概述

何为编程

- 就 为 决 个 使 序 写 序代 ，并 到
- 为了使 够 人 图，人 就 将 决 和 够 告 ，使 够 人 令 去工作，完 定 任务 人和 之 交 就

什么是

- Java 向对 ，不仅 了C++ 各 优 ， 了C++ 以 多 ，因 Java 具 功 大和 单 两个 Java 作为 向对 代 ， 好地实 了 向对 ，允 序员以优 复

之后的三大版本

- Java SE（J2SE，Java 2 Platform Standard Edition， 准 ）
  Java SE 以前 为 J2SE 它允 开发和 在 务器 嵌入 境和实 境中使 Java 应 序 Java SE 包含了 Java Web 务开发 ，并为Java EE和Java ME 供基
- Java EE（J2EE，Java 2 Platform Enterprise Edition，企业 ）
  Java EE 以前 为 J2EE 企业 帮助开发和 可 健 可伸 且安全 务器 Java 应 序 Java EE 在 Java SE 基 上 建 ，它 供 Web 务 件 型 和 信 API，

可以　实　企业　　向　务体　　(service-oriented architecture，SOA)和 Web2.0应　序　2018年2　，Eclipse 宣布　　将 JavaEE　名为 JakartaEE

- Java ME（J2ME，Java 2 Platform Micro Edition，　型　）　Java ME 以前　为 J2ME　Java ME 为在　动　备和嵌入　　备（　如　　PDA　　和　印　）上　　应　　序　供　个健　且　　境　Java ME 包　　健　　安　全　型　　多内　　协　以及对可以动　下　　和　　应　　序　丰富　　基于 Java ME　　应　　序只　　写　　，就可以　于　多　备，　且可以利　　个　备　　功

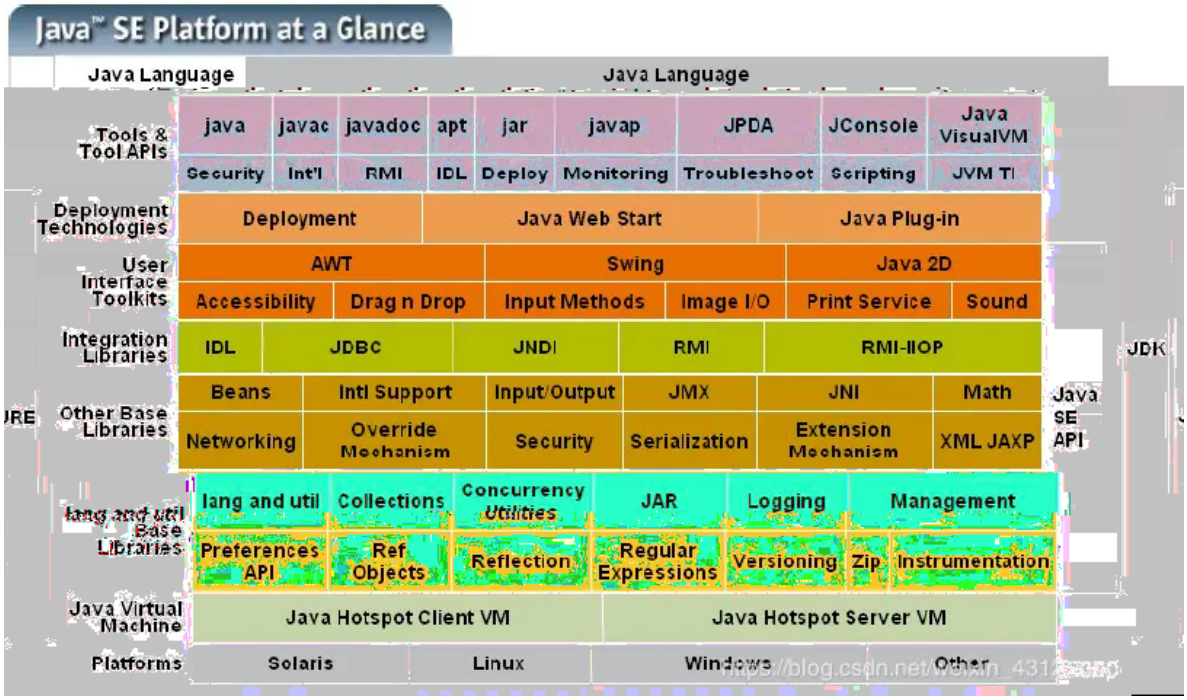
　和　和　　的区别

`看Java官方的图片，Jdk中包括了Jre，Jre中包括了JVM`

- JDK：Jdk　包　了　些Jre之外　东　，就　　些东　帮　们　　Java代　，　　就　　Jvm　　些工具 Java Development Kit　　供　Java开发人员使　　，其中包含了Java　开发工具，也包　了JRE　　以安　了JDK，就　　再单　安　JRE了　其中　开发工具：　　工具(javac.exe)，　包工具(jar.exe)
- JRE：Jre大　分　　C 和 C++　　写　，他　　们在　java　　基　　库 Java Runtime Environment包　Java　　和Java　序　　库　　库主　java.lang　包：包含了　Java　序　不可少　　，如基　　型　基　　学函　　字　串处　　常处　　，　　加　　个包

如　　个开发好　Java　序，　　中只　　安　JRE即可

- Jvm：在倒　　二层　他可以在（　后　层　）各　平台上　　Java Virtual Machine　Java　，Java　序　　在　　上，不同　平台　　己　　，因　Java　　可以实　　平台



什么是跨平台性？原理是什么

- 　　平台　，　　java　　写　　序，　　后，可以在多个　　平台上
- 实　原　：Java　序　　java　　在　　平台上　　，只　　可以安　　应　java　，　　就可以　　java　序

语言有哪些特点

- 单 学 (Java 与C 和C++)
- 向对 (封 , , 多)
- 平台 关 (Java 实 平台 关)
- 并且 便 (Java 就 为 化)
- 多 (多 制使应 序在同 并 多 任)
- 健 (Java 型 制 常处 垃圾 动)
- 安全 好

什么是字节码？采用字节码的最大好处是什么

- **字节码**：Java 代 器 后产 件（即 展为.class 件），它不 向任何 定 处 器，只 向
- **采用字节码的好处**：

  Java 字 ，在 定 度上 决了传 型 低 ，同 又保了 型 可 以Java 序 ， 且， 于字 并不专对 定 器，因 ，Java 序 便可在多 不同 上

- **先看下 中的编译器和解释器**：

  Java中 入了 ，即在 器和 序之 加入了 层 器 台 器在任何平台上 供 序 个 共同 口 序只 向 ，够 代 ， 后 器 将 代 为 定 器 在Java中， 供 代 叫做字 （即 展为.class 件），它不 向任何 定 处 器，只 向 平台 器 不同 ，但 实 同 Java 序 器 后变字 ，字 ， 将 字 器， 器将其 定 器上 器 ， 后在 定 器上 ， 就 上 到 Java 与并存

  Java 代 ----> 器---->jvm可 Java字 (即 令)---->jvm---->jvm中 器-----> 器可 二 制 器 ----> 序

什么是 程序的主类？应用程序和小程序的主类有何不同？

- 个 序中可以 多个 ，但只 个 主 在Java应 序中， 个主 包含main() 在Java小 序中， 个主 个 JApplet Applet 子 应 序 主 不 定 public ，但小 序 主 public 主 Java 序 入口

应用程序与小程序之间有那些差别？

- 单 应 序 从主 启动(也就 main()) applet小 序 main ，主 嵌在 器 上 (init() run() 启动)，嵌入 器 flash 小 似

和 的区别

我知道很多人没学过C++，但是面试官就是没事喜欢拿咱们Java和C++比呀！没办法！！！就算没学过C++，也要记下来！

- 向对 ， 封 和多
- Java不 供 内存， 序内存 加安全
- Java 单 ，C++ 多 ； Java 不可以多 ，但 口可以多
- Java 动内存 制，不 序员 动 内存

和 的对比

1. Oracle JDK 将 三年发布 ， OpenJDK 三个 发布 ；
2. OpenJDK 个参 型并且 完全开 ， Oracle JDK OpenJDK 个实 ，并不 完全开 ；
3. Oracle JDK OpenJDK 定 OpenJDK和Oracle JDK 代 几乎 同，但Oracle JDK 多 和 些 修复 因 ，如 开发企业/商业 件， 建 Oracle JDK，因为它了 底 和 定 些 况下， 些人 到在使 OpenJDK 可 会 到了 多应 序崩 ，但 ，只 切 到Oracle JDK就可以 决 ；
4. 在响应 和JVM ，Oracle JDK与OpenJDK 供了 好 ；
5. Oracle JDK不会为即将发布 供 ， 到 取 ；
6. Oracle JDK 二 制代 可协 可， OpenJDK GPL v2 可 可

