## 注解：Annotation

Java 5.0 新增注解特性，是一种特殊的接口。

注解是代码的附属信息，遵循一个基本原则：注解不能直接干扰程序代码的运行，无论增加或删除注解，代码都能够正常运行。

Java语言解释器会忽略这些注解，而由第三方工具负责对注解进行处理。

@interface修饰符定义注解，一个注解可以拥有多个成员，成员声明和接口方法声明类似，成员声明有以下几点限制：

1.成员以无入参，无抛出异常的方式声明

2.可以通过default为成员指定默认值

3.成员类型只能是基本数据类型及其包装类型、String、Class、enum、注解类型

4.如果只有一个成员，则成员的名字必须取为value()

元注解(Meta-Annotation)：Java预定义的注解，被Java编译器使用，会对注解类的行为产生影响。常用的元注解有：

@Retention(RetentionPolicy.RUNTIME) – 注解类信息保留期限

1. RetentionPolicy.SOURCE:注解信息仅保存在目标类的源码文件中，对应的字节码文件将不再保留

2. RetentionPolicy.CLASS:注解信息将进入目标类的字节码文件中，但类加载器加载字节码文件时不会将注解加载到JVM中，即运行期不能获取注解信息

3. RetentionPolicy.RUNTIME:注解信息在目标类加载到JVM后依然保留，在运行期可以通过反射机制读取类中的注解信息。

@Target(ElementType.METHOD):注解类的应用范围

## 反射：Reflect

源文件 🡪 类文件(二进制) 🡪 机器码

类对象:Class

元数据:方法、字段、构造函数、注解等

类文件需包含以下部分：

* 魔法数(CA FE BA BE)
* 使用的类文件标准版本
* 当前类的常量池
* 访问标志(abstract、public等)
* 当前类的名称
* 继承信息
* 实现的接口
* 字段
* 方法
* 属性

反射是在运行时审查、操作和修改对象的能力，可以修改对象的结构和行为。

## Swich

swich语句是一个分支语句，用来替代多层if判断的场景。使用结构如下：

swich(n){

case 1:

语句块1

break;

…

default:

语句块2

break;

}

其中n是一个表达式，这个表达式的返回值必须是：整型(int、short、char、byte)或其包装类型、String、枚举类型。

关于类型的说明：

1.为什么只能是整型，不能是float和double？

浮点型在计算机中只是一种近似的表示，总是会有精度的，因此不能很准确的进行比较。float是32位近似值，至少有6个有效数字；double是64位近似值，至少有15个有效数字。

2.long也是一种整型，为什么不能用long？

swich的本质是一种穷举，对于long类型来说，占64位，数据范围很大(19位数字)，对于int类型而已，32位已经能表示21亿的数据范围了，已经足够大了，所以根本无需使用到范围更大的long类型。

3.swich为什么能用String？

在Java 7之前，swich只支持整型和枚举，Java 7开始添加对String的支持，其本质，在内部是比较String的hashCode()返回值(int类型).

4.swich支持枚举又是何道理？

Java中的枚举是类的变种，而且运行使用的值很少，是能穷举完的。

## 枚举：Enum

Java中的枚举是类的变种，功能有限，而且允许使用的值很少。

由于枚举是特殊的类，所以可以拥有成员(字段和方法)，如果字段或方法有主体的话，那么实例列表后面必须加上分号。

枚举实例是由Java运行时创建的，在外部是不能实例化的，所以构造函数需要声明为private。

枚举有如下的特性：

1) 都隐式扩展自java.lang.Enum类

2) 不能泛型化

3) 可以实现接口

4) 不能被继承

5) 只能有一个private(或default)的构造方法

6) 如果枚举中的所有值都有实现主体，那么只能定义为抽象方法

## String

Java中的字符串是由一系列的Unicode字符组成。

在Java中，只有字符串常量是共享的，使用+或substring等操作产生的结果并不是共享的。

+ 运算符连接字符串时，实际上是先创建一个使用StringBuilder对象表现的工作区，javac会自动创建上述代码。

String是不可变类

StringBuilder可变的

StringBuffer 同步的，线程安全

当使用 s2 = "Monday" 这样的表达式创建字符串的时候，JAVA首先会在这个String缓冲池中寻找相同值的对象，找到了就将引用赋值给s2，找不到就把Monday添加到缓冲池，再把引用赋值给s2；

