**多线程、并发及线程**

一、多线程：

1. 状态及转换过程
2. 同步synchronized，lock
3. 同步Notify，notifyAll，wait
4. yeild，sleep，interrupt，join
5. Volatile（可见可排列），原子类，java内存模型
6. 线程安全的map，blockingqueue，单例
7. 不可变对象
8. Countdownlatch，CyclicBarrier，Semaphore等相关同步方法
9. 管理类threadpoolexecutor—executorservice
10. Concurrent包
11. Threadlocal

**与线程同步的区别：**

对于多线程资源共享的问题，同步机制采用了“以时间换空间”的方式：访问串行化，对象共享化。而ThreadLocal采用了“以空间换时间”的方式：访问并行化，对象独享化。前者仅提供一份变量，让不同的线程排队访问，而后者为每一个线程都提供了一份变量，因此可以同时访问而互不影响。

**使用场景：**

数据库连接管理

Session管理

1. exector包

**数据类型和 Java 基础**

1. java基础
2. 基本数据类型及封装类：

(byte，short，char)--int--long--float—double 相互转换

1. 数组工具类Arrays和集合工具类Collections

二、引用类型：

类 接口 数组 枚举 标注

JVM的内存空间：（1）. Heap 堆空间：分配对象 new Student（）

（2）. Stack 栈空间：临时变量 Student stu

（3）.Code 代码区 ：类的定义，静态资源 Student.class

三、cloneable，serializable

1.序列化只能保存对象的非静态成员交量，不能保存任何的成员方法和静态的成员变量

2.transient关键字的作用是：阻止实例中那些用此关键字声明的变量持久化；当对象被反序列化时（从源文件读取字节序列进行重构），这样的实例变量值不会被持久化和恢复。

3. 实现对象克隆有两种方式：

  1). 实现Cloneable接口并重写Object类中的clone()方法；

  2). 实现Serializable接口，通过对象的序列化和反序列化实现克隆，可以实现真正的深度克隆。

四、hashcode与equals

1、HashCode的存在主要是为了查找的快捷性，HashCode是用来在散列存储结构中确定对象的存储地址的

2、如果两个对象equals相等，那么这两个对象的HashCode一定也相同

3、如果对象的equals方法被重写，那么对象的HashCode方法也尽量重写

4、如果两个对象的HashCode相同，不代表两个对象就相同，只能说明这两个对象在散列存储结构中，存放于同一个位置

五、类间关系

* 1. 依赖关系：耦合最小
  2. 关联关系
  3. 聚合关系
  4. 组合关系
  5. 泛化关系

六、内部类

1. 成员内部类

即作为外部类的一个成员存在，与外部类的属性、方法并列。

注意：成员内部类中不能定义静态变量,但可以访问外部类的所有成员。

成员内部类的优点：  
⑴ 内部类作为外部类的成员，可以访问外部类的私有成员或属性。（即使将外部类声明为PRIVATE，但是对于处于其内部的内部类还是可见的。）  
⑵ 用内部类定义在外部类中不可访问的属性。这样就在外部类中实现了比外部类的private还要小的访问权限。  
注意：内部类是一个编译时的概念，一旦编译成功，就会成为完全不同的两类。对于一个名为outer的外部类和其内部定义的名为inner的内部类。编译完成后出现outer.class和outer$inner.class两类。

1. 局部内部类

即在方法中定义的内部类，与局部变量类似，在局部内部类前不加修饰符public或private，其范围为定义它的代码块。

注意：局部内部类中不可定义静态变量，可以访问外部类的局部变量(即方法内的变量)，但是变量必须是final的。

注意：  
在类外不可直接生成局部内部类（保证局部内部类对外是不可见的）。要想使用局部内部类时需要生成对象，对象调用方法，在方法中才能调用其局部内部类。通过内部类和接口达到一个强制的弱耦合，用局部内部类来实现接口，并在方法中返回接口类型，使局部内部类不可见，屏蔽实现类的可见性。

1. 静态内部类

静态内部类定义在类中，任何方法外，用static定义。

注意：静态内部类中可以定义静态或者非静态的成员

注意：\*\*\*\*\*\*\*生成（new）一个静态内部类不需要外部类成员：这是静态内部类和成员内部类的区别。静态内部类的对象可以直接生成：  
Outer.Inner in=new Outer.Inner()；  
而不需要通过生成外部类对象来生成。这样实际上使静态内部类成为了一个顶级类。静态内部类不可用private来进行定义。\*\*\*\*\*\*\*

1. 匿名内部类

匿名内部类是一种特殊的局部内部类，它是通过匿名类实现接口。

匿名内部类的特点：  
1，一个类用于继承其他类或是实现接口，并不需要增加额外的方法，只是对继承方法的事先或是覆盖。  
2，只是为了获得一个对象实例，不需要知道其实际类型。  
3，类名没有意义，也就是不需要使用到。