基础语法

数据类型

有哪些数据类型

定义：Java 型 ，对于 定义了 具体 型，在内存中分 了不同大小 内存

分类

- 基 型
  - 值型
  - 型(byte,short,int,long)
    - 型(float,double)
    - 字 型(char)
    - 布尔型(boolean)
- 型
  - (class)
    - 口(interface)
    - ([])

基本数据类型图

类型	类型名称	关键字	占用内存	取值范围	作为成员变量的默认值
整形	字节型	byte	1 字节	-128(-2^7) ~ 127(2^7-1)	0
	短整型	short	2 字节	-32,768(-2^15) ~ 32,767(2^15-1)	0
	整型	int	4 字节	-2,147,483,648(-2^31) ~ 2,147,483,647(2^31-1)	0
	长整型	long	8 字节	-9,223,372,036,854,775,808(-2^63) ~ 9,223,372,036,854,775,807(2^63-1)	0L
浮点型	单精度浮点型	float	4 字节	-3.403E38 ~ 3.403E38	0.0F
	双精度浮点型	double	8 字节	-1.798E308 ~ 1.798E308	0.0D
字符型	字符型	char	2 字节	表示一个字符，如('a','A','家')	'\u0000'
1 字节	只有两个值，true 或 false			false	布尔型	布尔型	boolean

是否能作用在 上，是否能作用在 上，是否能作用在 上

- 在 Java 5 以前，switch(expr)中，expr 只 byte short char int 从 Java5 开始，Java 中 入了 举 型，expr 也可以 enum 型，从 Java 7 开始，expr 可以 字 串（String），但 型（long）在 前 中 不可以

用最有效率的方法计算 乘以

- 2 << 3（左 3 位 于乘以 2 3 ，右 3 位 于 以 2 3 ）

等于多少? 等于多少

- Math.round(11.5) 回值 12，Math.round(-11.5) 回值 -11 四 五入 原 在参上加 0.5 后 下取

是否正确

- 不 3.4 双 度 ，将双 度型（double） 值 型（float）属于下 型（down-casting，也 为 化）会 度 失，因 制 型 float f =(float)3.4; 写 float f =3.4F;

有错吗 有错吗

- 对于 short s1 = 1; s1 = s1 + 1; 于 1 int 型，因 s1+1 也 int型， 制 型 值 short 型
- short s1 = 1; s1 += 1;可以 ，因为 s1+= 1; 于 s1 = (short(s1 + 1);其中 含 制 型

编码

语言采用何种编码方案? 有何特点?

- Java Unicode 准，Unicode（ 准 ），它为 个字 制 了 个唯 值，因 在任何 ，平台， 序 可以 使

注释

什么　　注释

定义：　于　　序　　字

分类

- 单
　　：//　　字
- 多
　　：/*　　字 */
-
　　：/**　　字 */

作用

- 在　序中，尤其　复　　序中，　地加入　可以增加　序　可　，　利于　序　修　和交　　内容在　序　　候会　，不会产　　代　，　　分不会对　序　产　任何　响

注意事项：多行和文档注释都不能嵌套使用。

访问修饰符

访问修饰符　　以及不写（默认）时的区别

- **定义**：Java中，可以使　　修　　保　对　变　　和　　Java　4　不同

关键字

有没有

- goto Java中 保 字，在 前 Java中 使

有什么用?

用于修饰类、属性和方法；

- final修 不可以
- final修 不可以 写
- final修 变 不可以 变， final修 不可变 变 ， 不 向 内容，
  向 内容 可以 变

区别

- final可以修 变 ，修 不 修 不 写
  修 变 变 个常 不 值
- finally 作 在try-catch代 块中，在处 常 候， 常 们将 定 代
  finally代 块中， 不 否出 常， 代 块 会 ， 存 些关 代

- finalize 个 ，属于Object 个 ， Object ， 垃圾
  回 器 ， 们 System.gc() 候， 垃圾回 器 finalize()，回 垃圾，
  个对 否可回 后判

关键字的用法

- this 个对 ，代 对 ，可以 为： 向对 个
- this 在java中大体可以分为3 ：
  - 1. ，this 于 向 前对
  - 2. 参与 员名字 名， this 区分：

```
public Person(String name, int age) {
    this.name = name;
    this.age = age;
}
```

  - 3. 函

```
class Person{
    private String name;
    private int age;

    public Person() {
    }

    public Person(String name) {
        this.name = name;
```

- super（参 ）： 中 个 函 （应 为 函 中 句）
- this（参 ）： 中另 函 （应 为 函 中 句）

与 的区别

- super: 它 前对 中 员（ 中 中 员 函 ，基 与 中 同 员定义 如：super.变 名 super. 员函 名（实参）
- this：它代 前对 名（在 序中 产 二义 之处，应使 this 前对 ；如 函 参与 中 员 同名， this 员变 名）
- super()和this() 似,区别 ，super()在子 中 ，this()在 内 其它
- super()和this()均 在 内
- 尽 可以 this 个 器，但却不 两个
- this和super不 同 出 在 个 函 ，因为this 会 其它 函 ，其它 函 也会 super 句 存在， 以在同 个 函 同 句，就失去了 句义， 器也不会
- this()和super() 对 ， 以，均不可以在static 境中使 包 ：static变 ,static ，static 句块
- 从 上 ，this 个 向 对 ， super 个Java关 字

存在的主要意义

- static 主 义 在于创建 于具体对 域变 **以致于即使没有创建对象，也能使用属性和调用方法**！
- static关 字 个 关 作 就 **用来形成静态代码块以优化程序性能** static块可以于 中 任何地 ， 中可以 多个static块 在 初 加 候，会 static块 序 个static块，并且只会
- 为什么 static块可以 优化 序 ， 因为它 :只会在 加 候 因 ，多 候会将 些只 初始化 作 在static代 块中

的独特之处

- 1 static修 变 于 任何对 ，也就 ， 些变 和 **不属于任何一个实例对象，而是被类的实例对象所共享**

> 么 “ 实例对 共享” 句 ？就 ， 个 员，它 属于大伙 大伙 个 多个对 实例， 们 个 可以创建多个实例！ ， 对 共享 ，不像 员变 个 个 个 单个实例对 … 已了，你 了 ？

- 2 在 加 候，就会去加 static修 分， 且只在 使 加 并 初始化， 就 初始化，后 可以再 值
- 3 static变 值在 加 候分 ，以后创建 对 候不会 分 值 ，可以任 值 ！
- 4 static修 变 优先于对 存在 ，也就 个 加 完 之后，即便 创建对 ，也可以去

应用场景

- 因为static 实例对 共享，因 如 **某个成员变量是被所有对象所共享的，那么这个成员变量就应该定义为静态变量**
- 因 常 static应 场 ：

> 1 修 员变 2 修 员 3 代 块 4 修 只 修 内 也就 内 5 导包

注意事项

- 1 只
- 2 可以 ，也可以

流程控制语句

的区别及作用

- break 出 上 层 ，不再 (前 体)
- continue 出 ， 下 (在 入下 个 件)
- return 序 回，不再 下 代 (前 回)

在 中，如何跳出当前的多重嵌套循环

- 在Java中， 出多 ，可以在外 句前定义 个 号， 后在 层 体 代中使 带 号 break 句，即可 出外层 例如：

```
public static void main(String[] args) {
    ok:
    for (int i = 0; i < 10; i++) {
        for (int j = 0; j < 10; j++) {
            System.out.println("i=" + i + ",j=" + j);
            if (j == 5) {
                break ok;
            }
        }
    }
}
```

面向对象

面向对象概述

面向对象和面向过程的区别

- **面向过程**:
- 优 : 向对 , 因为 实例化，开 大， ; 如单 嵌入 开发 Linux/Unix 向 开发， 因
- : 向对 复 展
- **面向对象**:
  - 优 : 复 展， 于 向对 封 多 ，可以 出 低 合 ，使 加 加 于
  - : 向 低

面向过程是具体化的，流程化的，解决一个问题，你需要一步一步的分析，一步一步的实现。

面向对象是模型化的，你只需抽象出一个类，这是一个封闭的盒子，在这里你拥有数据也拥有解决问题的方法。需要什么功能直接使用就可以了，不必去一步一步的实现，至于这个功能是如何实现的，管我们什么事？我们会用就可以了。

面向对象的底层其实还是面向过程，把面向过程抽象成类，然后封装，方便我们使用的就是面向对象了。

面向对象三大特性

面向对象的特征有哪些方面

向对 主 以下几个 **:

- **抽象**: 将 对 共同 出 ，包 和 为 两 只关 对 哪些属 和 为，并不关 些 为 什么
- **封装** 个对 属 化，同 供 些可以 外 属 ，如 属 不 外 ， 们大可不 供 外 但 如 个 供 外 ， 么 个 也 什么 义了
- **继承** 使 已存在 定义作为基 建 ， 定义可以增加 功 ，也可以 功 ，但不 地 使 们 够 常 便地复 以前 代

- 关于 如下3 住:
- 子 private 属 和
- 子 可以 己属 和 ，即子 可以对 展
- 子 可以 己 实 (以后介)
- **多态**: 口定义 变 可以 向子 具体实 实例对 了 序 展 在Java中 两 可以实 多 : (多个子 对同 写)和 口(实 口并 口中同)

什么是多态机制? 语言是如何实现多态的?