使用了 new 操作符，会创建一个新的String对象。

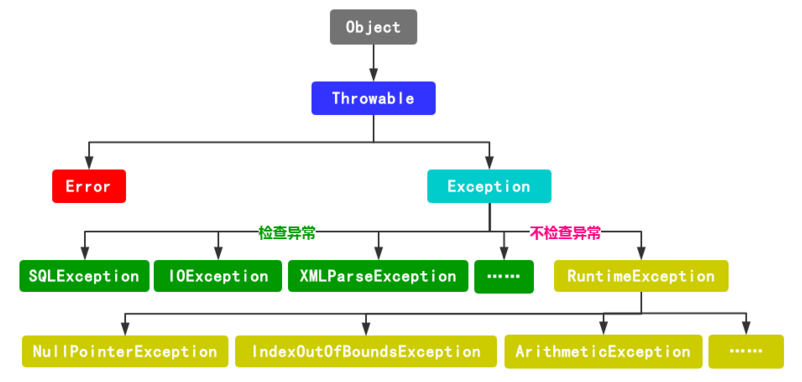
s2.intern()会将s2中值复制到常量池，如果常量池已经存在该字符串，则直接返回该字符串的引用。

## 基础类型

基本类型都是存在栈里面的，Int a=3;当你int b=3的时候，jvm会干嘛呢？首先，它会去栈里面找是不是已经有一个数据为3了。如果不是的话新建一个地址，是的话，就把b指向这个地址，**所以对于基本类型来说，当值相同的时候，地址也是相同的**，所以当你对基本类型==的时候，内容相同的肯定返回的是true。因为实质上，他们的地址也是相同的。

但是对于其他类型，都是存在堆里面的动态分配内存，所以每一次新建一个数据都会动态分配一个内存地址，所以==的时候，当然内存地址是不一样的。  
要记住，**==永远是用来比较内存中的地址的，基本类型感觉上是在比较内容，实际上还是在比较地址！**

## 异常体系



Error用来表示编译时和系统错误，一般我们不用关心。

Exception又分为不检查异常(运行时异常)和检查异常。

**运行时异常**类对应于编译错误，它是指Java程序在运行时产生的由解释器引发的各种异常。出现运行错误往往表示代码有错误，如：算数异常（如被0除）、下标异常（如数组越界）等。

**非运行时异常**Non\_RuntimeException类及其子类的实例，又称为检查异常。Java编译器利用分析方法或构造方法中可能产生的结果来检测Java程序中是否含有检测异常的处理程序，对于每个可能的可检测异常，方法或构造方法的throws子句必须列出该异常对应的类。在Java的标准包java.lang java.util 和 java.net 中定义的异常都是非运行异常。

## 数据库事务基础知识

### 事务的特性

数据库事务的4个特性：原子性(Atomic)、一致性(Consistency)、隔离性(Isolation)、持久性(Durability),简称ACID

原子性：组成事务的多个数据库操作是一个不可分割的原子单元；

一致性：事务操作成功后，数据库所处的状态和它的业务规则是一致的；

隔离性：并发数据操作时，不同的事务拥有各自的数据空间，操作不会对对方产生干扰；

持久性：一旦事务提交成功后，事务中的所有数据操作都必须被持久化到数据库中。

### 数据并发的问题

脏读(dirty read)、不可重复读、幻象读、第一类丢失更新、第二类丢失更新。

1.脏读：A事务读取了B事务尚未提交额更改数据，B事务最后没成功

2.不可重复读：A事务两次读取的数据不一致，两次读取中间有其他事务提交了**数据更改**。

3.幻象读：A事务两次读取的数据不一致，两次读取中间有其他事务提交了**数据新增**。

4.第一类丢失更新：A数据**撤销**时，把已经提交的B事务的更新数据覆盖了。

5.第二类丢失更新：A事务**成功提交**，覆盖了B事务已经提交的数据。

### 数据库锁

锁的粒度：表锁、页锁、行锁

锁的种类：共享锁(S:Share)、独占锁(X)、更新锁(U:Update)

