1. java小知识

1. TreeSet是有序的set集合

2. 用Scanner的next()和nextLine()读取字符串时的区别：

next()，对于最前面的空格、TAB、回车不识别，后面将这视为分隔符和结束符，所以不能得到带空格的字符串

nextLine()方法的结束符只有回车

next() 方法读取一个由分割符分割的字符串

nextLine()读取一个以换行符结束的行

3. parseInt()和Integer.valueOf()以及valueOf(),toString()

Integer.parseInt(String str)返回的是int

Integer.valueOf(String str)返回的是Integer(现在的API支持自动装箱拆箱)

intValue()是将integer转换成int

4. Arrays.sort()和Collection.sort()

底层都类似于归并排序

5. this和super的用法

super（参数）：调用基类中的某一个构造函数（应该为构造函数中的第一条语句）

this（参数）：调用本类中另一种形成的构造函数（应该为构造函数中的第一条语句）

super:　它引用当前对象的直接父类中的成员（用来访问直接父类中被隐藏的父类中成员数据或函数，基类与派生类中有相同成员定义时如：super.变量名 super.成员函数据名（实参）

this：它代表当前对象名（在程序中易产生二义性之处，应使用this来指明当前对象；如果函数的形参与类中的成员数据同名，这时需用this来指明成员变量名）

调用super()必须写在子类构造方法的第一行，否则编译不通过。每个子类构造方法的第一条语句，都是隐含地调用super()，如果父类没有这种形式的构造函数，那么在编译的时候就会报错。

super()和this()类似,区别是，super()从子类中调用父类的构造方法，this()在同一类内调用其它方法。

super()和this()均需放在构造方法内第一行。

尽管可以用this调用一个构造器，但却不能调用两个。

this和super不能同时出现在一个构造函数里面，因为this必然会调用其它的构造函数，其它的构造函数必然也会有super语句的存在，所以在同一个构造函数里面有相同的语句，就失去了语句的意义，编译器也不会通过。

this()和super()都指的是对象，所以，均不可以在static环境中使用。包括：static变量,static方法，static语句块。

从本质上讲，this是一个指向本对象的指针, 然而super是一个Java关键字。

6. 继承

在一个子类被创建的时候，首先会在内存中创建一个父类对象，然后在父类对象外部放上子类独有的属性，

两者合起来形成一个子类的对象。所以所谓的继承使子类拥有父类所有的属性和方法其实可以这样理解，

子类对象确实拥有父类对象中所有的属性和方法，但是父类对象中的私有属性和方法，子类是无法访问到的，

只是拥有，但不能使用。就像有些东西你可能拥有，但是你并不能使用。所以子类对象是绝对大于父类对象的，

所谓的子类对象只能继承父类非私有的属性及方法的说法是错误的。可以继承，只是无法访问到而已。

当子类覆盖父类的成员变量时，父类方法使用的是父类的成员变量，子类方法使用的是子类的成员变量

（1）子类覆盖父类的方法，必须有同样的参数返回类型，否则编译不能通过

（2）子类覆盖父类的方法，在jdk1.5后，参数返回类可以是父类方法返回类的子类

（3）子类覆盖父类方法，可以修改方法作用域修饰符，但只能把方法的作用域放大，而不能把public修改为private

(4)子类方法能够访问父类的protected作用域成员，不能够访问默认的作用域成员

（5）子类的静态方法不能隐藏同名的父类实例方法

那么在继承中子类如何才能继承到父类的私有域呢？

答案是：在子类的构造方法中通过super()方法调用父类的构造方法。

也就是，在构造子类的同时，为子类构造出跟父类相同的域。如此就在子类的对象中，也拥有了父类声明的域了。

7. <T extends Comparable<? super T>>

public static <T extends Comparable<? super T>> void sort(List<T> list)

<T extends Comparable<T>>

类型 T 必须实现 Comparable 接口，并且这个接口的类型是 T。只有这样，T 的实例之间才能相互比较大小。例如，在实际调用时若使用的具体类是 Dog，那么 Dog 必须 implements Comparable<Dog>

