J2SE

基础

八种基本数据类型的大小，以及他们的封装类。

八种基本数据类型，int ,double ,long ,float, short,byte,character,boolean

对应的封装类型是：Integer ,Double ,Long ,Float, Short,Byte,Character,Boolean

Switch能否用string做参数？

在Java 5以前，switch(expr)中，expr只能是byte、short、char、int。从Java 5开始，Java中引入了枚举类型，expr也可以是enum类型，从Java 7开始，expr还可以是字符串（String），但是长整型（long）在目前所有的版本中都是不可以的。

equals与==的区别。

http://www.importnew.com/6804.html

> ==与equals的主要区别是：==常用于比较原生类型，而equals()方法用于检查对象的相等性。另一个不同的点是：如果==和equals()用于比较对象，当两个引用地址相同，==返回true。而equals()可以返回true或者false主要取决于重写实现。最常见的一个例子，字符串的比较，不同情况==和equals()返回不同的结果。equals()方法最重要的一点是，能够根据业务要求去重写，按照自定义规则去判断两个对象是否相等。重写equals()方法的时候，要注意一下hashCode是否会因为对象的属性改变而改变，否则在使用散列集合储存该对象的时候会碰到坑！！理解equals()方法的存在是很重要的。

1. 使用==比较有两种情况：

   ```
   比较基础数据类型(Java中基础数据类型包括八中：short,int,long,float,double,char,byte,boolen)：这种情况下，==比较的是他们的值是否相等。
   引用间的比较：在这种情况下，==比较的是他们在内存中的地址，也就是说，除非引用指向的是同一个new出来的对象，此时他们使用`==`去比较得到true，否则，得到false。
   ```

   1. 使用equals进行比较：

      equals追根溯源，是Object类中的一个方法，在该类中，equals的实现也仅仅只是比较两个对象的内存地址是否相等，但在一些子类中，如：String、Integer 等，该方法将被重写。

2. 以 `String` 类为例子说明 `eqauls` 与 `==` 的区别：

   > 在开始这个例子之前，同学们需要知道JVM处理String的一些特性。*Java的虚拟机在内存中开辟出一块单独的区域，用来存储字符串对象，这块内存区域被称为字符串缓冲池。*当使用 `String a = "abc"` 这样的语句进行定义一个引用的时候，首先会在*字符串缓冲池*中查找是否已经相同的对象，如果存在，那么就直接将这个对象的引用返回给a，如果不存在，则需要新建一个值为"abc"的对象，再将新的引用返回a。 `String a = new String("abc");` 这样的语句明确告诉JVM想要产生一个新的String对象，并且值为"abc"，于是就*在堆内存中的某一个小角落开辟了一个新的String对象*。

   - `==` 在比较引用的情况下，会去比较两个引用的内存地址是否相等。 ``` String str1 = "abc"; String str2 = "abc";

     System.out.println(str1 == str2); System.out.println(str1.equals(str2));

     String str2 = new String("abc"); System.out.println(str1 == str2); System.out.println(str1.equals(str2));

   `以上代码将会输出 true true false true **第一个true：**因为在str2赋值之前，str1的赋值操作就已经在内存中创建了一个值为"abc"的对象了，然后str2将会与str1指向相同的地址。 **第二个true：**因为public boolean equals(Object anObject) { //如果比较的对象与自身内存地址相等的话 //就说明他两指向的是同一个对象 //所以此时equals的返回值跟==的结果是一样的。 if (this == anObject) { return true; } //当比较的对象与自身的内存地址不相等，并且 //比较的对象是String类型的时候 //将会执行这个分支 if (anObject instanceof String) { String anotherString = (String)anObject; int n = value.length; if (n == anotherString.value.length) { char v1[] = value; char v2[] = anotherString.value; int i = 0; //在这里循环遍历两个String中的char while (n-- != 0) { //只要有一个不相等，那么就会返回false if (v1[i] != v2[i]) return false; i++; } return true; } } return false; } ``` 进行以上分析之后，就不难理解第一段代码中的实例程序输出了。