更改数据，施加行独占锁

### 事务隔离级别

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 隔离级别 | 脏读 | 不可重复读 | 幻想读 | 第一类丢失更新 | 第二类丢失更新 |
| Read UnCommitted | **允许** | **允许** | **允许** | 不允许 | **允许** |
| Read Committed | 不允许 | **允许** | **允许** | 不允许 | **允许** |
| Repeatable Read | 不允许 | 不允许 | **允许** | 不允许 | 不允许 |
| Serializable | 不允许 | 不允许 | 不允许 | 不允许 | 不允许 |

**Read UnCommitted**:读未提交，就是一个事务可以读取另一个未提交事务的数据。

**Read Committed:** 读提交，就是一个事务要等另一个事务提交后才能读取数据。

**Repeatable Read：**重复读，就是在开始读取数据（事务开启）时，不再允许修改操作。

**Serializable ：**最高的事务隔离级别，事务串行化顺序执行。

**值得一提的是：大多数数据库默认的事务隔离级别是Read committed，比如Sql Server , Oracle。Mysql的默认隔离级别是Repeatable read。**

## ThreadLocal

1.2版本引入，5.0版本支持泛型

如果某个对象是非线程安全的，在多线程环境下，对对象的访问必须采用synchronized进行线程同步；线程同步会降低并发性，影响系统性能。

TreadLocal为解决多线程程序的并发问题提供一种新的思路；ThreadLocal是一个保存线程本地化对象的容器；当运行于多线程环境的某个对象使用ThreadLocal维护变量时，ThreadLocal为每个使用该变量的线程分配一个独立的变量副本。

ThreadLocal和线程同步机制都是为了解决多线程中相同变量的访问冲突问题。

在同步机制中，通过对象的锁机制保证同一时间只有一个线程访问变量，该变量是多个线程共享的；

ThreadLocal为每个线程提供一个独立的变量副本，从而隔离了多个线程对访问数据的冲突；

概况而言，对于多线程资源共享的问题，同步机制采用“时间换空间”的方式：访问串行化，对象共享化；而ThreadLocal采用“空间换时间”的方式：访问并行化，对象独立化。

## Java 集合框架



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 类 | 内部表现 | 首次出现版本 | 元素顺序 | 成员限制 | 基本操作 | 迭代性能 | 备注 |
| HashSet | 哈希表 | 1.2 | 无 | 无 | O(1) | O(capacity) | 最佳通用实现 |
| LinkedHashSet | 哈希链表 | 1.2 | 插入顺序 | 无 | O(1) | O(n) | 保留插入顺序 |
| EnumSet | 位域 | 5.0 | 枚举声明 | 枚举类型 | O(1) | O(n) | 值不能为null |
| TreeSet | 红黑树 | 1.2 | 升序排列 | 可比较 | O(log(n)) | O(n) | 元素需要实现Comparable或Comparator接口 |
| CopyOnWriteArraySet | 数组 | 5.0 | 插入顺序 | 无 | O(n) | O(n) | 线程安全 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 类 | 表示方式 | 首次出现版本 | 随机访问 | 备注 |
| ArrayList | 数组 | 1.2 | 能 | 最佳全能实现 |
| LinkedList | 双向链表 | 1.2 | 否 | 高效插入和删除 |
| CopyOnWriteArrayList | 数组 | 5.0 | 能 | 线程安全,遍历快,修改慢 |
| Vector | 数组 | 1.0 | 能 | 已过时,同步 |
| Stack | 数组 | 1.0 | 能 | 扩展Vector,添加了push()/pop()/peek(),过时，用Deque替代 |

列表中的元素在集合里是有位置的(下标)

数组和数组列表的缺点:从数组的中间位置删除元素要付出很大的代价，原因是数组中处于被删除元素之后的所有元素都要向数据的前端移动。

Java中的链表都是双向列表，删除和添加元素效率很高，不支持随机访问，指定访问链表中第n个元素，需要越过前面n-1个元素。

使用列表唯一的理由是：尽可能减少在列表中间插入或删除元素所服务的代价，如果列表元素不多，完全可以使用ArrayList。

Vector的所有方法都是同步的，多线程环境下可以安全的访问Vector对象，但同步操作需要耗费大量时间；而ArrayList方法不是同步的。

HashSet为对象计算hashCode，然后再存到对应的地方(称为桶)，基于链表实现，每个链表成为桶。要想查找表中对象的位置，先计算对象的hashCode，然后与桶的总数取余，得到的结果就是保存这个元素的桶的索引。