<T extends Comparable<? super T>>

类型 T 必须实现 Comparable 接口，并且这个接口的类型是 T 或 T 的任一父类。这样声明后，T 的实例之间，T 的实例和它的父类的实例之间，可以相互比较大小。例如，在实际调用时若使用的具体类是 Dog

(假设 Dog 有一个父类 Animal），Dog 可以从 Animal 那里继承 Comparable<Animal> ，或者自己 implements Comparable<Dog> 。

8. 访问修饰符

访问权限 类 包 子类 其他包

public： ok ok ok ok

protected ok ok ok no

默认 ok ok no no

private ok no no no

9. Comparator 和 Comparable

Comparable定义在类的内部

实现compareTo(Object o2) o1.compareTo(Object o)

Comparator定义在类的外部

比如：对list排序时，Collections.sort(personList, new PersonComparator() )，将list和比较器都传进去.

实现compare(Object o1, Object o2)

10. Entry是Map的内部类

11. java集合框架

集合(collection)

Set:存储不重复的元素

HashSet

LinkedHashSet

TreeSet 有序的

List

ArrayList

LinkedList：实现了Deque接口和List接口

Queue

Queue

双向队列 Deque：用LinkedList实现双向队列

优先队列：拥有最小数值的元素被赋予最高优先级

向量Vector和栈类Stack：对于不需要同步的程序来说，使用ArrayList比使用Vector效率高

规则集比线性表更加高效，如果应用程序用规则集足够，那就是用规则集，除此之外，如果程序不需要特别的顺序，就选择散列集

图(Map)

内部接口Entry

HashMap 随机 拉链法

LinkedHashMap 访问的时间顺序

TreeMap 有序的 基于红黑树实现

HashMap 不是同步的，HashTable是同步的，HashTable不允许null键或值，HashMap允许null键或值。

11. String[] words = text.split("[ \n\t\r.,:;!?(){]")

12. Reverse Polish Notation:逆波兰式：

运算符写在操作数之后；

3 32 + 5 3 \* -

　　12 34 2 - \* 8 /

　　乍一看上面两个式子很奇怪，是吗？它们就是一种表达式的记法——逆波兰表达式。

　　现在，准备一个很窄的圆筒，筒是有底的，像一个细长的杯子，粗细刚好和一枚硬币相当。再做几个和硬币一样大的小圆纸片，在纸片上依次写上“3”“32”“+”“5”“3”“\*”“-”，记住，每个纸片上要么只写一个数，要么只写一个运算符号，把它们按上面的顺序排好。好，现在仔细听我说，按顺序一个接一个地拿起小圆纸片，反复执行以下几个规则：

　　(1) 如果你拿着的是一个数，不多说，直接把它放进圆筒；

　　(2) 如果你拿着的是一个运算符号，不要把它放进去。先从圆筒里取出两个数（当然是先取最上而的啦，筒很细的），然后处这两个数作运算符号指定的运算，并把结果写在一张新的纸片上，然后放进筒里。比如你拿着的是“+”，你要依次取出“32”和“3”，让它们相加，得“35”，把“35”写在一张新纸片上（现在“34”和“12”可以扔掉了），并把这张新纸片放进圆筒。