注：一个匿名内部类一定是在new的后面，用其隐含实现一个接口或实现一个类，没有类名，根据多态，我们使用其父类名。因他是局部内部类，那么局部内部类的所有限制都对其生效。匿名内部类是唯一一种无构造方法类。大部分匿名内部类是用于接口回调用的。匿名内部类在编译的时候由系统自动起名Out$1.class。如果一个对象编译时的类型是接口，那么其运行的类型为实现这个接口的类。因匿名内部类无构造方法，所以其使用范围非常的有限。当需要多个对象时使用局部内部类，因此局部内部类的应用相对比较多。匿名内部类中不能定义构造方法。如果一个对象编译时的类型是接口，那么其运行的类型为实现这个接口的类。

**JVM 底层 与 GC（Garbage Collection）**

1. Java GC机制主要完成3件事：确定哪些内存需要回收，确定什么时候需要执行GC，如何执行GC。
2. 虚拟机栈（JVM Stack）：局部变量表、操作站、动态链接、方法出口等，局部变量表中存储着方法的相关局部变量，包括各种基本数据类型，对象的引用，返回地址等。
3. 堆区（Heap）：堆区由所有线程共享，在虚拟机启动时创建。堆区的存在是为了存储对象实例。
4. 方法区（Method Area）：存储已经被虚拟机加载的类信息（即加载类时需要加载的信息，包括版本、field、方法、接口等信息）、final常量、静态变量、编译器即时编译的代码等。一般的，方法区上执行的垃圾收集是很少的。
5. reference类型引用访问具体对象的方式：通过句柄访问，通过直接指针访问。
6. java内存分配机制

Java内存分配和回收的机制概括的说，就是：分代分配，分代回收。对象将根据存活的时间被分为：年轻代（Young Generation）、年老代（Old Generation）、永久代（Permanent Generation，也就是方法区）。

绝大多数刚创建的对象会被分配在Eden区，其中的大多数对象很快就会消亡。Eden区是连续的内存空间，因此在其上分配内存极快；

最初一次，当Eden区满的时候，执行Minor GC，将消亡的对象清理掉，并将剩余的对象复制到一个存活区Survivor0（此时，Survivor1是空白的，两个Survivor总有一个是空白的）；

下次Eden区满了，再执行一次Minor GC，将消亡的对象清理掉，将存活的对象复制到Survivor1中，然后清空Eden区；

将Survivor0中消亡的对象清理掉，将其中可以晋级的对象晋级到Old区，将存活的对象也复制到Survivor1区，然后清空Survivor0区；

当两个存活区切换了几次（HotSpot虚拟机默认15次，用-XX:MaxTenuringThreshold控制，大于该值进入老年代，但这只是个最大值，并不代表一定是这个值）之后，仍然存活的对象（其实只有一小部分，比如，我们自己定义的对象），将被复制到老年代。

从上面的过程可以看出，Eden区是连续的空间，且Survivor总有一个为空。经过一次GC和复制，一个Survivor中保存着当前还活着的对象，而Eden区和另一个Survivor区的内容都不再需要了，可以直接清空，到下一次GC时，两个Survivor的角色再互换。因此，这种方式分配内存和清理内存的效率都极高，这种垃圾回收的方式就是著名的“停止-复制（Stop-and-copy）”清理法（将Eden区和一个Survivor中仍然存活的对象拷贝到另一个Survivor中）

年老代（Old Generation）：对象如果在年轻代存活了足够长的时间而没有被清理掉（即在几次Young GC后存活了下来），则会被复制到年老代，年老代的空间一般比年轻代大，能存放更多的对象，在年老代上发生的GC次数也比年轻代少。当年老代内存不足时，将执行Major GC，也叫 Full GC。

1. Java GC机制

GC机制的基本算法是：分代收集。

年轻代：“停止-复制”算法，用-XX:SurvivorRatio参数来配置Eden区域Survivor区的容量比值，默认是8，代表Eden：Survivor1：Survivor2=8:1:1.

老年代：标记-整理算法，即：标记出仍然存活的对象（存在引用的），将所有存活的对象向一端移动，以保证内存的连续。在发生Minor GC时，虚拟机会检查每次晋升进入老年代的大小是否大于老年代的剩余空间大小，如果大于，则直接触发一次Full GC，否则，就查看是否设置了-XX:+HandlePromotionFailure（允许担保失败），如果允许，则只会进行MinorGC，此时可以容忍内存分配失败；如果不允许，则仍然进行Full GC（这代表着如果设置-XX:+Handle PromotionFailure，则触发MinorGC就会同时触发Full GC，哪怕老年代还有很多内存，所以，最好不要这样做）。