TreeSet是一个有序的集合，可以以任意顺序将元素插入到集合中，每个值将自动按照元素排序后的顺序呈现。

队列，从尾部添加元素，头部删除元素，Java 6引入Deque接口和ArrayDeque、LinkedList类实现



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 类 | 表现方式 | 出现版本 | null键 | null值 | 备注 |
| HashMap | 哈希表 | 1.2 | 是 | 是 | 通用实现 |
| ConcurrentHashMap | 哈希表 | 5.0 | 否 | 否 | 通用线程安全实现 |
| ConcurrentSkipListMap | 哈希表 | 6.0 | 否 | 否 | 专用的线程安全实现 |
| EnumMap | 数组 | 5.0 | 否 | 是 | 键是枚举类型 |
| LinkedHashMap | 哈希表+列表 | 1.4 | 是 | 是 | 保留插入或访问顺序 |
| TreeMap | 红黑树 | 1.2 | 否 | 是 | 按键排序 |
| IdentityHashMap | 哈希表 | 1.4 | 是 | 是 | 比较时使用==,而不是equals() |
| WeakHashMap | 哈希表 | 1.2 | 是 | 是 | 不会阻止垃圾回收键 |
| HashTable | 哈希表 | 1.0 | 否 | 否 | 过时，同步 |
| Properties | 哈希表 | 1.0 | 否 | 否 | 使用String类的方法扩展Hashtable |