Object有哪些公用方法？

http://www.cnblogs.com/yumo/p/4908315.html

1．clone方法

保护方法，实现对象的浅复制，只有实现了Cloneable接口才可以调用该方法，否则抛出CloneNotSupportedException异常。

主要是JAVA里除了8种基本类型传参数是值传递，其他的类对象传参数都是引用传递，我们有时候不希望在方法里讲参数改变，这是就需要在类中复写clone方法。

2．getClass方法

final方法，获得运行时类型。

3．toString方法

该方法用得比较多，一般子类都有覆盖。

4．finalize方法

该方法用于释放资源。因为无法确定该方法什么时候被调用，很少使用。

5．equals方法

该方法是非常重要的一个方法。一般equals和==是不一样的，但是在Object中两者是一样的。子类一般都要重写这个方法。

6．hashCode方法

该方法用于哈希查找，可以减少在查找中使用equals的次数，重写了equals方法一般都要重写hashCode方法。这个方法在一些具有哈希功能的Collection中用到。

一般必须满足obj1.equals(obj2)==true。可以推出obj1.hashCode()==obj2.hashCode()，但是hashCode相等不一定就满足equals。不过为了提高效率，应该尽量使上面两个条件接近等价。

如果不重写hashCode(),在HashSet中添加两个equals的对象，会将两个对象都加入进去。

7．wait方法

wait方法就是使当前线程等待该对象的锁，当前线程必须是该对象的拥有者，也就是具有该对象的锁。wait()方法一直等待，直到获得锁或者被中断。wait(long timeout)设定一个超时间隔，如果在规定时间内没有获得锁就返回。

调用该方法后当前线程进入睡眠状态，直到以下事件发生。

（1）其他线程调用了该对象的notify方法。

（2）其他线程调用了该对象的notifyAll方法。

（3）其他线程调用了interrupt中断该线程。

（4）时间间隔到了。

此时该线程就可以被调度了，如果是被中断的话就抛出一个InterruptedException异常。

8．notify方法

该方法唤醒在该对象上等待的某个线程。

9．notifyAll方法

该方法唤醒在该对象上等待的所有线程。

Java的四种引用，强弱软虚，用到的场景。

JDK1.2之前只有强引用,其他几种引用都是在JDK1.2之后引入的.

- 强引用（Strong Reference） 最常用的引用类型，如Object obj = new Object(); 。只要强引用存在则GC时则必定不被回收。
- 软引用（Soft Reference） 用于描述还有用但非必须的对象，当堆将发生OOM（Out Of Memory）时则会回收软引用所指向的内存空间，若回收后依然空间不足才会抛出OOM。一般用于实现内存敏感的高速缓存。当真正对象被标记finalizable以及的finalize()方法调用之后并且内存已经清理, 那么如果SoftReference object还存在就被加入到它的 ReferenceQueue.只有前面几步完成后,Soft Reference和Weak Reference的get方法才会返回null
- 弱引用（Weak Reference） 发生GC时必定回收弱引用指向的内存空间。 和软引用加入队列的时机相同
- 虚引用（Phantom Reference) 又称为幽灵引用或幻影引用，虚引用既不会影响对象的生命周期，也无法通过虚引用来获取对象实例，仅用于在发生GC时接收一个系统通知。 当一个对象的finalize方法已经被调用了之后，这个对象的幽灵引用会被加入到队列中。通过检查该队列里面的内容就知道一个对象是不是已经准备要被回收了. 虚引用和软引用和弱引用都不同,它会在内存没有清理的时候被加入引用队列.虚引用的建立必须要传入引用队列,其他可以没有

Hashcode的作用。

http://c610367182.iteye.com/blog/1930676

以Java.lang.Object来理解,JVM每new一个Object,它都会将这个Object丢到一个Hash哈希表中去,这样的话,下次做Object的比较或者取这个对象的时候,它会根据对象的hashcode再从Hash表中取这个对象。这样做的目的是提高取对象的效率。具体过程是这样:

1. new Object(),JVM根据这个对象的Hashcode值,放入到对应的Hash表对应的Key上,如果不同的对象确产生了相同的hash值,也就是发生了Hash key相同导致冲突的情况,那么就在这个Hash key的地方产生一个链表,将所有产生相同hashcode的对象放到这个单链表上去,串在一起。
2. 比较两个对象的时候,首先根据他们的hashcode去hash表中找他的对象,当两个对象的hashcode相同,那么就是说他们这两个对象放在Hash表中的同一个key上,那么他们一定在这个key上的链表上。那么此时就只能根据Object的equal方法来比较这个对象是否equal。当两个对象的hashcode不同的话，肯定他们不能equal.