方法区（永久代）：永久代的回收有两种：常量池中的常量，无用的类信息，常量的回收很简单，没有引用了就可以被回收。对于无用的类进行回收，必须保证3点：

类的所有实例都已经被回收

加载类的ClassLoader已经被回收

类对象的Class对象没有被引用（即没有通过反射引用该类的地方）

永久代的回收并不是必须的，可以通过参数来设置是否对类进行回收。HotSpot提供-Xnoclassgc进行控制使用-verbose，-XX:+TraceClassLoading、-XX:+TraceClassUnLoading可以查看类加载和卸载信息-verbose、-XX:+TraceClassLoading可以在Product版HotSpot中使用； -XX:+TraceClassUnLoading需要fastdebug版HotSpot支持

1. 垃圾收集器

在GC机制中，起重要作用的是垃圾收集器，垃圾收集器是GC的具体实现。

总结：java GC是在什么时候，对什么东西，做了什么事情？

回答：

程序员不能具体控制时间，系统在不可预测的时间调用System.gc()函数的时候；当然可以通过调优，用NewRatio控制newObject和oldObject的比例，用MaxTenuringThreshold 控制进入oldObject的次数，使得oldObject 存储空间延迟达到full gc,从而使得计时器引发gc时间延迟OOM的时间延迟，以延长对象生存期。

超出了作用域或引用计数为空的对象；从gc root开始搜索找不到的对象，而且经过一次标记、清理，仍然没有复活的对象。

删除不使用的对象，回收内存空间；运行默认的finalize,当然程序员想立刻调用就用dipose调用以释放资源如文件句柄，JVM用from survivor、to survivor对它进行标记清理，对象序列化后也可以使它复活。

1. 作用域

**内存泄漏**

**NIO**

**http2**

https://www.cnblogs.com/ranyonsue/p/5984001.html

**数据库**

1. 索引
2. 主键（聚集索引）

数据结构：「平衡树」(非二叉)，也就是b tree或者 b+ tree

一个表只能有一个主键， 一个表只能有一个「聚集索引」，因为主键的作用就是把「表」的数据格式转换成「索引（平衡树）」的格式放置

时间复杂度：O(log n)，n是记录总树，底数是树的分叉数，结果就是树的层次数。

索引能让数据库查询数据的速度上升， 而使写入数据的速度下降，原因很简单的， 因为平衡树这个结构必须一直维持在一个正确的状态， 增删改数据都会改变平衡树各节点中的索引数据内容，破坏树结构， 因此，在每次数据改变时， DBMS必须去重新梳理树（索引）的结构以确保它的正确。

1. 非聚集索引

非聚集索引和聚集索引的区别在于， 通过聚集索引可以查到需要查找的数据， 而通过非聚集索引可以查到记录对应的主键值 ， 再使用主键的值通过聚集索引查找到需要的数据；不管以任何方式查询表， 最终都会利用主键通过聚集索引来定位到数据， 聚集索引（主键）是通往真实数据所在的唯一路径。

例外：「覆盖索引」（复合索引/多字段索引查询）

create index index\_birthday\_and\_user\_name on user\_info(birthday, user\_name);

查询时使用联合索引的一个字段，如果这个字段在联合索引中所有字段的第一个，那就会用到索引，否则就无法使用到索引。

select user\_name from user\_info where birthday = '1991-11-1';

一般来说mysql会选择其中一个索引， mysq会统计每个索引上的重复度，选用低重复度的字段。

什么情况下考虑联合索引？

1. 唯一性索引（unique index）
2. 数据库原理和常用性能优化技术
3. 选取最适用的字段属性
4. 使用连接（JOIN）来代替子查询(Sub-Queries)
5. 使用联合(UNION)来代替手动创建的临时表
6. 事务

事务的另一个重要作用是当多个用户同时使用相同的数据源时，它可以利用锁定数据库的方法来为用户提供一种安全的访问方式，这样可以保证用户的操作不被其它的用户所干扰。

1. 锁定表
2. 使用外键

锁定表的方法可以维护数据的完整性，但是它却不能保证数据的关联性。这个时候我们就可以使用外键。

1. 使用索引
2. 优化查询语句

首先，最好是在相同类型的字段间进行比较的操作。

其次，在建有索引的字段上尽量不要使用函数进行操作。

第三，在搜索字符型字段时，我们有时会使用LIKE关键字和通配符，这种做法虽然简单，但却也是以牺牲系统性能为代价的。

最后，应该注意避免在查询中让MySQL进行自动类型转换，因为转换过程也会使索引变得不起作用。

1. NoSQL、Queue的原理、使用场景及限制

Redis，mogodb

1. 事务
2. ACID
3. 并行问题
4. 脏读：指一个线程中的事务读取到了另外一个线程中未提交的数据。
5. 不可重复读（虚读）：指一个线程中的事务读取到了另外一个线程中提交的update的数据。
6. 幻读：指一个线程中的事务读取到了另外一个线程中提交的insert(delete)的数据。
7. 隔离级别