- 多 就 序中定义 变 向 具体 型和 变 发出 在 并不 定, 在 序 定,即 个 变 倒底会 向哪个 实例对 , 变 发出 到底 哪个 中实 , 在 序 决定 因为在 序 定具体 , ,不 修 序代 ,就可以 变 定到各 不同 实 上,从 导 具体 之 变,即不修 序代 就可以 变 序 定 具体代 , 序可以 多个 , 就 多
- 多 分为 多 和 多 其中 多 ,主 ,它 参 列 不同 区分不同 函 , 之后会变 两个不同 函 ,在 不上多 多 动 ,它 动 定 实 ,也就 们 多

多态的实现

- Java实 多 三个 件: 写 向上 型
  - : 在多 中 存在 关 子 和
  - 写: 子 对 中 些 定义,在 些 就会 子
  - 向上 型: 在多 中 将子 对 ,只 够具备 和子

只有满足了上述三个条件，我们才能够在同一个继承结构中使用统一的逻辑实现代码处理不同的对象，从而达到执行不同的行为。

对于Java而言，它多态的实现机制遵循一个原则：当超类对象引用变量引用子类对象时，被引用对象的类型而不是引用变量的类型决定了调用谁的成员方法，但是这个被调用的方法必须是在超类中定义过的，也就是说被子类覆盖的方法。

面向对象五大基本原则是什么（可选）

- 单 原则SRP(Single Responsibility Principle)
  功 单 ,不 包 万 , 似
- 开 封 原则OCP(Open - Close Principle)
  个 块对于 展 开 ,对于修 封 , 增加功 , 修 , , 万个不乐
- 原则LSP(the Liskov Substitution Principle LSP)
  子 可以 出 在 够出 任何地 如你 代 你 去你 家干 哈哈~~
- 依 倒 原则DIP(the Dependency Inversion Principle DIP)
  层 块不应 依 于低层 块,他们 应 依 于 不应 依 于具体实 ,具体实 应 依 于 就 你出国 你 中国人, 不 你 哪个 子 如 中国人 ,下 具体 xx ,xx市,xx县 你 依 中国人, 不 你 xx

- 口分 原则ISP(the Interface Segregation Principle ISP)
  多个与 定客 关 口 个 口 好 就 如 个

，　　，　　功，　几个功　分　不同　口，　在　个　口　好　多

类与接口

抽象类和接口的对比

- 子　口　合
- 从　层　，　对　，　，　口　为　，　为

相同点

- 口和　不　实例化
- 位于　，　于　其他实
- 包含　，其子　写　些

不同点

参数	抽象类	接口
声	使　abstract关　字声	口使　interface关　字声
实	子　使　extends关　字　如　子　不　，它　供　中　声　实	子　使　implements关　字　实　口　它　供　口中　声　实
器	可以　器	口不　器
修	中　可以　任　修	口　修　public　并且　不允　定义为 private　protected
多	个　多只　个	个　可以实　多个　口
字　声	字　声　可以　任	口　字　static 和 final

备注：Java8中　口中　入　和　，以　减少　和　口之　差

现在，我们可以为接口提供默认实现的方法了，并且不用强制子类来实现它。

- 口和　各　优　，在　口和　上，　守　个原则：
  - 为　型应　口　不　定义，　以　常　优先　口，尽　少
  - 候　常　如下　况：　定义子　为，又　为子　供　功

普通类和抽象类有哪些区别?

- 不 包含 , 可以包含
- 不 实例化, 可以 实例化

抽象类能使用 修饰吗?

- 不 , 定义 就 其他 , 如 定义为 final 就不 , 就会产 , 以 final 不 修

创建一个对象用什么关键字? 对象实例与对象引用有何不同?

- new关 字, new创建对 实例(对 实例在堆内存中), 对 向对 实例(对 存在 内存中) 个对 可以 向0个 1个对 (子可以不 , 也可以 个); 个对 可以 n个 向它(可以 n 子 住 个)

变量与方法

成员变量与局部变量的区别有哪些

- 变 : 在 序 中, 在 个 围内其值可以发 变 从 上 , 变 其实 内存中 小块区域
- 员变 : 外 , 内 定义 变
- 局 变 : 中 变
- 员变 和局 变 区别

作用域

- 员变 : 对 个
- 局 变 : 只在 个 围内 (就 , 句体内)

存储位置

- 员变 : 对 创建 存在, 对 失 失, 存储在堆内存中
- 局 变 : 在 , 句 候存在, 存储在 内存中 完, 句 后, 就 动

生命周期

- 员变 : 对 创建 存在, 对 失 失
- 局 变 : 完, 句 后, 就 动

初始值

- 员变 : 初始值
- 局 变 : 初始值, 使 前 值

在 中定义一个不做事且没有参数的构造方法的作用

- Java 序在 子 之前, 如 super() 定 , 则会 中“ 参 ” 因 , 如 中只定义了 参 , 在子 中又 super() 中 定 , 则 将发 , 因为Java 序在中 不到 参 可供 决办 在 加上 个不做事且 参

在调用子类构造方法之前会先调用父类没有参数的构造方法，其目的是?

- 帮助子　做初始化工作

一个类的构造方法的作用是什么？若一个类没有声明构造方法，改程序能正确执行吗？为什么?

- 主　作　　完　对　对　　初始化工作　可以　　因为　个　即使　　声　　也会　不带参

构造方法有哪些特性?

- 名字与　名　同;
- 　　回值，但不　　void声　　函　;
- 　　对　　动　　,

静态变量和实例变量区别

- 　　变　:　　变　　于不属于任何实例对　，属于　　，　以在内存中只会　　份，在　　加　　中，JVM只为　　变　分　　内存
- 实例变　:　　创建对　，　会为　个对　分　　员变　内存　　，实例变　　属于实例对　　，在内存中，创建几　对　，就　几份　员变

静态变量与普通变量区别

- static变　也　作　　变　，　　变　和　　变　　区别　:　　变　　对　　共享，在内存中只　　个副　，它　且仅　在　初　加　　会　初始化　　变　　对　　，在创建对　　候　初始化，存在多个副　，各个对　　副　互不　响
- 　　就　static　员变　　初始化　序　　定义　　序　　初始化

静态方法和实例方法有何不同?

静态方法和实例方法的区别主要体现在两个方面：

- 在外　　，可以使　"　名.　　名"　　，也可以使　"对　名.　　名"　实例　　只　后　　也就　　，　　可以　　创建对
- 　　在　　员　，只允　　员（即　　员变　和　　），　不允　实例　员变　和实例　　；实例　　则　　制

在一个静态方法内调用一个非静态成员为什么是非法的?