String、StringBuffer与StringBuilder的区别。

Java 平台提供了两种类型的字符串：String和StringBuffer / StringBuilder，它们可以储存和操作字符串。其中String是只读字符串，也就意味着String引用的字符串内容是不能被改变的。而StringBuffer和StringBulder类表示的字符串对象可以直接进行修改。StringBuilder是JDK1.5引入的，它和StringBuffer的方法完全相同，区别在于它是单线程环境下使用的，因为它的所有方面都没有被synchronized修饰，因此它的效率也比StringBuffer略高。

try catch finally，try里有return，finally还执行么？

会执行，在方法 返回调用者前执行。Java允许在finally中改变返回值的做法是不好的，因为如果存在finally代码块，try中的return语句不会立马返回调用者，而是纪录下返回值待finally代码块执行完毕之后再向调用者返回其值，然后如果在finally中修改了返回值，这会对程序造成很大的困扰，C#中就从语法规定不能做这样的事。

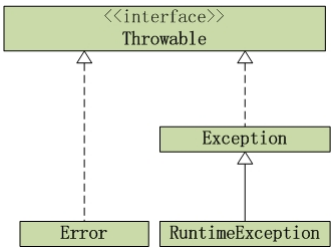
Excption与Error区别

Error表示系统级的错误和程序不必处理的异常，是恢复不是不可能但很困难的情况下的一种严重问题；比如内存溢出，不可能指望程序能处理这样的状况；Exception表示需要捕捉或者需要程序进行处理的异常，是一种设计或实现问题；也就是说，它表示如果程序运行正常，从不会发生的情况。

Excption与Error包结构。OOM你遇到过哪些情况，SOF你遇到过哪些情况。

http://www.cnblogs.com/yumo/p/4909617.html

Java异常架构图



1. Throwable Throwable是 Java 语言中所有错误或异常的超类。 Throwable包含两个子类: Error 和 Exception 。它们通常用于指示发生了异常情况。 Throwable包含了其线程创建时线程执行堆栈的快照，它提供了printStackTrace()等接口用于获取堆栈跟踪数据等信息。
2. Exception Exception及其子类是 Throwable 的一种形式，它指出了合理的应用程序想要捕获的条件。
3. RuntimeException RuntimeException是那些可能在 Java 虚拟机正常运行期间抛出的异常的超类。 编译器不会检查RuntimeException异常。 例如，除数为零时，抛出ArithmeticException异常。RuntimeException是ArithmeticException的超类。当代码发生除数为零的情况时，倘若既"没有通过throws声明抛出ArithmeticException异常"，也"没有通过try...catch...处理该异常"，也能通过编译。这就是我们所说的"编译器不会检查RuntimeException异常"！ 如果代码会产生RuntimeException异常，则需要通过修改代码进行避免。 例如，若会发生除数为零的情况，则需要通过代码避免该情况的发生！
4. Error 和Exception一样， Error也是Throwable的子类。 它用于指示合理的应用程序不应该试图捕获的严重问题，大多数这样的错误都是异常条件。 和RuntimeException一样， 编译器也不会检查Error。

Java将可抛出(Throwable)的结构分为三种类型： 被检查的异常(Checked Exception)，运行时异常(RuntimeException)和错误(Error)。

(01) 运行时异常 定义 : RuntimeException及其子类都被称为运行时异常。 特点 : Java编译器不会检查它。 也就是说，当程序中可能出现这类异常时，倘若既"没有通过throws声明抛出它"，也"没有用try-catch语句捕获它"，还是会编译通过。例如，除数为零时产生的ArithmeticException异常，数组越界时产生的IndexOutOfBoundsException异常，fail-fail机制产生的ConcurrentModificationException异常等，都属于运行时异常。 虽然Java编译器不会检查运行时异常，但是我们也可以通过throws进行声明抛出，也可以通过try-catch对它进行捕获处理。 如果产生运行时异常，则需要通过修改代码来进行避免。 例如，若会发生除数为零的情况，则需要通过代码避免该情况的发生！