1）**Read uncommitted未提交读，所有并发问题都会发生**

2）**Read committed已提交读，可以解决脏读**

3）**Repeatable read 可重复读，可以解决脏读，不可重复读**

4）**Serialization 可串行读，可以解决并发事务所有问题**

1. MYSQL

**算法以及优化**

1. 排序
2. 分治

https://www.cnblogs.com/leishitou/p/5436201.html

1. 特征

1) 该问题的规模缩小到一定的程度就可以容易地解决

2) 该问题可以分解为若干个规模较小的相同问题，即该问题具有最优子结构性质。

3) 利用该问题分解出的子问题的解可以合并为该问题的解；

4) 该问题所分解出的各个子问题是相互独立的，即子问题之间不包含公共的子子问题。

总结：第一条特征是绝大多数问题都可以满足的，因为问题的计算复杂性一般是随着问题规模的增加而增加；

第二条特征是应用分治法的前提它也是大多数问题可以满足的，此特征反映了递归思想的应用；

第三条特征是关键，能否利用分治法完全取决于问题是否具有第三条特征，如果具备了第一条和第二条特征，而不具备第三条特征，则可以考虑用贪心法或动态规划法。

第四条特征涉及到分治法的效率，如果各子问题是不独立的则分治法要做许多不必要的工作，重复地解公共的子问题，此时虽然可用分治法，但一般用动态规划法较好。

1. 综合起来概括，在问题规模与计算量成正比的算法中，分治算法不是最好的解法，并且有可能是效率极其底下的算法。如果存在某个问题，线性增长的问题规模可能带动计算量的非线性增长，并且符合分治算法的三个特征，那么分治算法是一个很不错的选择。
2. 实例
3. 动态规划

https://blog.csdn.net/baidu\_28312631/article/details/47418773

1. 特征
2. **具有相同的子问题**：我们必须保证我们分割成的子问题也能按照相同的方法分割成更小的自问题， 并这些自问题的最终分割情况是可以解决的。
3. **满足最优子结构**：就是一个决策的子决策也是最优的
4. **无后效性**：这是DP中最重要的一点， 他要求每个子问题的决策不能对后面其他未解决的问题产影响， 如果产生就无法保证决策的最优性， 这就是无后效性。往往需要我们找到一个合适的状态。
5. 基本思路

第一步：**确定子问题**。 在这一步重点是分析那些**变量是随着问题规模的变小而变小的**， 那些变量与问题的规模无关。   
第二步：**确定状态**：根据上面找到的子问题来给你分割的**子问题限定状态**   
第三步：**推到出状态转移方程**：这里要注意你的状态转移方程是不是满足所有的条件， 注意不要遗漏。   
第四步：**确定边界条件**：先根据题目的限制条件来确定题目中给出的边界条件是否能直接推导出， 如果不行也可以尝试从边界条件反推（举个例子：a(n)→a(2)有递推关系， 但是a(2)→a(1)不符合上述递推关系， 我们就可以考虑用a(1)来倒推出a(2)， 然后将递推的终点设置为a(2)）;   
第五步：**确定实现方式**：这个依照个人习惯 就像是01背包的两层for循环的顺序   
第六步：**确定优化方法**：很多时候你会发现走到这里步的时候你需要返回第1步重来。首先考虑降维问题（优化内存）， 优先队列、四边形不等式（优化时间）等等。

1. 实例

https://blog.csdn.net/u010398493/article/details/52809957

1. 贪心
2. 基本思路
   1. 建立数学模型来描述问题。
   2. 把求解的问题分成若干个子问题。
   3. 对每一子问题求解，得到子问题的局部最优解。
   4. 把子问题的解局部最优解合成原来解问题的一个解。
3. 实现框架

从问题的某一初始解出发；

while （能朝给定总目标前进一步）{

利用可行的决策，求出可行解的一个解元素；

                                                   }

由所有解元素组合成问题的一个可行解；

1. 实例
2. 回溯

https://blog.csdn.net/sinat\_24436879/article/details/44985535

https://blog.csdn.net/qq\_32400847/article/details/51474105

1. 一般思路

在当前场景下，存在若干种**选择**去操作，有可能两种结果：一是违反相应**条件限制**，只能返回（back），另一种是该选择选到最后居然**正确并结束**。

1. 实例

八皇后问题，图的着色问题，装载问题

1. 分支界定
2. 一般思路

采用广度优先的策略，依次搜索E-结点的所有分支，也就是所有相邻结点，抛弃不满足约束条件的结点，其余结点加入活结点表。然后从表中选择一个结点作为下一个E-结点，继续搜索