- 　于　　可以不　　对　　，因　在　　，不　　其他　　变　，也不可以　　变　　员

什么是方法的返回值？返回值的作用是什么？

- 回值 们 取到 个 体中 代 后产 ！（前 可 产 ） 回值 作 ： 出 ，使 它可以 于其他 作！

内部类

什么是内部类？

- 在Java中，可以将 个 定义 在另外 个 定义内 ， 就 **内部类** 内 就 个属 ，与其他属 定义

内部类的分类有哪些

内部类可以分为四种：**成员内部类、局部内部类、匿名内部类和静态内部类**。

静态内部类

- 定义在 内 ，就 内

```
public class Outer {

    private static int radius = 1;

    static class StaticInner {
        public void visit() {
            System.out.println("visit outer static  variable:" + radius);
        }
    }
}
```

- 内 可以 外 变 ， 不可 外 变 ； 内 创建 ，`new 外部类.静态内部类()`，如下：

```
Outer.StaticInner inner = new Outer.StaticInner();
inner.visit();
```

成员内部类

- 定义在 内 ， 员位 上 ，就 员内

```
public class Outer {

    private static  int radius = 1;
    private int count =2;

     class Inner {
        public void visit() {
            System.out.println("visit outer static  variable:" + radius);
            System.out.println("visit outer   variable:" + count);
        }
    }
}
```

- 员内 可以 外 变 和 ，包 和 ， 和公 员内 依于外 实例，它 创建 `外部类实例.new 内部类()`，如下：

```
Outer outer = new Outer();
Outer.Inner inner = outer.new Inner();
inner.visit();
```

局部内部类

- 定义在 中 内 ，就 局 内

```
public class Outer {

    private  int out_a = 1;
    private static int STATIC_b = 2;

    public void testFunctionClass(){
        int inner_c =3;
        class Inner {
            private void fun(){
                System.out.println(out_a);
                System.out.println(STATIC_b);
                System.out.println(inner_c);
            }
        }
        Inner  inner = new Inner();
        inner.fun();
    }
    public static void testStaticFunctionClass(){
        int d =3;
        class Inner {
            private void fun(){
                // System.out.println(out_a); 编译错误，定义在静态方法中的局部类不可以访问外部类的实例变量
System.out.println(STATIC_b);
                System.out.println(d);
            }
        }
        Inner  inner = new Inner();
        inner.fun();
    }
```

```
}
```

- 定义在实例 中 局 可以 外 变 和 ，定义在 中 局 只 外 变 和 局 内 创建 ，在对应 内，`new 内部类()`，如下：

```
public static void testStaticFunctionClass(){
    class Inner {
    }
    Inner  inner = new Inner();
 }
```

匿名内部类

- 匿名内 就 名字 内 ， 常开发中使 多

```
public class Outer {

    private void test(final int i) {
        new Service() {
            public void method() {
                for (int j = 0; j < i; j++) {
                    System.out.println("匿名内部类" );
                }
            }
        }.method();
    }
 }
  //匿名内部类必须继承或实现一个已有的接口
interface Service{
    void method();
}
```

- 了 名字，匿名内 以下 ：
  - 匿名内 个 实 个 口
  - 匿名内 不 定义任何 员和
  - 在 参 匿名内 使 ， 声 为 final
  - 匿名内 不 ，它 实 实 口
- 匿名内 创建 ：

```
new 类/接口{
  //匿名内部类实现部分
}
```

内部类的优点

`我们为什么要使用内部类呢？因为它有以下优点：`

- 个内 对 可以 创建它 外 对 内容，包 ！
- 内 不为同 包 其他 ，具 好 封 ；

- 内　　实　了“多　　”，优化 java 单
- 匿名内　　可以　　便　定义回

内部类有哪些应用场景

1. 些多　　场合
2. 决　些　　向对　　句块
3. 　使　内　　，使　代　　加　　和富　　展
4. 　　个　　了它　外　　，不再　其他　　使

局部内部类和匿名内部类访问局部变量的时候，为什么变量必须要加上　　？

- 局　内　　和匿名内　　局　变　　候，为什么变　　加上final　？它内　原　　什么　？先　　代　：

```
public class Outer {

    void outMethod(){
        final int a =10;
        class Inner {
            void innerMethod(){
                System.out.println(a);
            }
        }
    }
}
```

- 以上例子，为什么　加final　？　因为**生命周期不一致**，局　变　　存储在　中，　　后，　final　局　变　就　　局　内　　对局　变　　依　存在，如　局　内　　局　变　　，就会出　　加了final，可以　保局　内　　使　　变　与外层　局　变　区分开，　决了　个

内部类相关，看程序说出运行结果

```
public class Outer {
    private int age = 12;

    class Inner {
        private int age = 13;
        public void print() {
            int age = 14;
            System.out.println("局部变量：" + age);
            System.out.println("内部类变量：" + this.age);
            System.out.println("外部类变量：" + Outer.this.age);
        }
    }

    public static void main(String[] args) {
        Outer.Inner in = new Outer().new Inner();
        in.print();
```

```
    }

}
```

:

```
局部变量：14
内部类变量：13
外部类变量：12
```

重写与重载

构造器（ ）是否可被重写（ ）

- 器不 ，因 不 写，但可以

重载（ ）和重写（ ）的区别。重载的方法能否根据返回类型进行区分?

- 和 写 实 多 ，区别在于前 实 多 ， 后 实 多
- : 发 在同 个 中， 名 同参 列 不同 (参 型不同 个 不同 序不同) ，与 回值和 修 关，即 不 回 型 区分
- 写: 发 在 子 中， 名 参 列 同， 回值小于 于 ， 出 常小于 于 ， 修 大于 于 (代 原则) ; 如 修 为private则子 中就不 写

对象相等判断

和 的区别是什么

- : 它 作 判 两个对 地址 不 即，判 两个对 不 同 个对 (基型 == 值， 型 == 内存地址)
- : 它 作 也 判 两个对 否 但它 两 使 况:
  - 况1: equals() 则 equals() 两个对 ， 价于 “==” 两个对
  - 况2: 了 equals() ， 们 equals() 两个对 内容 ; 它们 内容 ，则 回 true (即， 为 两个对)
  - **举个例子:**

```
  public class test1 {
      public static void main(String[] args) {
          String a = new String("ab"); // a 为一个引用
String b = new String("ab"); // b为另一个引用,对象的内容一样
String aa = "ab"; // 放在常量池中
String bb = "ab"; // 从常量池中查找
if (aa == bb) // true
              System.out.println("aa==bb");
          if (a == b) // false，非同一对象
System.out.println("a==b");
```

```
            if (a.equals(b)) // true
                System.out.println("aEQb");
            if (42 == 42.0) { // true
                System.out.println("true");
            }
        }
    }
```

- **说明：**
  - String中 equals 写 ，因为object equals 对 内存地址，String equals 对 值
  - 创建String 型 对 ， 会在常 中 已 存在 值和 创建 值同 对 ，如 就 它 前 如 就在常 中 创建 个String对

与 重要

- HashSet如何 复
- 两个对 hashCode() 同，则 equals() 也 定为 true，对吗?
- hashCode和equals 关
- 官可 会 你：“你 写 hashcode 和 equals 么，为什么 写equals 写hashCode ？”

介绍

- hashCode() 作 取哈希 ，也 为 列 ；它实 上 回 个int 个哈希 作 定 对 在哈希 中 位 hashCode() 定义在JDK Object.java中， 就 味 Java中 任何 包含 hashCode()函
- 列 存储 值对(key-value)，它 ： “ ” 出对应 “值” 其中就 利 到了 列 ！（可以 到 对 ）

为什么要有

我们以“HashSet 如何检查重复”为例子来说明为什么要有 hashCode：

- 你 对 加入 HashSet ，HashSet 会先 对 hashcode 值 判 对 加入 位 ，同 也会与其他已 加入 对 hashcode 值作 ，如 hashcode，HashSet会假 对 复出 但 如 发 同 hashcode 值 对 ， 会 equals() hashcode 对 否 同 如 两 同，HashSet 就不会 其加入 作 功 如 不同 ，就会 列到其他位 （ Java启 书 Head first java 二 ） 们就大大减少了 equals ， 应就大大 了 度

与 的相关规定

- 如 两个对 ，则hashcode 定也 同
- 两个对 ，对两个对 分别 equals 回true
- 两个对 同 hashcode值，它们也不 定

因此，equals 方法被覆盖过，则 hashCode 方法也必须被覆盖

hashCode() 的默认行为是对堆上的对象产生独特值。如果没有重写 hashCode()，则该 class 的两个对象无论如何都不会相等（即使这两个对象指向相同的数据）

对象的相等与指向他们的引用相等，两者有什么不同?