(02) 被检查的异常 定义 : Exception类本身，以及Exception的子类中除了"运行时异常"之外的其它子类都属于被检查异常。 特点 : Java编译器会检查它。 此类异常，要么通过throws进行声明抛出，要么通过try-catch进行捕获处理，否则不能通过编译。例如，CloneNotSupportedException就属于被检查异常。当通过clone()接口去克隆一个对象，而该对象对应的类没有实现Cloneable接口，就会抛出CloneNotSupportedException异常。 被检查异常通常都是可以恢复的。

(03) 错误 定义 : Error类及其子类。 特点 : 和运行时异常一样，编译器也不会对错误进行检查。 当资源不足、约束失败、或是其它程序无法继续运行的条件发生时，就产生错误。程序本身无法修复这些错误的。例如，VirtualMachineError就属于错误。 按照Java惯例，我们是不应该是实现任何新的Error子类的！

对于上面的3种结构，我们在抛出异常或错误时，到底该哪一种？《Effective Java》中给出的建议是： 对于可以恢复的条件使用被检查异常，对于程序错误使用运行时异常。

OOM：

1. OutOfMemoryError异常

除了程序计数器外，虚拟机内存的其他几个运行时区域都有发生OutOfMemoryError(OOM)异常的可能，

Java Heap 溢出

一般的异常信息：java.lang.OutOfMemoryError:Java heap spacess

java堆用于存储对象实例，我们只要不断的创建对象，并且保证GC Roots到对象之间有可达路径来避免垃圾回收机制清除这些对象，就会在对象数量达到最大堆容量限制后产生内存溢出异常。

出现这种异常，一般手段是先通过内存映像分析工具(如Eclipse Memory Analyzer)对dump出来的堆转存快照进行分析，重点是确认内存中的对象是否是必要的，先分清是因为内存泄漏(Memory Leak)还是内存溢出(Memory Overflow)。

如果是内存泄漏，可进一步通过工具查看泄漏对象到GC Roots的引用链。于是就能找到泄漏对象时通过怎样的路径与GC Roots相关联并导致垃圾收集器无法自动回收。

如果不存在泄漏，那就应该检查虚拟机的参数(-Xmx与-Xms)的设置是否适当。

2. 虚拟机栈和本地方法栈溢出

   如果线程请求的栈深度大于虚拟机所允许的最大深度，将抛出StackOverflowError异常。

   如果虚拟机在扩展栈时无法申请到足够的内存空间，则抛出OutOfMemoryError异常

   这里需要注意当栈的大小越大可分配的线程数就越少。

3. 运行时常量池溢出

   异常信息：java.lang.OutOfMemoryError:PermGen space

   如果要向运行时常量池中添加内容，最简单的做法就是使用String.intern()这个Native方法。该方法的作用是：如果池中已经包含一个等于此String的字符串，则返回代表池中这个字符串的String对象；否则，将此String对象包含的字符串添加到常量池中，并且返回此String对象的引用。由于常量池分配在方法区内，我们可以通过-XX:PermSize和-XX:MaxPermSize限制方法区的大小，从而间接限制其中常量池的容量。

4. 方法区溢出

   方法区用于存放Class的相关信息，如类名、访问修饰符、常量池、字段描述、方法描述等。

   异常信息：java.lang.OutOfMemoryError:PermGen space

   方法区溢出也是一种常见的内存溢出异常，一个类如果要被垃圾收集器回收，判定条件是很苛刻的。在经常动态生成大量Class的应用中，要特别注意这点。

Java面向对象的三个特征与含义。

继承：继承是从已有类得到继承信息创建新类的过程。提供继承信息的类被称为父类（超类、基类）；得到继承信息的类被称为子类（派生类）。继承让变化中的软件系统有了一定的延续性，同时继承也是封装程序中可变因素的重要手段。

封装：通常认为封装是把数据和操作数据的方法绑定起来，对数据的访问只能通过已定义的接口。面向对象的本质就是将现实世界描绘成一系列完全自治、封闭的对象。我们在类中编写的方法就是对实现细节的一种封装；我们编写一个类就是对数据和数据操作的封装。可以说，封装就是隐藏一切可隐藏的东西，只向外界提供最简单的编程接口（可以想想普通洗衣机和全自动洗衣机的差别，明显全自动洗衣机封装更好因此操作起来更简单；我们现在使用的智能手机也是封装得足够好的，因为几个按键就搞定了所有的事情）。