1. 和回溯法的区别

回溯法深度优先搜索堆栈活结点的所有可行子结点被遍历后才被从栈中弹出找出满足约束条件的所有解

分支限界法广度优先或最小消耗优先搜索队列、优先队列每个结点只有一次成为活结点的机会找出满足约束条件的一个解或特定意义下的最优解

1. 实例

**数据结构**

**类加载**

https://www.cnblogs.com/xdouby/p/5829423.html

1. 加载过程
2. 装载（load）：将类的.class文件中的二进制数据读入到内存中，将其放在运行时数据区的方法区内，然后在堆区创建一个这个类的java.lang.Class对象，用来封装类在方法区类的对象
3. 链接（link）
4. 验证：确保被加载类的正确性；
5. 准备：为类的静态变量分配内存，并将其初始化为默认值；
6. 解析：把类中的符号引用转换为直接引用；
7. 初始化（initialize）：为类的静态变量赋予正确的初始值
8. 类什么时候才被初始化
9. 遇到new,getstatic,putstatic,invokestatic这失调字节码指令时，如果类没有进行过初始化，则需要先触发其初始化。生成这4条指令的最常见的Java代码场景是：使用new关键字实例化对象的时候、读取或设置一个类的静态字段（被final修饰、已在编译器把结果放入常量池的静态字段除外）的时候，以及调用一个类的静态方法的时候。
10. 使用java.lang.reflect包的方法对类进行反射调用的时候，如果类没有进行过初始化，则需要先触发其初始化。
11. 当初始化一个类的时候，如果发现其父类还没有进行过初始化，则需要先触发其父类的初始化。
12. 当虚拟机启动时，用户需要指定一个要执行的主类（包含main()方法的那个类），虚拟机会先初始化这个主类。
13. 当使用jdk1.7动态语言支持时，如果一个java.lang.invoke.MethodHandle实例最后的解析结果REF\_getstatic,REF\_putstatic,REF\_invokeStatic的方法句柄，并且这个方法句柄所对应的类没有进行初始化，则需要先出触发其初始化。
14. 类的初始化步骤
15. 如果这个类还没有被加载和链接，那先进行加载和链接
16. 假如这个类存在直接父类，并且这个类还没有被初始化（注意：在一个类加载器中，类只能初始化一次），那就初始化直接的父类（不适用于接口）
17. 假如类中存在初始化语句（如static变量和static块），那就依次执行这些初始化语句。
18. 加载类的方式

类的加载的最终产品是位于堆区中的Class对象。Class对象封装了类在方法区内的数据结构，并且向Java程序员提供了访问方法区内的数据结构的接口

1. 从本地系统直接加载
2. 通过网络下载.class文件
3. 从zip，jar等归档文件中加载.class文件
4. 从专有数据库中提取.class文件
5. 将Java源文件动态编译为.class文件（服务器）
6. 加载器
7. 引导类加载器（bootstrap class loader）
8. 扩展类加载器（extensions class loader）
9. 系统类加载器（system class loader）
10. 类加载器的“双亲委派”机制

通俗的讲，就是某个特定的类加载器在接到加载类的请求时，首先将加载任务委托给父加载器，依次递归，如果父加载器可以完成类加载任务，就成功返回；只有父加载器无法完成此加载任务时，才自己去加载。

1. Java程序动态扩展方式
2. 调用java.lang.Class.forName(…)
3. 用户自定义类加载器

**当自定义类加载器没有指定父类加载器的情况下，默认的父类加载器即为系统类加载器**

**异常**

**Stream,IO**

**反射**

**设计模式**

1. 设计原则

总的原则：高内聚，低耦合

单一职责原则（Single Responsibility Principle）：一个类只负责一件事

开闭原则（Open Close Principle）：对于扩展开发，对于修改关闭

里式替换原则（Liskov Substitution Principle）：子类可以扩展父类的功能，但不能改变原有的功能

依赖倒转原则（Dependency Inversion Principle）：高层模块不应该依赖底层模块，两者都应该依赖其抽象；实现类之间不发生直接的依赖关系，其依赖关系是通过接口是来实现的

接口隔离原则（Interface Segregation Principle）：类间的依赖关系应该建立在最小的接口上

迪米特原则（Least KnowLedge Principle）：一个对象应该对其他对象保持最小的了解

1. 单例模式

饿汉，枚举，懒汉（优化），静态内部类

1. 简单工厂模式
2. 工厂方法模式
3. 抽象工厂模式
4. 策略模式
5. 适配器模式
6. 代理模式
7. 模板方法模式
8. 建造者模式
9. 外观模式

**常用框架（Spring）**

问答：

1.http://www.importnew.com/15851.html

2.<https://www.zhihu.com/question/48427693?sort=created>

spring 是一个 轻量级的 实现IOC 和 AOP的框架。

举些衍生问题的例子：

什么IOC？--> 为什么要IOC-->控制反转 是控制了谁对谁的反转?-->IOC容器完成了哪些事情-->做的这些事情与工厂模式有什么异同。

什么是AOP？--> 能做什么--> 一个简单的AOP打算怎么实现?

spring 的IOC除了做了bean实例化，还做了注入，注入可以对哪些要素进行注入？比如对字段注入，还可以对哪些要素注入？

除了做了实例化 与 注入 spring的IOC还干些什么事，让你在用IOC时更优雅？

如果让你做这个IOC，结合自己的思考或者曾经参与过的项目/产品 需要，你打算再做些什么设计能让使用者更优雅的使用?