　　当圆筒里只有一个数时，你就可以停下来了，我猜这个数是20，没错，这就是这个表达式的值！

13. Object1.equals(Object2) 和 ==

关系操作符生成的是一个boolean结果，它们计算的是操作数的值之间的关系；

说的简单点，==就是用来比较值是否相等，引用相同时返回值为true

equals方法是基类Object中的方法

很显然，在Object类中，equals方法是用来比较两个对象的引用是否相等，即是否指向同一个对象。

String类对equals方法进行了重写，用来比较指向的字符串对象所存储的字符串是否相等。

其他的一些类诸如Double，Date，Integer等，都对equals方法进行了重写用来比较指向的对象所存储的内容是否相等。

总结来说：

　　1）对于==，如果作用于基本数据类型的变量，则直接比较其存储的 “值”是否相等；

　　　　如果作用于引用类型的变量，则比较的是所指向的对象的地址

　　2）对于equals方法，注意：equals方法不能作用于基本数据类型的变量

　　　　如果没有对equals方法进行重写，则比较的是引用类型的变量所指向的对象的地址；

　　　　诸如String、Date等类对equals方法进行了重写的话，比较的是所指向的对象的内容。

14. 内部类

（1）成员内部类：成员内部类可以无条件访问外部类的所有成员属性和成员方法(包含private，原因是编译器会默认为成员内部类添加一个指向外部类对象的引用)。当成员内部类拥有和外部类同名的成员变量或者方法时，会发生隐藏现象，即默认情况下访问的是成员内部类的成员。如果要访问外部类的同名成员，需要以下面的形式进行访问：

外部类.this.成员变量

外部类.this.成员方法

外部类想访问成员内部类的成员不是随心所欲的。必须先创建一个成员内部类的对象，再通过这个对象的引用来访问。成员内部类是依附外部类而存在的，也就是说，如果要创建成员内部类的对象，前提是必须存在一个外部类的对象。

（2）局部内部类：局部内部类定义在一个方法或者一个作用域里面的类，他和成员内部类的区别在于局部内部类的访问权限仅限于方法内或者该作用域内。（和方法的一个局部变量一样，不能有public、protected、private和static修饰的）

（3）匿名内部类：多用于设置监听器对象。匿名内部类不能有访问修饰符和static修饰符。是唯一没有构造器的类，所以匿名类的使用非常有限，大部分匿名内部类用于接口回调。匿名内部类在编译的时候由系统自动起名为Outter$1.class。一般来说，匿名内部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，只是对继承方法的实现或是重写。

（4）静态内部类：静态内部类也是定义在另一个类里面的类，只不过在类的前面多了一个关键字static。不需要依赖于外部类的，这点和类的静态成员属性有点类似，并且它不能使用外部类的非static成员变量或者方法，这点很好理解，因为在没有外部类的对象的情况下，可以创建静态内部类的对象，如果允许访问外部类的非static成员就会产生矛盾，因为外部类的非static成员必须依附于具体的对象。

**局部内部类和匿名内部类只能访问局部final变量（为什么，反编译）**

**使用内部类的好处：**

1.每个内部类都能独立的继承一个接口的实现，所以无论外部类是否已经继承了某个(接口的)实现，对于内部类都没有影响。内部类使得多继承的解决方案变得完整，

2.方便将存在一定逻辑关系的类组织在一起，又可以对外界隐藏。

3.方便编写事件驱动程序

4.方便编写线程代码

15. 反射

Java的反射机制是在编译并不确定是哪个类被加载了，而是在程序运行的时候才加载、探知、自审。使用在编译期并不知道的类。这样的特点就是反射。

要正确使用Java反射机制就得使用java.lang.Class这个类。它是Java反射机制的起源。当一个类被加载以后，Java虚拟机就会自动产生一个Class对象。通过这个Class对象我们就能获得加载到虚拟机当中这个Class对象对应的方法、成员以及构造方法的声明和定义等信息。

使用反射机制的步骤：

获得想操作类的java.lang.Class对象

获得方法和数据域

使用反射API来操作这些信息

**构造对象**

**无参**

Class cla = Class.forName(“类的路径”);

one = (类名)cla.newInstance();产生这个Class类对象的一个实例，调用无参的构造方法。

**有参**

//获得指定字符串类对象

Class cla=Class.forName("reflect.Tests");

 //设置Class对象数组，用于指定构造方法类型

 Class[] cl=**new** Class[]{**int**.**class**,**int**.**class}**

 //获得Constructor构造器对象。并指定构造方法类型

  Constructor con=cla.getConstructor(cl)

//给传入参数赋初值

 Object[] x={**new** Integer(33),**new** Integer(67)};

 //得到实例

 Object obj=con.newInstance(x);