多态：多态性是指允许不同子类型的对象对同一消息作出不同的响应。简单的说就是用同样的对象引用调用同样的方法但是做了不同的事情。多态性分为编译时的多态性和运行时的多态性。如果将对象的方法视为对象向外界提供的服务，那么运行时的多态性可以解释为：当A系统访问B系统提供的服务时，B系统有多种提供服务的方式，但一切对A系统来说都是透明的（就像电动剃须刀是A系统，它的供电系统是B系统，B系统可以使用电池供电或者用交流电，甚至还有可能是太阳能，A系统只会通过B类对象调用供电的方法，但并不知道供电系统的底层实现是什么，究竟通过何种方式获得了动力）。方法重载（overload）实现的是编译时的多态性（也称为前绑定），而方法重写（override）实现的是运行时的多态性（也称为后绑定）。运行时的多态是面向对象最精髓的东西，要实现多态需要做两件事：1. 方法重写（子类继承父类并重写父类中已有的或抽象的方法）；2. 对象造型（用父类型引用引用子类型对象，这样同样的引用调用同样的方法就会根据子类对象的不同而表现出不同的行为）。

Override和Overload的含义与区别。

Overload：顾名思义，就是Over(重新)——load（加载），所以中文名称是重载。它可以表现类的多态性，可以是函数里面可以有相同的函数名但是参数名、类型不能相同；或者说可以改变参数、类型但是函数名字依然不变。

Override：就是ride(重写)的意思，在子类继承父类的时候子类中可以定义某方法与其父类有相同的名称和参数，当子类在调用这一函数时自动调用子类的方法，而父类相当于被覆盖（重写）了。

方法的重写Overriding和重载Overloading是Java多态性的不同表现。重写Overriding是父类与子类之间多态性的一种表现，重载Overloading是一个类中多态性的一种表现。如果在子类中定义某方法与其父类有相同的名称和参数，我们说该方法被重写 (Overriding)。子类的对象使用这个方法时，将调用子类中的定义，对它而言，父类中的定义如同被“屏蔽”了。如果在一个类中定义了多个同名的方法，它们或有不同的参数个数或有不同的参数类型，则称为方法的重载(Overloading)。Overloaded的方法是可以改变返回值的类型。

Interface与abstract类的区别。

抽象类和接口都不能够实例化，但可以定义抽象类和接口类型的引用。一个类如果继承了某个抽象类或者实现了某个接口都需要对其中的抽象方法全部进行实现，否则该类仍然需要被声明为抽象类。接口比抽象类更加抽象，因为抽象类中可以定义构造器，可以有抽象方法和具体方法，而接口中不能定义构造器而且其中的方法全部都是抽象方法。抽象类中的成员可以是private、默认、protected、public的，而接口中的成员全都是public的。抽象类中可以定义成员变量，而接口中定义的成员变量实际上都是常量。有抽象方法的类必须被声明为抽象类，而抽象类未必要有抽象方法。

Static class 与non static class的区别。

内部静态类不需要有指向外部类的引用。但非静态内部类需要持有对外部类的引用。非静态内部类能够访问外部类的静态和非静态成员。静态类不能访问外部类的非静态成员。他只能访问外部类的静态成员。一个非静态内部类不能脱离外部类实体被创建，一个非静态内部类可以访问外部类的数据和方法，因为他就在外部类里面。

java多态的实现原理。

http://blog.csdn.net/zzzhangzhun/article/details/51095075

当JVM执行Java字节码时，类型信息会存储在方法区中，为了优化对象的调用方法的速度，方法区的类型信息会增加一个指针，该指针指向一个记录该类方法的方法表，方法表中的每一个项都是对应方法的指针。

方法区：方法区和JAVA堆一样，是各个线程共享的内存区域，用于存储已被虚拟机加载的类信息、常量、静态变量、即时编译器编译后的代码等数据。 运行时常量池：它是方法区的一部分，Class文件中除了有类的版本、方法、字段等描述信息外，还有一项信息是常量池，用于存放编译器生成的各种符号引用，这部分信息在类加载时进入方法区的运行时常量池中。 方法区的内存回收目标是针对常量池的回收及对类型的卸载。