除了spring实现了IOC 还了解过什么框架(或者框架内部)也实现了IOC？

spring与其他框架对接时，是怎么将IOC的容器中的bean交与其他框架使用的？比如对接struts2或者JUint 或者dubbo。

......

1. Spring IOC
2. Spring AOP

https://www.cnblogs.com/zhaozihan/p/5953063.html

1. 作用：在OOP中，正是这种分散在各处且与对象核心功能无关的代码（横切代码）的存在，使得模块复用难度增加。AOP则将封装好的对象剖开，找出其中对多个对象产生影响的公共行为，并将其封装为一个可重用的模块，这个模块被命名为“切面”（Aspect），切面将那些与业务无关，却被业务模块共同调用的逻辑提取并封装起来，减少了系统中的重复代码，降低了模块间的耦合度，同时提高了系统的可维护性。
2. 实现策略

（1）Java SE动态代理：  
    使用动态代理可以为一个或多个接口在运行期动态生成实现对象，生成的对象中实现接口的方法时可以添加增强代码，从而实现AOP。缺点是只能针对接口进行代理，另外由于动态代理是通过反射实现的，有时可能要考虑反射调用的开销。  
（2）字节码生成（CGLib 动态代理）  
    动态字节码生成技术是指在运行时动态生成指定类的一个子类对象，并覆盖其中特定方法，覆盖方法时可以添加增强代码，从而实现AOP。其常用工具是cglib。  
（3）定制的类加载器  
    当需要对类的所有对象都添加增强，动态代理和字节码生成本质上都需要动态构造代理对象，即最终被增强的对象是由AOP框架生成，不是开发者new出来的。解决的办法就是实现自定义的类加载器，在一个类被加载时对其进行增强。JBoss就是采用这种方式实现AOP功能。  
（4）代码生成  
    利用工具在已有代码基础上生成新的代码，其中可以添加任何横切代码来实现AOP。  
（5）语言扩展  
    可以对构造方法和属性的赋值操作进行增强，AspectJ是采用这种方式实现AOP的一个常见Java语言扩展。

1. 增强（Advice）
2. 织入（对方法增强）

①前置增强：实现MethodBeforeAdvice接口；

②后置增强：实现AfterReturningAdvice接口；

③环绕增强：实现MethodInterceptor接口。

1. Introduction Advice（引入增强）：对类的增强

继承DelegatingIntroductionInterceptor方法。

1. 基本概念
2. 通知（增强）Advice

通知定义了切面是什么以及何时使用，应该应用在某个方法被调用之前？之后？还是抛出异常时？等等。

1. 连接点 Join point

连接点是在应用执行过程中能够插入切面的一个点。这个点可以是调用方法时，抛出异常时，甚至修改一个字段时。切面代码可以利用这些点插入到应用的正常流程中，并添加新的行为。

1. 切点 Pointcut

切点有助于缩小切面所通知的连接点的范围。如果说通知定义了切面的“什么”和“何时”的话，那么切点就定义了“何处”，切点会匹配通知所要织入的一个或多个连接点，一般常用正则表达式定义所匹配的类和方法名称来指定这些切点。

1. 切面 Aspect

切面是通知和切点的结合。通知和切点定义了切面的全部内容——它是什么，在何时何处完成其功能。

1. 引入 Introduction

引入允许我们向现有的类添加新方法或属性，从而无需修改这些现有类的情况下，让他们具有新的行为和状态。

1. 织入 Weaving

在过去我常常把织入与引入的概念混淆，我是这样来辨别的，“引入”我把它看做是一个定义，也就是一个名词，而“织入”我把它看做是一个动作，一个动词，也就是切面在指定的连接点被织入到目标对象中。

1. 总结

通知包含了需要用于多个应用对象的横切行为；连接点是程序执行过程中能够应用通知的所有点；切点定义了通知被应用的具体位置（在哪些连接点）。其中关键的概念是切点定义了哪些连接点会得到通知（增强）。创建切点来定义切面所织入的连接点是AOP框架的基本功能。

另外，Spring是基于动态代理的，所以Spring只支持方法连接点，而像AspectJ和JBoss除了方法切点，它们还提供字段和构造器接入点。如果需要方法拦截之外的连接点拦截功能，则可以利用AspectJ来补充SpringAOP的功能。

1. 使用基于正则表达式的SpringAOP切面类
2. AOP自动代理
3. Spring框架自动生成代理。
4. spring根据Bean名称来生成自动代理
5. AspectJ execution 表达式拦截

定义一个切面类，实现环绕增强。@Aspect注解就不需要类再实现接口，@Around注解为AspectJ切点表达式，参数ProceedingJoinPoint的对象即为连接点，此连接点可以取得方法名，参数等等。

1. 常用注解（执行顺序）：

[@aspect](https://github.com/aspect) 定义切面

[@pointcut](https://github.com/pointcut) 定义切点 如：@Pointcut("execution(\* com.ctj.service.\*.\*(..))")