**执行实例方法**

代码示例：

**package** reflect;

**import** java.lang.reflect.Method;

/\*\*

 \*

 \* 本类测试反射获得类的方法对象，

 \* 并通过类对象和类方法对象，运行该方法

 \*

 \*/

**public** **class** MethodTest {

**public** **static** **void** main(String[] args) {

**try** {

           //获得窗体类的Class对象

           Class cla=Class.forName("javax.swing.JFrame");

           //生成窗体类的实例

           Object obj=cla.newInstance();

       //获得窗体类的setSize方法对象，并指定该方法参数类型为int,int

           Method methodSize=cla.getMethod("setSize", **new** Class[]{**int**.**class**,**int**.**class**});

           /\*

            \* 执行setSize()方法，并传入一个Object[]数组对象，

            \* 作为该方法参数，等同于  窗体对象.setSize(300,300);

            \*/

           methodSize.invoke(obj, **new** Object[]{**new** Integer(300),**new** Integer(300)});

       //获得窗体类的setSize方法对象，并指定该方法参数类型为boolean

           Method methodVisible=cla.getMethod("setVisible", **new** Class[]{**boolean**.**class**});

           /\*

            \* 执行setVisible()方法，并传入一个Object[]数组对象，              \*作为该方法参数。 等同于  窗体对象.setVisible(true);

            \*/

           methodVisible.invoke(obj, **new** Object[]{**new** Boolean(**true**)});

       } **catch** (Exception e) {

           e.printStackTrace();

       }

    }

}

16. java输入中mark和reset的用法

17. final的用法

String就是一个final类。

final修饰一个类时，表明这个类不能被继承。final类中的成员变量可以根据需要设为final，但要注意final类中的所有成员方法都会被隐式的指定为final方法。

final修饰一个方法时，原因有二：

1. 把方法锁定，以防任何继承类修改它的含义
2. 效率，；（现已不常用）

final修饰一个变量时，如果是基本数据类型的变量，则其数值一旦初始化之后便不能更改；如果是引用变量时，则在对其初始化之后便不能再让其指向另外一个对象。

final修饰的引用变量永远无法改变，但是对象的值（实例变量）本身仍然是可变的。

18. static的用法

“static方法就是没有this的方法。在static方法内部不能调用非静态方法，反过来是可以的。而且可以在没有创建任何对象的前提下，仅仅通过类本身来调用static方法。这实际上正是static方法的主要用途。”

在静态方法中不能访问非静态成员方法和非静态成员变量，但是在非静态成员方法中是可以访问静态成员方法/变量的。

另外记住，即使没有显示地声明为static，类的构造器实际上也是静态方法。

static关键字还有一个比较关键的作用就是 用来形成静态代码块以优化程序性能。static块可以置于类中的任何地方，类中可以有多个static块。在类初次被加载的时候，会按照static块的顺序来执行每个static块，并且只会执行一次。

static是不允许用来修饰局部变量

19. random的用法

Random类中实现的随机算法是伪随机，也就是有规则的随机。在进行随机时，随机算法的起源数字称为种子(seed)，在种子数的基础上进行一定的变换，从而产生需要的随机数字。

相同种子数的Random对象，相同次数生成的随机数字是完全相同的。也就是说，两个种子数相同的Random对象，第一次生成的随机数字完全相同，第二次生成的随机数字也完全相同。