Map，HashMap与TreeMap，TreeMap会对Key进行排序。

最终的集合框架

获取长度：

// args.length; 数组长度

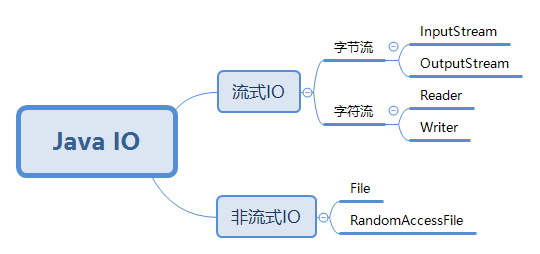
// "".length(); 字符串长度

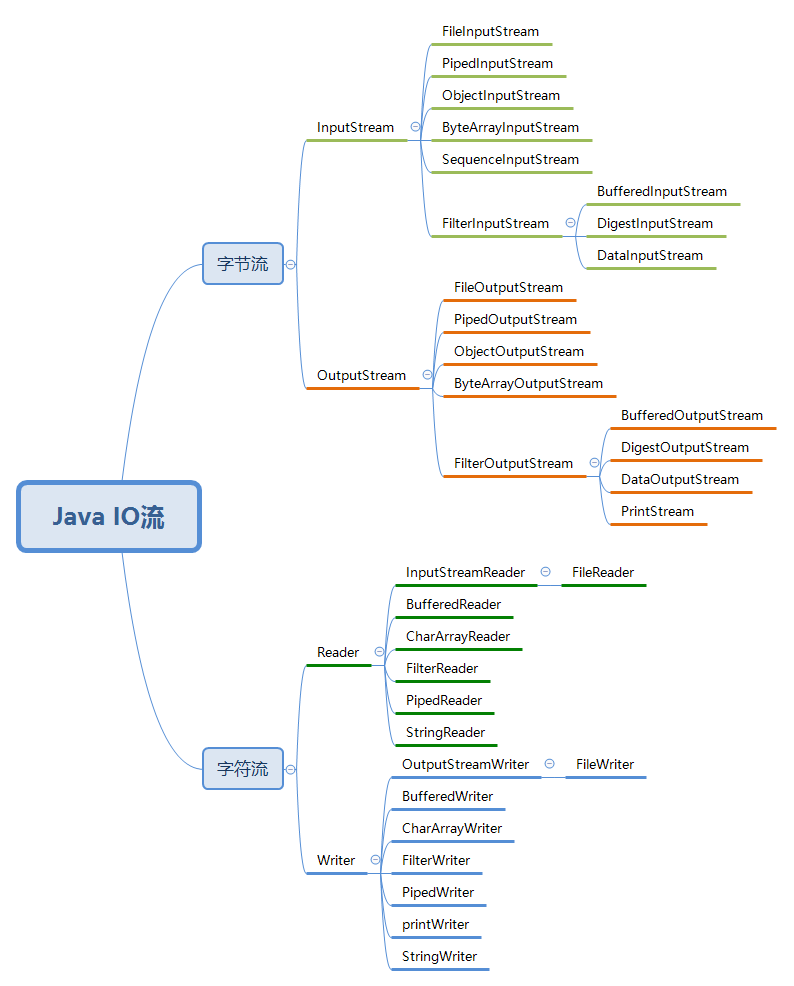
// list.size() 集合框架长度

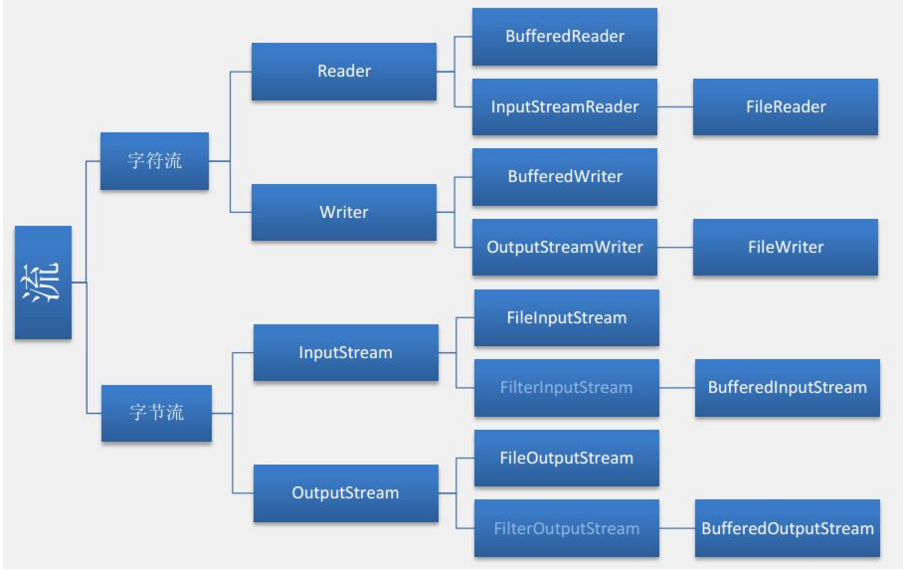
// set.size() 集合框架长度

// map.size() 集合框架长度

## Java IO体系







**那么字节流和字符流又有什么区别呢？**

1.字节流也称为原始数据，需要用户读入后进行相应的编码转换。而字符流的实现是基于自动转换的，读取数据时会把数据按照JVM的默认编码自动转换成字符。

2.字符流处理的单元为2个字节的Unicode字符，分别操作字符、字符数组或字符串，而字节流处理单元为1个字节，操作字节和字节数组。

所以字符流是由Java虚拟机将字节转化为2个字节的Unicode字符为单位的字符而成的。

3.字节流可用于任何类型的对象，包括二进制对象，而字符流只能处理字符或者字符串，字节流提供了处理任何类型的IO操作的功能，但它不能直接处理Unicode字符，而字符流就可以；

**基于以上的区别，那么什么情况下用字符流，什么情况下用字节流呢？**

如果是音频文件、图片、歌曲，就用字节流好点；如果是中文（文本）的，用字符流更好；

## Java命令行工具

* javac
* java
* jar
* javadoc
* jdeps
* jps
* jstat
* jstatd
* jinfo
* jstack
* jmap
* javap

**javac**:**java源码编译器，把.java源码文件编译成字节码.class文件**

javac some/package/MyClass.java

**java:启动Java虚拟机的可执行文件**

java some.package.MyClass

java –jar my-packaged.jar

**jar:操作.jar文件：创建、更新、索引、列表、提取**

jar [选项] my.jar someDir/

**javadoc:从java源码中生成文档**

javadoc some.package

**jdeps:分析包或类的依赖**

jdeps com.me.MyClass

**jps:列出本地所有活动的JVM进程**

jps

**jstat:指定java进程的一些基本信息**

jstat <pid>

**jstatd:能让本地的JVM的信息通过网络传出去，通过RMI实现**

**jinfo:显示Java进程的系统属性和JVM选项**

jinfo <pid>

**jstack:输出进程中每个Java线程的堆栈跟着**

jstack <pid>

**jmap:查看Java进程的内存分配情况**

jmap <pid>

**javap:java类的反汇编程序，能查看类文件内容的工具**

javap <classname>

VisualVM:聚合上面的工具，并提供图形化界面。Java 6引入。