[@before](https://github.com/before) 标注Before Advice定义所在的方法

@afterreturning 标注After Returning Advice定义所在的方法

@afterthrowing 标注After Throwing Advice定义所在的方法

[@after](https://github.com/after) 标注 After(Finally) Advice定义所在的方法

[@around](https://github.com/around) 标注Around Advice定义所在的方法

1. AspectJ @DeclareParents 注解（引入增强）
2. Spring的AspectJ自动代理
3. 在XML中声明切面
4. Spring MVC
5. Spring Dao（JDBC）
6. Spring JDBC
7. Spring data jpa
8. Spring声明式事务管理，核心实现就是基于Aop
9. Spring声明式事务的属性

@Transactional(

readOnly = false, //读写事务

timeout = -1 , //事务的超时时间，-1为无限制

noRollbackFor = ArithmeticException.class, //遇到指定的异常不回滚

isolation = Isolation.DEFAULT, //事务的隔离级别，此处使用后端数据库的默认隔离级别

propagation = Propagation.REQUIRED //事务的传播行为

)

1. 基于注解的事务配置
2. 基于xml的事务配置
3. Spring Boot

**网络编程(socket，RMI)**

**Linux**

1. 常用指令

ls　　        显示文件或目录

     -l           列出文件详细信息l(list)

     -a          列出当前目录下所有文件及目录，包括隐藏的a(all)

mkdir         创建目录

     -p           创建目录，若无父目录，则创建p(parent)

cd               切换目录

touch          创建空文件

echo            创建带有内容的文件。

cat              查看文件内容

cp                拷贝

mv               移动或重命名

rm               删除文件

     -r            递归删除，可删除子目录及文件

     -f            强制删除

find              在文件系统中搜索某文件 如（find . -name “\*.txt”）

wc                统计文本中行数、字数、字符数

grep             在文本文件中查找某个字符串 如（grep "real" a.txt）

rmdir           删除空目录

tree             树形结构显示目录，需要安装tree包

pwd              显示当前目录

ln                  创建链接文件

more、less  分页显示文本文件内容

head、tail    显示文件头、尾内容 如（head -n2 a.txt）

ctrl+alt+F1  命令行全屏模式

1. 系统管理命令

stat              显示指定文件的详细信息，比ls更详细

who               显示在线登陆用户

whoami          显示当前操作用户

hostname      显示主机名

uname           显示系统信息

top                动态显示当前耗费资源最多进程信息

ps                  显示瞬间进程状态 ps -aux

du                  查看目录大小 du -h /home带有单位显示目录信息

df                  查看磁盘大小 df -h 带有单位显示磁盘信息

ifconfig          查看网络情况

ping                测试网络连通

netstat          显示网络状态信息

man                命令不会用了，找男人  如：man ls

clear              清屏

alias               对命令重命名 如：alias showmeit="ps -aux" ，另外解除使用unalias showmeit

kill                 杀死进程，可以先用ps 或 top命令查看进程的id，然后再用kill命令杀死进程。

HUP 1 终端断线

INT 2 中断（同 Ctrl + C）

QUIT 3 退出（同 Ctrl + \）

TERM 15 终止

KILL 9 强制终止

CONT 18 继续（与STOP相反， [fg](http://man.linuxde.net/fg)/[bg](http://man.linuxde.net/bg)命令）

STOP 19 暂停（同 Ctrl + Z）

1. 打包压缩相关命令

gzip：

bzip2：

tar:                打包压缩

     -c              归档文件

     -x              压缩文件

     -z              gzip压缩文件

     -j              bzip2压缩文件

     -v              显示压缩或解压缩过程 v(view)