Random() 使用一个和当前系统时间对应的相对时间有关的数字作为种子数，然后使用这个种子数构造Random对象。

Random(long seed) 该构造方法可以通过制定一个种子数进行创建。

**再次强调：种子数只是随机算法的起源数字，和生成的随机数字的区间无关。**

**Math类中也有一个random方法。生成一个[0, 1.0)区间的随机小数，通过阅读Math类的源代码可以发现，Math类中的random方法就是直接调用Random类中的nextDouble方法实现的。**

20. 在java中，没有被显式的初始化的字段会被自动初始化。

boolean：false 数值：0 引用类型：null

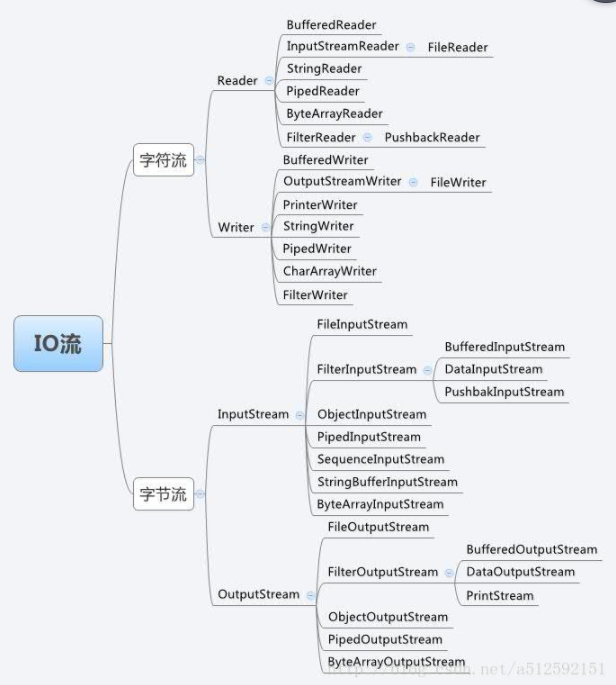
虽然字段会被自动初始化，但局部变量不会被自动初始化

21. 接口中无法定义实例变量； 必须是初始化的

22. 进程和线程的区别

进程就是一个人软件，进程可以包含许多线程；

23. 输入输出流



读写对象时，被读取和写入的类必须实现Serializable接口。

PrintStream就是System.out这个对象。

Java使用Unicode存储字符串。

Java的输入流的read(byte[] b)有参方法返回的是填充给缓冲数组的长度，无参read()返回的是一个字节，给int变量。

Scanner从文件读取文件，PrintWriter向文件中写入数据

Scanner ：标记读取方法不能读取后面的分隔符(nextInt nextDouble)。

任何AutoClosable类型的对象可以用于try-with-resource语法中，实现自动关闭。

24.Serializable接口

该接口中并没有什么方法，是标记能序列化和反序列化，加载到输入输出流的对象

25.Properties

用于读取java的配置文件

26.反编译

27.Double.POSITIVE\_INFINITY和Double.NEGATIVE\_INFINITY

28.字符串String没有实现Iterable接口

29.java内存分配大小（算法 v4 P126）

对象本身的开销为16字节

30.Deprecated？

31.堆、方法区、栈区

堆区：只存放类对象，线程共享；

方法区：又叫静态存储区，存放class文件和静态数据，线程共享;

栈区：存放方法局部变量，基本类型变量区、执行环境上下文、操作指令区，线程不共享;

32.String、StringBuffer、StringBuilder

执行速度比较：StringBuilder>StringBuffer>String

String:字符串常量，适用于少量数据

StringBuilder:字符串变量，线程非安全，单线程，操作大量数据

StringBuffer：字符串变量，线程安全的，多线程，操作大量数据

当用String操作字符串时，实际上是在不断的创建新的对象，而原来的对象就会变为垃圾被ＧＣ回收掉，可想而知这样执行效率会有多底。

而StringBuffer与StringBuilder就不一样了，他们是字符串变量，是可改变的对象，每当我们用它们对字符串做操作时，实际上是在一个对象上操作的，这样就不会像String一样创建一些而外的对象进行操作了，当然速度就快了。

33.Arrays.sort()

对于原始数据类型，使用三向切分的快速排序；

对于引用类型使用的是归并排序；

34.Class类

String className = "Example";

          class c = Class.forName(className);

          factory = (ExampleInterface)c.newInstance();

（1）getName()

一个Class对象描述了一个特定类的属性，Class类中最常用的方法getName以 String 的形式返回此 Class 对象所表示的实体（类、接口、数组类、基本类型或 void）名称。

（2）newInstance()

Class还有一个有用的方法可以为类创建一个实例，这个方法叫做newInstance()。例如：  
    x.getClass.newInstance()，创建了一个同x一样类型的新实例。newInstance()方法 调用默认构造器（无参数构造器）初始化新建对象。

