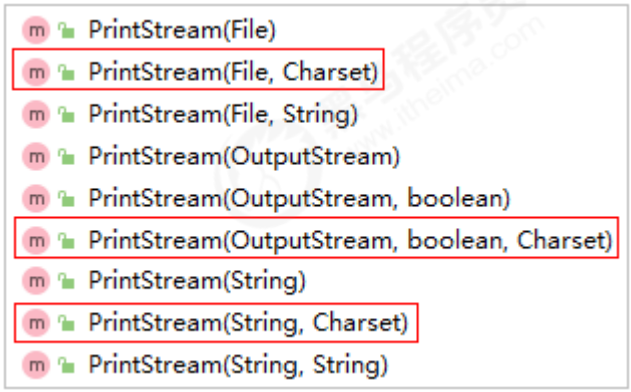
小结

JDK10中给输入流InputStream&Reader新增了transferTo,可以将输入流中的数据转到输出流中.方便文件的复制

目标

学习IO流大家族添加Charset参数的方法

PrintStream，PrintWriter，Scanner添加了带Charset参数的构造方法，通过Charset可以指定IO流操作文本时的编码。



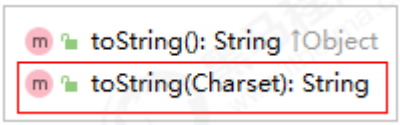

```
package com.itheima.demo04printstream;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.PrintStream;
import java.nio.charset.Charset;

public class Demo04 {
    public static void main(String[] args) throws IOException {
        test01();
        test02();
    }

    // 使用指定的GBK编码打印数据
    private static void test02() throws IOException {
        PrintStream ps = new PrintStream("JDK10\\files\\ps2.txt",Charset.forName("GBK"));
        ps.println("你好");
        ps.close();
    }

    // 使用IDEA默认的UTF-8编码打印数据
    private static void test01() throws FileNotFoundException {
        PrintStream ps = new PrintStream("JDK10\\files\\ps1.txt");
        ps.println("你好");
        ps.close();
    }
}
```

小结

在JDK10中给IO流中的很多类都添加了带Charset参数的方法，比如PrintStream,PrintWriter,Scanner。通过Charset可以指定编码来操作文本。Charset是一个抽象类，我们不能直接创建对象，需要使用Charset的静态方法forName("编码")，返回Charset的子类实例。

目标

学习ByteArrayOutputStream新增toString方法



JDK10给`ByteArrayOutputStream`新增重载`toString(Charset charset)`方法，通过指定的字符集编码字节，将缓冲区的内容转换为字符串。

```
package com.itheima.demo06bytearrayoutputstream;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.UnsupportedEncodingException;

public class Demo06 {
    public static void main(String[] args) throws UnsupportedEncodingException {
        String str = "你好中国";

        ByteArrayInputStream bis = new ByteArrayInputStream(str.getBytes("GBK"));
        ByteArrayOutputStream bos = new ByteArrayOutputStream();
        int b;
        while ((b = bis.read()) != -1) {
            bos.write(b);
        }

        System.out.println(bos.toString());
        System.out.println(bos.toString("GBK"));
    }
}
```

小结

通过ByteArrayOutputStream新增的toString(Charset)， 可以将字节数组输出流中的数据按照指定的编码转成字符串