     -f              使用档名

例：

tar -cvf /home/abc.tar /home/abc              只打包，不压缩

tar -zcvf /home/abc.tar.gz /home/abc        打包，并用gzip压缩

tar -jcvf /home/abc.tar.bz2 /home/abc      打包，并用bzip2压缩

当然，如果想解压缩，就直接替换上面的命令  tar -cvf  / tar -zcvf  / tar -jcvf 中的“c” 换成“x” 就可以了。

1. 关机/重启机器

shutdown

     -r             关机重启

     -h             关机不重启

     now          立刻关机

halt               关机

reboot          重启

1. Linux管道

将一个命令的标准输出作为另一个命令的标准输入。也就是把几个命令组合起来使用，后一个命令除以前一个命令的结果。

例：grep -r "close" /home/\* | more       在home目录下所有文件中查找，包括close的文件，并分页输出。

1. Linux软件包管理

**dpkg** (Debian Package)管理工具，软件包名以.deb后缀。这种方法适合系统不能联网的情况下。

比如安装tree命令的安装包，先将tree.deb传到Linux系统中。再使用如下命令安装。

sudo dpkg -i tree\_1.5.3-1\_i386.deb         安装软件

sudo dpkg -r tree                                     卸载软件

注：将tree.deb传到Linux系统中，有多种方式。VMwareTool，使用挂载方式；使用winSCP工具等；

**APT**（Advanced Packaging Tool）高级软件工具。这种方法适合系统能够连接互联网的情况。

依然以tree为例

sudo apt-get install tree                         安装tree

sudo apt-get remove tree                       卸载tree

sudo apt-get update                                 更新软件

sudo apt-get upgrade

将.**rpm**文件转为.**deb**文件

.rpm为RedHat使用的软件格式。在Ubuntu下不能直接使用，所以需要转换一下。

sudo alien abc.rpm

1. vim使用

vim三种模式：命令模式、插入模式、编辑模式。使用ESC或i或：来切换模式。

命令模式下：

:q                      退出

:q!                     强制退出

:wq                   保存并退出

:set number     显示行号

:set nonumber  隐藏行号

/apache            在文档中查找apache 按n跳到下一个，shift+n上一个

yyp                   复制光标所在行，并粘贴

h(左移一个字符←)、j(下一行↓)、k(上一行↑)、l(右移一个字符→)

vi操作

1.跳到文本的最后一行：按“G”,即“shift+g”

2.跳到最后一行的最后一个字符 ： 先重复1的操作即按“G”，之后按“$”键，即“shift+4”。

3.跳到第一行的第一个字符：先按两次“g”，

4.跳转到当前行的第一个字符：在当前行按“0”。

5.vi加密。进入vi，输入":"  + "X" 之后就提示你输入两次密码。

6.文件重新载入 :e!

7.单行复制  将光标移到复制行 按 'yy'进行复制

8.多行复制  将光标移到复制首行 按 'nyy'进行复制 n=1.2.3.4。。。。。

9.粘贴 将光标移到粘贴行 按 'p'进行粘贴

10.查找 /pattern Enter

1. 用户及用户组管理

/etc/passwd    存储用户账号

/etc/group       存储组账号

/etc/shadow    存储用户账号的密码

/etc/gshadow  存储用户组账号的密码

useradd 用户名

userdel 用户名

adduser 用户名

groupadd 组名

groupdel 组名

passwd root     给root设置密码

su root

su - root

/etc/profile     系统环境变量

bash\_profile     用户环境变量

.bashrc              用户环境变量

su user              切换用户，加载配置文件.bashrc

su - user            切换用户，加载配置文件/etc/profile ，加载bash\_profile

**更改文件的用户及用户组**

sudo chown [-R] owner[:group] {File|Directory}

例如：还以jdk-7u21-linux-i586.tar.gz为例。属于用户hadoop，组hadoop

要想切换此文件所属的用户及组。可以使用命令。

sudo chown root:root jdk-7u21-linux-i586.tar.gz

1. 文件权限管理

三种基本权限

R           读         数值表示为4

W          写         数值表示为2

X           可执行  数值表示为1

如图所示，jdk-7u21-linux-i586.tar.gz文件的权限为-rw-rw-r--

-rw-rw-r--一共十个字符，分成四段。

第一个字符“-”表示普通文件；这个位置还可能会出现“l”链接；“d”表示目录

第二三四个字符“rw-”表示当前所属用户的权限。   所以用数值表示为4+2=6

第五六七个字符“rw-”表示当前所属组的权限。      所以用数值表示为4+2=6

第八九十个字符“r--”表示其他用户权限。              所以用数值表示为2

所以操作此文件的权限用数值表示为662

**更改权限**

sudo chmod [u所属用户  g所属组  o其他用户  a所有用户]  [+增加权限  -减少权限]  [r  w  x]   目录名

例如：有一个文件filename，权限为“-rw-r----x” ,将权限值改为"-rwxrw-r-x"，用数值表示为765

sudo chmod u+x g+w o+r  filename

上面的例子可以用数值表示

sudo chmod 765 filename

链接：<http://www.importnew.com/17232.html>

<http://www.importnew.com/21089.html>

<http://www.importnew.com/18126.html>

<https://www.cnblogs.com/zhguang/p/3257367.html>

<http://blog.csdn.net/cy609329119/article/details/64905053>

<http://www.importnew.com/19816.html>

<https://www.cnblogs.com/Binhua-Liu/p/5326540.html>

<https://www.cnblogs.com/ranyonsue/p/5984001.html>