（3）getClassLoader()

返回该类的类加载器。

（4）getComponentType()   
    返回表示数组组件类型的 Class。

（5）getSuperclass()   
    返回表示此 Class 所表示的实体（类、接口、基本类型或 void）的超类的 Class。

（6）isArray()   
    判定此 Class 对象是否表示一个数组类。

new关键字和newInstance的区别：

（1） newInstance: 弱类型。低效率。只能调用无参构造。    
       （2） new: 强类型。相对高效。能调用任何public构造。

得到Class的对象的三种方法：

1. MyObject x;

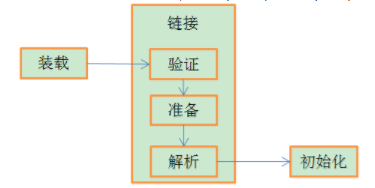
Class c1 = x.getClass();

(2) Class c2=Class.forName("MyObject");

(3) Class cl1 = Manager.class;  
    Class cl2 = int.class;  
    Class cl3 = Double[].class;

35.类加载器

JVM将类加载分为三个过程：装载（Load），链接（Link）和初始化（Initialize），链接又分为三个步骤：



1) 装载：查找并加载类的二进制数据；

2) 链接：

验证：确保被加载类的正确性；

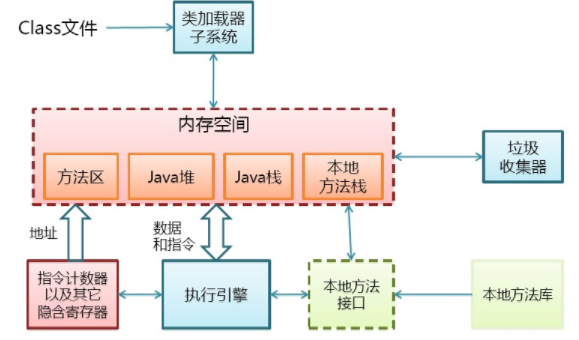
准备：为类的静态变量分配内存，并将其初始化为默认值；

解析：把类中的符号引用转换为直接引用；

3)初始化：为类的静态变量赋予正确的初始值；

那为什么我要有验证这一步骤呢？首先如果由编译器生成的class文件，它肯定是符合JVM字节码格式的，但是万一有高手自己写一个class文件，让JVM加载并运行，用于恶意用途，就不妙了，因此这个class文件要先过验证这一关，不符合的话不会让它继续执行的，也是为了安全考虑吧。

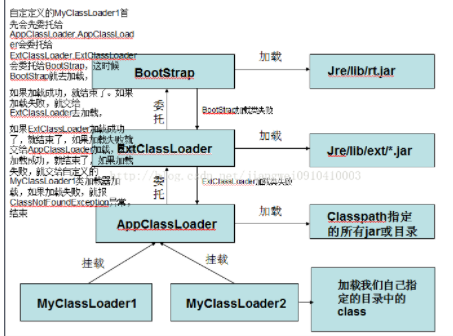
准备阶段和初始化阶段看似有点牟盾，其实是不牟盾的，如果类中有语句：private static int a = 10，它的执行过程是这样的，首先字节码文件被加载到内存后，先进行链接的验证这一步骤，验证通过后准备阶段，给a分配内存，因为变量a是static的，所以此时a等于int类型的默认初始值0，即a=0,然后到解析（后面在说），到初始化这一步骤时，才把a的真正的值10赋给a,此时a=10。



Java虚拟机中可以安装多个类加载器，系统默认三个主要的类加载器，每个类负责加载特定位置的类：

**BootStrap**,**ExtClassLoader**,**AppClassLoader**

类加载器也是Java类，因为Java类的类加载器本身也是要被类加载器加载的，显然必须有第一个类加载器不是Java类，这个正是BootStrap,使用C/C++代码写的，已经封装到JVM内核中了，而ExtClassLoader和AppClassLoader是Java类。