方法表的构造

由于java的单继承机制，一个类只能继承一个父类，而所有的类又都继承Object类，方法表中最先存放的是Object的方法，接下来是父类的方法，最后是该类本身的方法。如果子类改写了父类的方法，那么子类和父类的那些同名的方法共享一个方法表项。

由于这样的特性，使得方法表的偏移量总是固定的，例如，对于任何类来说，其方法表的equals方法的偏移量总是一个定值，所有继承父类的子类的方法表中，其父类所定义的方法的偏移量也总是一个定值。

实例

假设Class A是Class B的子类，并且A改写了B的方法的method()，那么B来说，method方法的指针指向B的method方法入口；对于A来说，A的方法表的method项指向自身的method而非父类的。

流程：调用方法时，虚拟机通过对象引用得到方法区中类型信息的方法表的指针入口，查询类的方法表 ，根据实例方法的符号引用解析出该方法在方法表的偏移量，子类对象声明为父类类型时，形式上调用的是父类的方法，此时虚拟机会从实际的方法表中找到方法地址，从而定位到实际类的方法。 注：所有引用为父类，但方法区的类型信息中存放的是子类的信息，所以调用的是子类的方法表。

foreach与正常for循环效率对比。

http://904510742.iteye.com/blog/2118331

直接for循环效率最高，其次是迭代器和 ForEach操作。 作为语法糖，其实 ForEach 编译成 字节码之后，使用的是迭代器实现的，反编译后，testForEach方法如下：

```
public static void testForEach(List list) {
    for (Iterator iterator = list.iterator(); iterator.hasNext();) {
        Object t = iterator.next();
        Object obj = t;
    }
}
```

可以看到，只比迭代器遍历多了生成中间变量这一步，因为性能也略微下降了一些。

反射机制

JAVA反射机制是在运行状态中, 对于任意一个类, 都能够知道这个类的所有属性和方法; 对于任意一个对象, 都能够调用它的任意一个方法和属性; 这种动态获取的信息以及动态调用对象的方法的功能称为java语言的反射机制.

主要作用有三：

运行时取得类的方法和字段的相关信息。

创建某个类的新实例(.newInstance())