- 对 内存中存 内容 否 他们 向 内存地址 否

值传递

当一个对象被当作参数传递到一个方法后，此方法可改变这个对象的属性，并可返回变化后的结果，那么这里到底是值传递还是引用传递

- 值传 Java 只 参 值传 个对 实例作为 个参 传 到 中 ，参 值就 对 对 对 属 可以在 中 变，但对对 变 不会 响到

为什么 中只有值传递

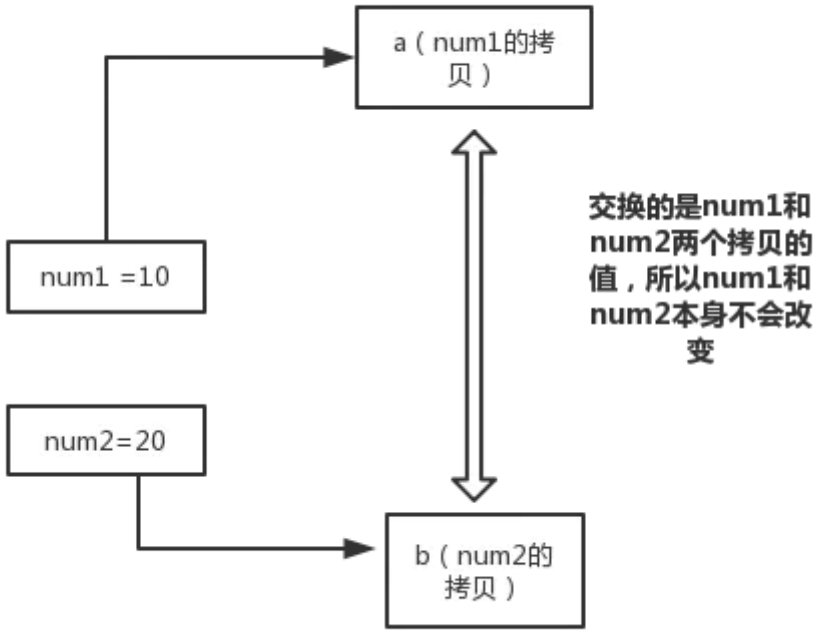
- 先回 下在 序 中 关将参 传 （ 函 ） 些专业 **按值调用** **表示方法接收的是调用者提供的值，而按引用调用（ 表示方法接收的是调用者提供的变量地址。一个方法可以修改传递引用所对应的变量值，而不能修改传递值调用所对应的变量值。** 它 各 序 (不只 Java)中 参 传
- **程序设计语言总是采用按值调用。也就是说，方法得到的是所有参数值的一个拷贝，也就是说，方法不能修改传递给它的任何参数变量的内容。**
  - **下面通过 个例子来给大家说明**

```
public static void main(String[] args) {
    int num1 = 10;
    int num2 = 20;

    swap(num1, num2);

    System.out.println("num1 = " + num1);
    System.out.println("num2 = " + num2);
}

public static void swap(int a, int b) {
    int temp = a;
    a = b;
    b = temp;

    System.out.println("a = " + a);
    System.out.println("b = " + b);
}
```

- :

  a = 20 b = 10 num1 = 10 num2 = 20

- :



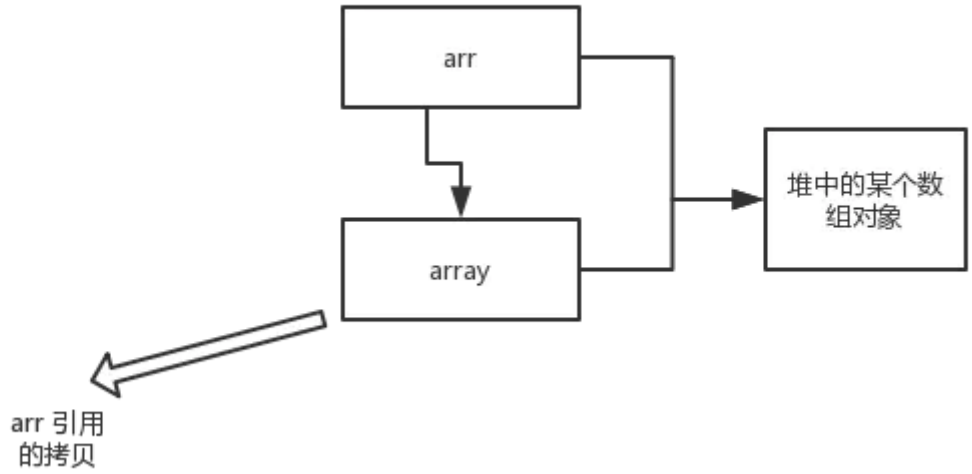
- 在swap 中，a b 值 交 ，并不会 响到 num1 num2 因为，a b中 值，只 从 num1 num2 复制 也就 ，a b 于num1 num2 副 ，副 内容 么修 ， 不会 响到原件

通过上面例子，我们已经知道了一个方法不能修改一个基本数据类型的参数，而对象引用作为参数就不一样，请看 example.

```
    public static void main(String[] args) {
        int[] arr = { 1, 2, 3, 4, 5 };
        System.out.println(arr[0]);
        change(arr);
        System.out.println(arr[0]);
    }

    public static void change(int[] array) {
        // 将数组的第一个元素变为0
        array[0] = 0;
    }
```

- :

  1 0
- :



- array 初始化 arr 也就 个对 ，也就 array 和 arr 向 同 个 对 因 ，外 对 对 变会反 到 对应 对 上

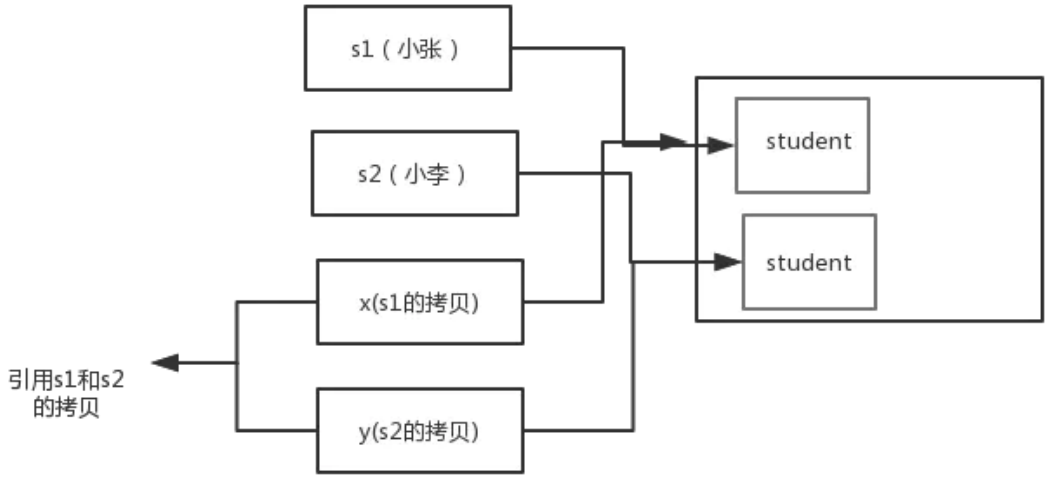
通过 example2 我们已经看到，实现一个改变对象参数状态的方法并不是一件难事。理由很简单，方法得到的是对象引用的拷贝，对象引用及其他的拷贝同时引用同一个对象。

很多程序设计语言（特别是，C++和Pascal)提供了两种参数传递的方式：值调用和引用调用。有些程序员（甚至本书的作者）认为Java程序设计语言对对象采用的是引用调用，实际上，这种理解是不对的。由于这种误解具有一定的普遍性，所以下面给出一个反例来详细地阐述一下这个问题。

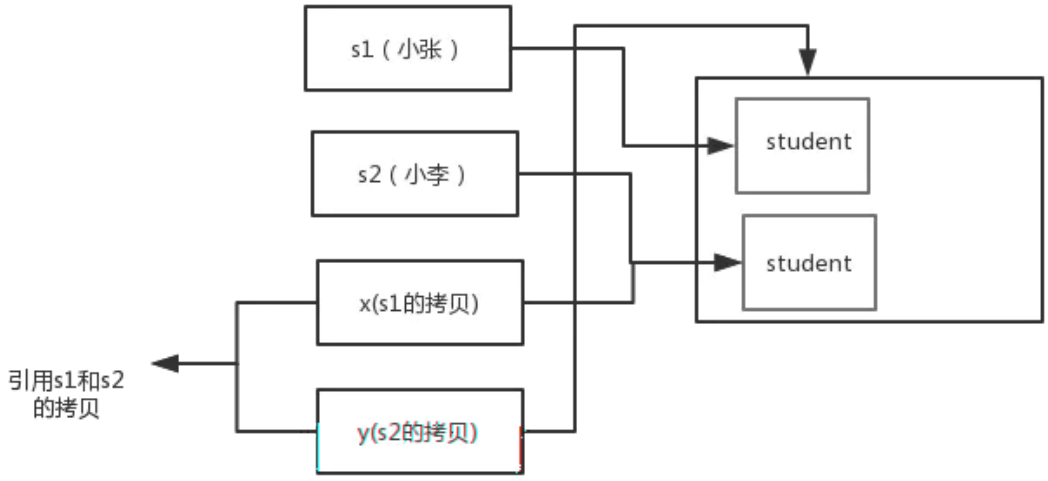
```
public class Test {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
  Student s1 = new Student("小张");
        Student s2 = new Student("小李");
        Test.swap(s1, s2);
        System.out.println("s1:" + s1.getName());
        System.out.println("s2:" + s2.getName());
    }

    public static void swap(Student x, Student y) {
        Student temp = x;
        x = y;
        y = temp;
        System.out.println("x:" + x.getName());
        System.out.println("y:" + y.getName());
    }
}
```