**类加载器的委托机制：**

当Java虚拟机要加载第一个类的时候，到底派出哪个类加载器去加载呢？

(1)首先当前线程的类加载器去加载线程中的第一个类(当前线程的类加载器：Thread类中有一个get/setContextClassLoader(ClassLoader cl);方法，可以获取/指定本线程中的类加载器)。

(2)如果类A中引用了类B,Java虚拟机将使用加载类A的类加载器来加载类B。

(3)还可以直接调用ClassLoader.loadClass(String className)方法来指定某个类加载器去加载某个类。

每个类加载器加载类时，又先委托给其上级类加载器当所有祖宗类加载器没有加载到类，回到发起者类加载器，还加载不了，则会抛出 ClassNotFoundException,不是再去找发起者类加载器的儿子，因为没有getChild()方法。例如：如上图所示： MyClassLoader->AppClassLoader->Ext->ClassLoader->BootStrap.自定定义的MyClassLoader1首先会先委托给AppClassLoader,AppClassLoader会委托给ExtClassLoader,ExtClassLoader会委托给BootStrap，这时候BootStrap就去加载，如果加载成功，就结束了。如果加载失败，就交给ExtClassLoader去加载，如果ExtClassLoader加载成功了，就结束了，如果加载失败就交给AppClassLoader加载，如果加载成功，就结束了，如果加载失败，就交给自定义的MyClassLoader1类加载器加载，如果加载失败，就报ClassNotFoundException异常，结束。

**类的唯一性：**

对于任意一个类，都需要由*加载它的类加载器*和*类的全限定名*一同确定其在Java虚拟机中的***唯一性***。

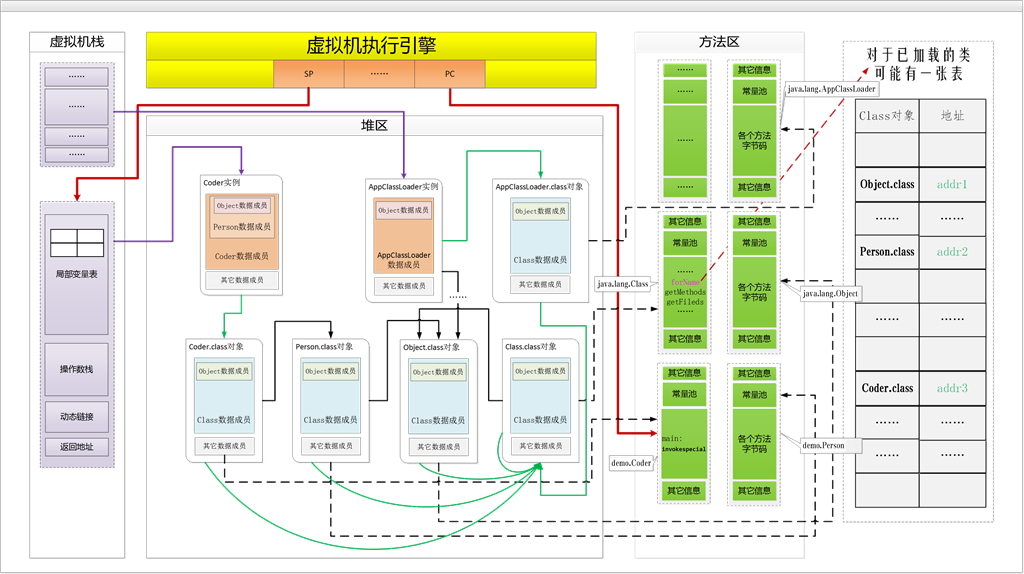
换句话说就是，比较两个类是否“相等”，只有在这两个类是由同一个类加载器加载的前提下才有意义，否则，即使两个类来源于同一个Class文件，被同一个虚拟机加载，只要加载他们的类加载器不同，那这两个类就必定不相等。

这里所指的“相等”，包括Class对象的equals()方法，isAssignableFrom()方法、isInstance()方法的返回结果，也包括使用instanceof关键字做对象所属关系判定等情况。