取得字段引用直接获取和设置对象字段，无论访问修饰符是什么。

用处如下：

观察或操作应用程序的运行时行为。

调试或测试程序，因为可以直接访问方法、构造函数和成员字段。

通过名字调用不知道的方法并使用该信息来创建对象和调用方法。

String类内部实现，能否改变String对象内容

String源码分析

http://blog.csdn.net/zhangjg_blog/article/details/18319521

try catch 块，try里有return，finally也有return，如何执行

http://qing0991.blog.51cto.com/1640542/1387200

泛型的优缺点

优点：

使用泛型类型可以最大限度地重用代码、保护类型的安全以及提高性能。

泛型最常见的用途是创建集合类。

缺点：

在性能上不如数组快。

泛型常用特点，List `<String>` 能否转为List `<Object>`

能，但是利用类都继承自Object，所以使用是每次调用里面的函数都要通过强制转换还原回原来的类，这样既不安全，运行速度也慢。

解析XML的几种方式的原理与特点：DOM、SAX、PULL。

http://www.cnblogs.com/HaroldTihan/p/4316397.html

Java与C++对比。

http://developer.51cto.com/art/201106/270422.htm

Java1.7与1.8新特性。

http://blog.chinaunix.net/uid-29618857-id-4416835.html

JNI的使用。

http://landerlyoung.github.io/blog/2014/10/16/java-zhong-jnide-shi-yong/

集合

ArrayList、LinkedList、Vector的底层实现和区别

- 从同步性来看，ArrayList和LinkedList是不同步的，而Vector是的。所以线程安全的话，可以使用ArrayList或LinkedList，可以节省为同步而耗费的开销。但在多线程下，有时候就不得不使用Vector了。当然，也可以通过一些办法包装ArrayList、LinkedList，使我们也达到同步，但效率可能会有所降低。
- 从内部实现机制来讲ArrayList和Vector都是使用Object的数组形式来存储的。当你向这两种类型中增加元素的时候，如果元素的数目超出了内部数组目前的长度它们都需要扩展内部数组的长度，Vector缺省情况下自动增长原来一倍的数组长度，ArrayList是原来的50%，所以最后你获得的这个集合所占的空间总是比你实际需要的要大。如果你要在集合中保存大量的数据，那么使用Vector有一些优势，因为你可以通过设置集合的初始化大小来避免不必要的资源开销。
- ArrayList和Vector中，从指定的位置（用index）检索一个对象，或在集合的末尾插入、删除一个对象的时间是一样的，可表示为O(1)。但是，如果在集合的其他位置增加或者删除元素那么花费的时间会呈线性增长O(n-i)，其中n代表集合中元素的个数，i代表元素增加或移除元素的索引位置，因为在进行上述操作的时候集合中第i和第i个元素之后的所有元素都要执行(n-i)个对象的位移操作。LinkedList底层是由双向循环链表实现的，LinkedList在插入、删除集合中任何位置的元素所花费的时间都是一样的O(1)，但它在索引一个元素的时候比较慢，为O(i)，其中i是索引的位置，如果只是查找特定位置的元素或只在集合的末端增加、移除元素，那么使用Vector或ArrayList都可以。如果是对其它指定位置的插入、删除操作，最好选择LinkedList。

HashMap和HashTable的底层实现和区别，两者和ConcurrentHashMap的区别。

http://blog.csdn.net/xuefeng0707/article/details/40834595

HashTable线程安全则是依靠方法简单粗暴的sychronized修饰，HashMap则没有相关的线程安全问题考虑。。

在以前的版本貌似ConcurrentHashMap引入了一个“分段锁”的概念，具体可以理解为把一个大的Map拆分成N个小的HashTable，根据key.hashCode()来决定把key放到哪个HashTable中。在ConcurrentHashMap中，就是把Map分成了N个Segment，put和get的时候，都是现根据key.hashCode()算出放到哪个Segment中。

通过把整个Map分为N个Segment（类似HashTable），可以提供相同的线程安全，但是效率提升N倍。

HashMap的hashcode的作用？什么时候需要重写？如何解决哈希冲突？查找的时候流程是如何？

从源码分析HashMap

Arraylist和HashMap如何扩容？负载因子有什么作用？如何保证读写进程安全？

http://m.blog.csdn.net/article/details?id=48956087

http://hovertree.com/h/bjaf/2jdr60li.htm

ArrayList 本身不是线程安全的。 所以正确的做法是去用 java.util.concurrent 里的 CopyOnWriteArrayList 或者某个同步的 Queue 类。

HashMap实现不是同步的。如果多个线程同时访问一个哈希映射，而其中至少一个线程从结构上修改了该映射，则它必须 保持外部同步。（结构上的修改是指添加或删除一个或多个映射关系的任何操作；仅改变与实例已经包含的键关联的值不是结构上的修改。）这一般通过对自然封装该映射的对象进行同步操作来完成。如果不存在这样的对象，则应该使用 Collections.synchronizedMap 方法来“包装”该映射。最好在创建时完成这一操作，以防止对映射进行意外的非同步访问.

TreeMap、HashMap、LinkedHashMap的底层实现区别。

http://blog.csdn.net/lolashe/article/details/20806319

Collection包结构，与Collections的区别。

Collection是一个接口，它是Set、List等容器的父接口；Collections是一个工具类，提供了一系列的静态方法来辅助容器操作，这些方法包括对容器的搜索、排序、线程安全化等等。

Set、List之间的区别是什么？

http://developer.51cto.com/art/201309/410205_all.htm

Map、Set、List、Queue、Stack的特点与用法。

http://www.cnblogs.com/yumo/p/4908718.html

Collection 是对象集合， Collection 有两个子接口 List 和 Set

List 可以通过下标 (1,2..) 来取得值，值可以重复

而 Set 只能通过游标来取值，并且值是不能重复的

ArrayList ， Vector ， LinkedList 是 List 的实现类

ArrayList 是线程不安全的， Vector 是线程安全的，这两个类底层都是由数组实现的

LinkedList 是线程不安全的，底层是由链表实现的

Map 是键值对集合

HashTable 和 HashMap 是 Map 的实现类
HashTable 是线程安全的，不能存储 null 值
HashMap 不是线程安全的，可以存储 null 值

Stack类：继承自Vector，实现一个后进先出的栈。提供了几个基本方法，push、pop、peak、empty、search等。

Queue接口：提供了几个基本方法，offer、poll、peek等。已知实现类有LinkedList、PriorityQueue等。