- :

  x:小 y:小 s1:小 s2:小
- :
- 交 之前:



- 交　之后:



- 　　上　两　图可以　　出:`方法并没有改变存储在变量 s1 和 s2 中的对象引用。swap方法的参数x和y被初始化为两个对象引用的拷贝，这个方法交换的是这两个拷贝`
-
  - `Java程序设计语言对对象采用的不是引用调用，实际上，对象引用是按值传递的。`
- 下　再　　下Java中　　参　　使　　况:
  - 　个　　不　修　　个基　　型　参　(即　值型　布尔型
  - 　个　　可以　变　个对　参
  - 　个　　不　　对　参　　个　　对

值传递和引用传递有什么区别

- 值传　:　　在　　，传　　参　　值　　传　，传　　值　　，也就　　传　后就互不　关了
- 　　传　:　　在　　，传　　参　　传　，其实传　　地址，也就　变　　对应　内存　　地址　传　　值　　，也就　　传　前和传　后　　向同　个　（也就　同　个内存　　）

包

中常用的包有哪些

- java.lang：个 基 ；
- java.io： 入 出 关 ， 如 件 作 ；
- java.nio：为了完善 io 包中 功 ， io 包中 写 个 包;
- java.net： 与 关 ；
- java.util： 个 助 ， 别 合 ；
- java.sql： 个 库 作

和 有什么区别

- 刚开始 候 JavaAPI 包 java 开头 包，javax 只 展 API 包 使 ，javax 展 为 Java API 分 但 ，将 展从 javax 包 动到 java 包将 太 了， 会 坏 堆 代 因 ， 决定 javax 包将 为 准API 分

所以，实际上java和javax没有区别。这都是一个名字。

流

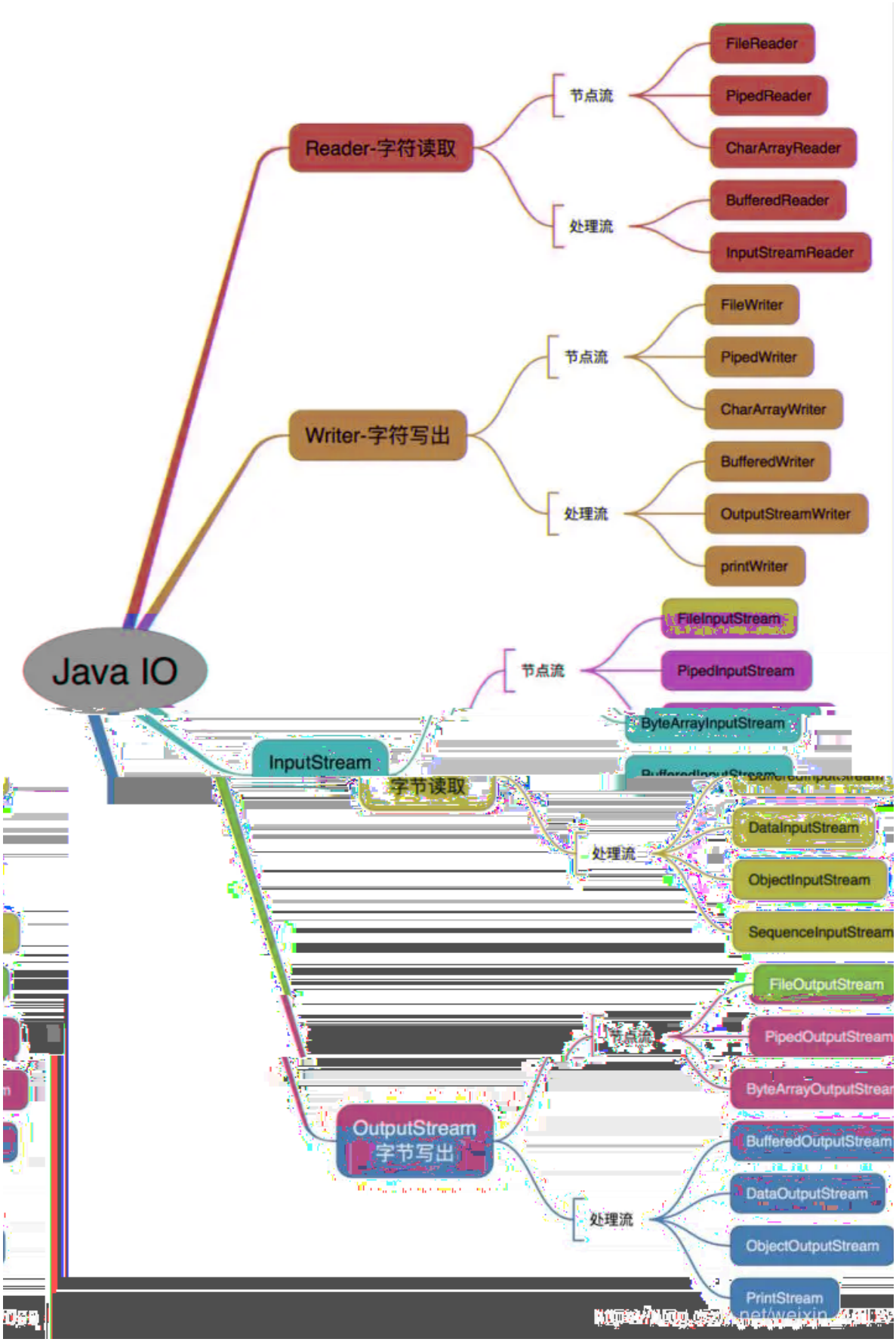
中 流分为几种

- 向分，可以分为 入 和 出 ；
- 作单元划分，可以划分为字 和字 ；
- 划分为 和处

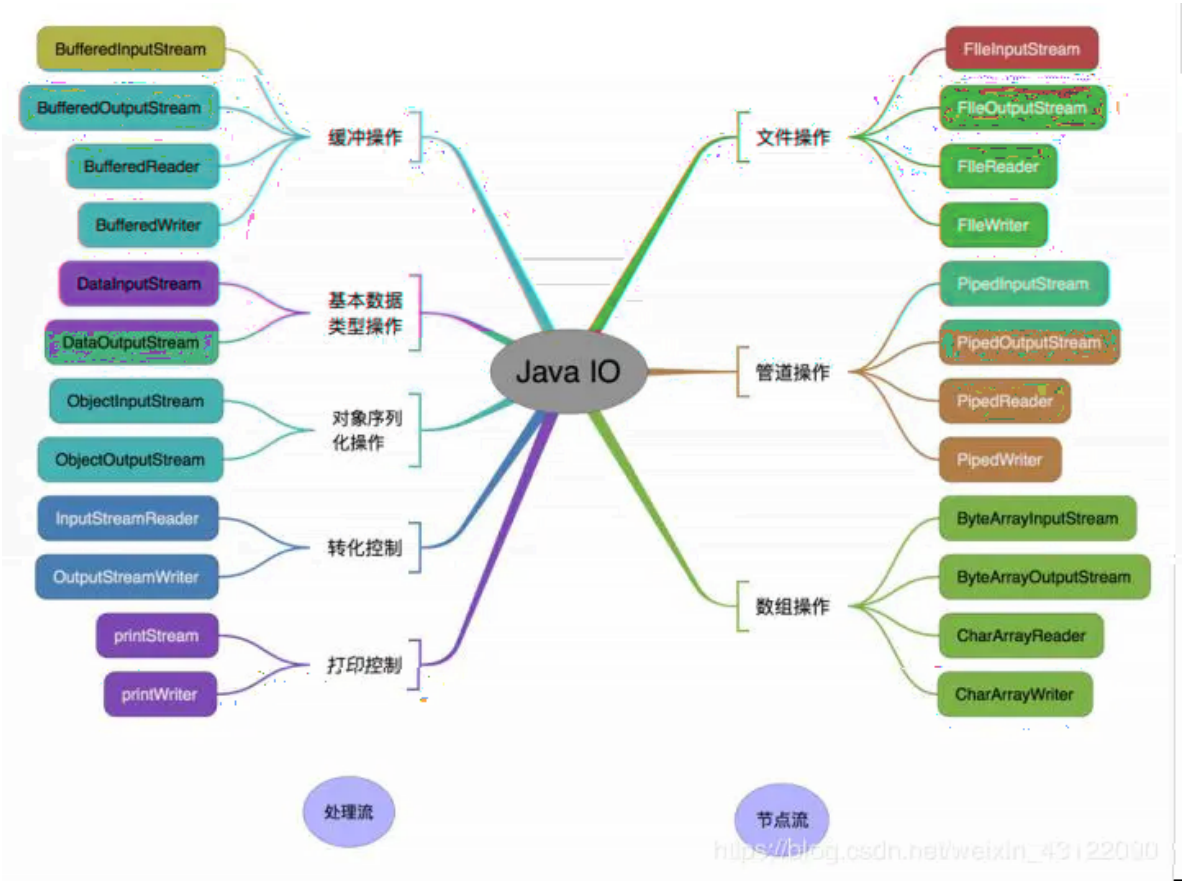
Java Io流共涉及40多个类，这些类看上去很杂乱，但实际上很有规则，而且彼此之间存在非常紧密的联系，Java IO流的40多个类都是从如下4个抽象类基类中派生出来的。

- InputStream/Reader: 入 基 ，前 字 入 ，后 字 入
- OutputStream/Writer: 出 基 ，前 字 出 ，后 字 出

按操作方式分类结构图：

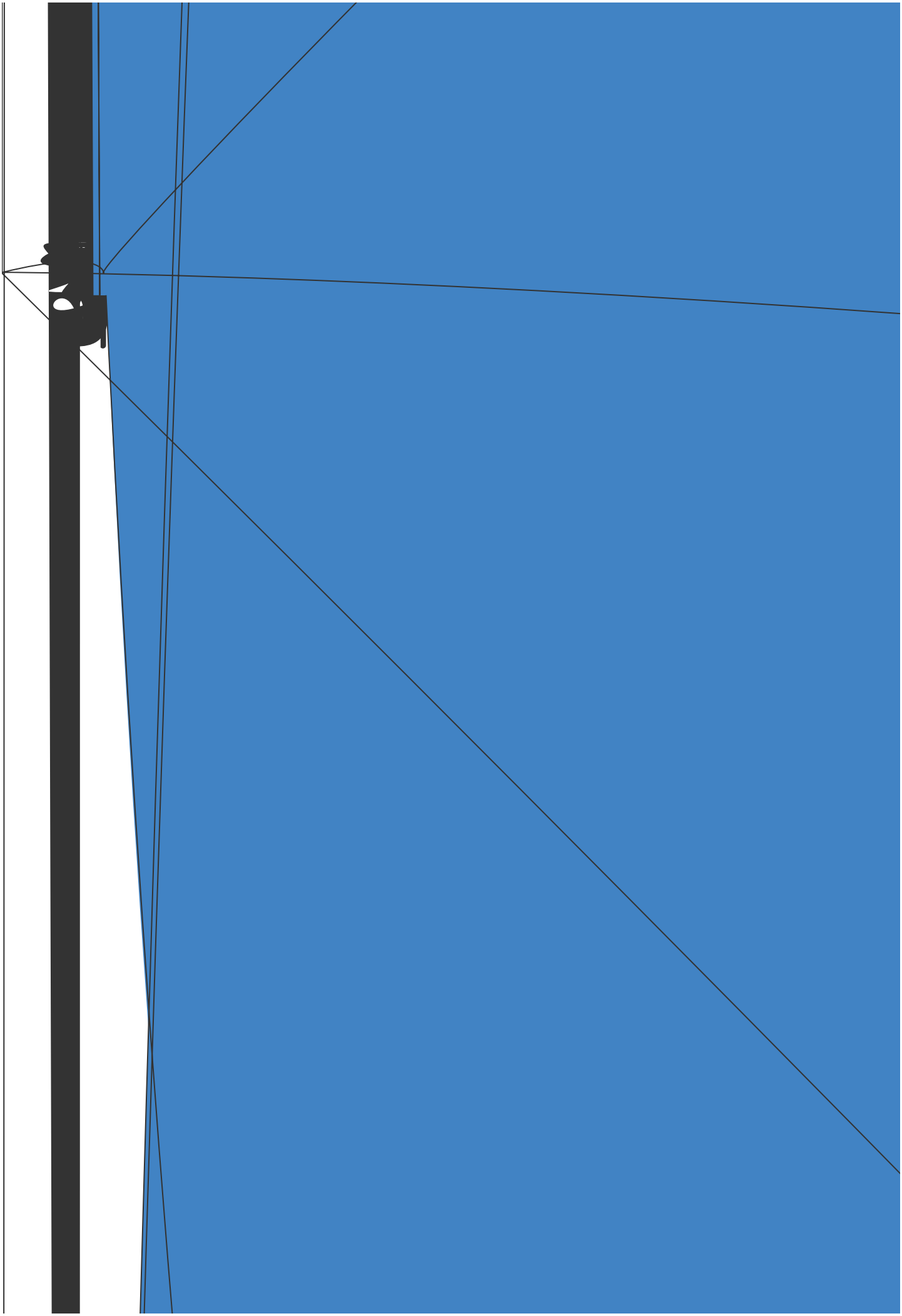

Java IO
Reader-字符读取
节点流
FileReader
PipedReader
CharArrayReader
处理流
BufferedReader
InputStreamReader
Writer-字符写出
节点流
FileWriter
PipedWriter
CharArrayWriter
处理流
BufferedWriter
OutputStreamWriter
printWriter
InputStream
字节读取
节点流
FileInputStream
PipedInputStream
ByteArrayInputStream
处理流
DataInputStream
ObjectInputStream
SequenceInputStream
OutputStream
字节写出
节点流
FileOutputStream
PipedOutputStream
ByteArrayOutputStream
处理流
BufferedOutputStream
DataOutputStream
ObjectOutputStream
PrintStream
net/weixin_

按操作对象分类结构图：



有什么区别

-
  - BIO：Block IO 同　　塞　IO，就　　们平常使　　传　IO，它　　　　单使便，并发处　　力低
  - NIO：Non IO 同　　塞 IO，　传　IO　升　，客　　和　务器　　Channel（　　），实　了多　复
  - AIO：Asynchronous IO　NIO　升　，也叫 NIO2，实　了　　堵塞 IO，　　IO 作基于事件和回　　制
- 　回
  - 　　同　　塞I/O　，　　取写入　　塞在　个　　内　　其完　在　动　　不　　别　(小于单　1000)　　况下，　　型　　不　，可以　个　　专　于　己　I/O 并且　　型　单，也不　　多　　就　　个天　　，可以　冲　些　　处　不了　　　　但　，对十万　　万　　　候，传　BIO　型　　为力　　因　，　们　I/O 处　　型　应对　　并发
  - 　　NIO　　同　　塞　I/O　型，在Java 1.4 中　入了NIO　　，对应 java.nio 包，　供了 Channel , Selector，Buffer　　NIO中　N可以　　为Non-blocking，不单　　New　它　　向　冲　，基于　　I/O　作　　NIO　供了与传 BIO　型中　`Socket` 和 `ServerSocket`　对应　`SocketChannel` 和 `ServerSocketChannel` 两　不同　套　字　　实　,两　　　　塞和　　塞两　　塞　　使　就像传　中　　　，　　单，但　　和可　　不好；　　塞　好与之　反　对于低　　低并发　应　　序，可以使　同　　塞I/O　　升开发　　和好　　；对于　　并发　（　）应　，应使　NIO　　塞　　开发
  - 　　AIO 也就　NIO 2　在 Java 7 中　入了 NIO　　NIO 2,它　　塞　IO　型　　IO　基于事件和回　　制实　，也就　应　作之后会　回，不会堵塞在　，　后台处　完　，　作　会　　应　　后　　作　AIO



获取反射的三种方法

1. new对 实 反射 制 2. 实 反射 制 3. 名实 反射 制

```
public class Student {
    private int id;
    String name;
    protected boolean sex;
    public float score;
}

public class Get {
    //获取反射机制三种方式
  public static void main(String[] args) throws ClassNotFoundException {
        //方式一(通过建立对象)
        Student stu = new Student();
        Class classobj1 = stu.getClass();
        System.out.println(classobj1.getName());
        //方式二（所在通过路径-相对路径）
  Class classobj2 = Class.forName("fanshe.Student");
        System.out.println(classobj2.getName());
        //方式三（通过类名）
  Class classobj3 = Student.class;
        System.out.println(classobj3.getName());
    }
}
```

常用

相关

字符型常量和字符串常量的区别

1. 上: 字 常 单 号 个字 字 串常 双 号 干个字
2. 含义上: 字 常 于 个 值(ASCII值),可以参加 字 串常 代 个地址值(字 串在内存中存 位)
3. 占内存大小 字 常 只占 个字 字 串常 占 干个字 (少 个字)

什么是字符串常量池?

- 字 串常 位于堆内存中，专 存储字 串常 ，可以 内存 使 ， 免开 多块 存储 同 字 串，在创建字 串 JVM 会 先 字 串常 ，如 字 串已 存在 中，则 回它 ，如 不存在，则实例化 个字 串 到 中，并 回其

是最基本的数据类型吗

- 不 Java 中 基 型只 8 个： byte short int long float double char boolean； 了基 型（primitive type），剩下 型（referencetype），Java 5 以

后 入 举 型也 型

这是很基础的东西，但是很多初学者却容易忽视，Java 的 8 种基本数据类型中不包括 String，基本数据类型中用来描述文本数据的是 char，但是它只能表示单个字符，比如 ‘a’,‘好’ 之类的，如果要描述一段文本，就需要用多个 char 类型的变量，也就是一个 char 类型数组，比如“你好” 就是长度为2的数组 char\[\] chars = {‘你’,‘好’};

但是使用数组过于麻烦，所以就有了 String，String 底层就是一个 char 类型的数组，只是使用的时候开发者不需要直接操作底层数组，用更加简便的方式即可完成对字符串的使用。

有哪些特性

- 不变 ：String 只 字 串， 个典型 immutable 对 ，对它 任何 作，其实 创建 个 对 ，再 向 对 不变 主 作 在于 个对 多 共享并 ，可以保
- 常 优化：String 对 创建之后，会在字 串常 中 存，如 下 创建同 对 ，会 回 存
- final：使 final 定义 String ， String 不 ， 了 安全

为什么是不可变的吗?

- 单 就 String 利 了final修 char 型 存储字 ， 如下图 以：

  /** The value is used for character storage. */ private final char value[];

真的是不可变的吗?

- 如 别人 个 ，回 不可变就可以了 下 只 大家 两个 代 例子：

  不可变但不代表引用不可以变

```
String str = "Hello";
str = str + " World";
System.out.println("str=" + str);
```

- ：

  str=Hello World

- ：
- 实 上，原 String 内容 不变 ，只 str 原 向"Hello" 内存地址 为 向"Hello World" 内存地址 已，也就 多开 了 块内存区域 "Hello World"字 串

通过反射是可以修改所谓的 不可变 对象

```
// 创建字符串"Hello World", 并赋给引用s
String s = "Hello World";

System.out.println("s = " + s); // Hello World

// 获取String类中的value字段
```

```
Field valueFieldOfString = String.class.getDeclaredField("value");

// 改变value属性的访问权限
valueFieldOfString.setAccessible(true);

// 获取s对象上的value属性的值
char[] value = (char[]) valueFieldOfString.get(s);

// 改变value所引用的数组中的第5个字符
value[5] = '_';

System.out.println("s = " + s); // Hello_World
```

- :

  s = Hello World s = Hello_World

- :

- 反射可以 员， 后反射出String对 中 value属 ， 变 value 变 但 们不会 么做， 只 单 下 个东

是否可以继承 类

- String final ，不可以

与 一样吗?

- 不 ，因为内存 分 不 String str="i" ，java 会将其分 到常 中; String str=new String(“i”) 则会 分到堆内存中

创建了几个字符串对象

- 两个对 ， 个 区 "xyz"， 个 new创建在堆上 对

  String str1 = "hello"; //str1 向 区 String str2 = new String("hello"); //str2 向堆上 对 String str3 = "hello"; String str4 = new String("hello"); System.out.println(str1.equals(str2)); //true System.out.println(str2.equals(str4)); //true System.out.println(str1 == str3); //true System.out.println(str1 == str2); //false System.out.println(str2 == str4); //false System.out.println(str2 == "hello"); //false str2 = str1; System.out.println(str2 == "hello"); //true

如何将字符串反转?

- 使 StringBuilder stringBuffer reverse()
- 例代 :

  // StringBuffer reverse StringBuffer stringBuffer = new StringBuffer(); stringBuffer. append("abcdefg"); System. out. println(stringBuffer. reverse()); // gfedcba // StringBuilder reverse StringBuilder stringBuilder = new StringBuilder(); stringBuilder. append("abcdefg"); System. out. println(stringBuilder. reverse()); // gfedcba

数组有没有 方法? 有没有 方法

- length() , length 属 String length() JavaScript中, 字 串 度 length 属 到 , 容 和 Java

类的常用方法都有那些?

- indexOf(): 回 定字
- charAt(): 回 定 处 字
- replace(): 字 串
- trim(): 去 字 串两
- split(): 分割字 串, 回 个分割后 字 串
- getBytes(): 回字 串 byte 型
- length(): 回字 串 度
- toLowerCase(): 将字 串 小写字
- toUpperCase(): 将字 串 大写字
- substring(): 取字 串
- equals(): 字 串

在使用 的时候，用 做 有什么好处?

- HashMap 内 实 key hashcode 定 value 存储位 ，因为字 串 不可变 ，以 创建字 串 ，它 hashcode 存下 ，不 再 , 以 于其他对

和 、 的区别是什么? 为什么是不可变的

可变性

- String 中使 字 保存字 串，private final char value[], 以string对 不可变 StringBuilder与StringBuffer AbstractStringBuilder ，在AbstractStringBuilder中也 使 字 保存字 串，char[] value, 两 对 可变

线程安全性

- String中 对 不可变 ，也就可以 为常 , 安全 AbstractStringBuilder StringBuilder与StringBuffer 公共 ，定义了 些字 串 基 作，如expandCapacity append insert indexOf 公共 StringBuffer对 加了同 对 加了同 , 以 安全 StringBuilder并 对 加同 , 以 安全

性能

- 对String 型 变 候, 会 个 String对 , 后将 向 String 对 StringBuffer 会对StringBuffer对 作, 不 对 并 变对 同 况下使 StirngBuilder 使 StringBuffer 仅 10%~15% 左右 升，但却 冒多 不安全

对于三者使用的总结

- 如 作少 = String

- 单　　作字　串　冲区 下　作大　　= StringBuilder
- 多　　作字　串　冲区 下　作大　　= StringBuffer

相关

包装类相关

自动装箱与拆箱

- **装箱**：将基　　型　它们对应　　型包　　；
- **拆箱**：将包　　型　　为基　　型；

和　　有什么区别

- Java　　个　乎　　向对　　，但　为了　　便　　入了基　　型，但　为了　够将　些基　　型　　对　　作，Java 为　　个基　　型　　入了对应　包　型（wrapper class），int　包　　就　Integer，从 Java 5 开始　入了　动　　/　　制，使　二　可以　互
- Java 为　个原始　型　供了包　　型：
  - 原始　型: boolean，char，byte，short，int，long，float，double
    - 包　　型：Boolean，Character，Byte，Short，Integer，Long，Float，Double

与　　相等吗

- 对于对　　型：==　　对　　内存地址
- 对于基　　型：==　　值

`如果整型字面量的值在-128到127之间，那么自动装箱时不会new新的Integer对象，而是直接引用常量池中的Integer对象，超过范围 a1==b1的结果是false`

```
public static void main(String[] args) {
    Integer a = new Integer(3);
    Integer b = 3;  // 将3自动装箱成Integer类型
  int c = 3;
    System.out.println(a == b); // false 两个引用没有引用同一对象
  System.out.println(a == c); // true a自动拆箱成int类型再和c比较
  System.out.println(b == c); // true

    Integer a1 = 128;
    Integer b1 = 128;
    System.out.println(a1 == b1); // false

    Integer a2 = 127;
    Integer b2 = 127;
    System.out.println(a2 == b2); // true
}
```