**数组类本身不通过类加载器创建，是由虚拟机直接创建的。**

**当我们编写好一个“.java”文件，通过javac编译器编译后会形成一个“.class”文件。当我们运行该文件时，Java虚拟机就通过类加载器（类加载器本质就是一段程序）把“.class”文件加载到内存，在方法区形成该类各方法的代码段和描述该类细节信息的常量池，同时在堆区形成一个表示该类的Class对象（一个java.lang.Class类的实例）。Class对象中存储了指向了该类所有属性和方法的详细信息的指针（同时，还存储了指向该类的父类的Class对象的指针）。我们能够通过Class对象直接创建该类的实例，并调用该类的所有方法，这就是我们所说的反射。**

**类加载器不仅仅可以加载本地文件系统中的“.class”文件，还可以通过各种形式进行加载，比如通过网络上的获取的数据流作为 “.class”。**



36.本地（native）方法

37.synchronized

Synchronized块：

锁住此对象：Synchronized（this）

锁住生成的对象：Synchronized（object）

wait(), notify(), notifyAll() 上厕所理论

互斥锁：当存在多个线程操作共享数据时，需要保证同一时刻有且只有一个线程在操作共享数据，其他线程必须等到该线程处理完数据后再进行，这种方式有个高尚的名称叫互斥锁

三种应用方式：

* 修饰实例方法，作用于当前实例加锁，进入同步代码前要获得当前实例的锁
* 修饰静态方法，作用于当前类对象加锁，进入同步代码前要获得当前类对象的锁
* 修饰代码块，指定加锁对象，对给定对象加锁，进入同步代码库前要获得给定对象的锁。

synchronized的可重入性

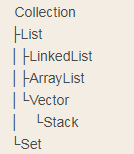
38.反射返回方法

**public Method[] getDeclaredMethods()**返回类或接口声明的所有方法，包括public, protected, default (package) 访问和private方法的Method对象，但不包括继承的方法。当然也包括它所实现接口的方法。

**public Method[] getMethods()**返回类的所有public方法，包括其继承类的公用方法，当然也包括它所实现接口的方法。

39.Collection和Collections

java.util.Collection 是一个集合接口（集合类的一个顶级接口）



Collections则是集合类的一个工具类/帮助类，其中提供了一系列静态方法，用于对集合中元素进行排序、搜索以及线程安全等各种操作。

1. 混排（Shuffling）
2. 反转(Reverse)
3. 排序(Sort)
4. 替换所以的元素(Fill)
5. 拷贝(Copy)。用两个参数，一个目标 List 和一个源 List, 将源的元素拷贝到目标，并覆盖它的内容。目标 List 至少与源一样长。如果它更长，则在目标 List 中的剩余元素不受影响。Collections.copy(list,li): 后面一个参数是目标列表 ,前一个是源列表
6. 返回Collections中最小元素(min)
7. 返回Collections中最小元素(max)

40.java语言复制数组的四种方法

for循环逐一复制

System.arraycopy()

System.copyof()

使用clone()

效率：System.arrayCopy() > clone > Arrays.copyof() > for循环

41.clone()的用法

java.lang.Object类的clone()方法为protected类型，不可直接调用，需要先对要克隆的类进行下列操作：

首先被克隆的类实现Cloneable接口；然后在该类中覆盖clone()方法，并且在该clone()方法中调用super.clone()；这样，super.clone()便可以调用java.lang.Object类的clone()方法。

42.java位运算

-1>>>1 == Integer.MAX\_VALUE

m\*2^n 位运算：m << n

判断奇数偶数

不用临时变量交换两个数：

int a, b

a=a^b

b=b^a

a=a^b

0^a = a

取绝对值：（a^(a>>31)）-(a>>31）

a>>31取得a的符号，若a为正数，a>>31等于0，a^0=a，不变；若a为负数,a>>31等于-1 ，a^-1翻转每一位.

43.工厂模式

定义一个用于创建对象的接口，让子类决定实例化哪一个类。工厂方法使一个类的实例化延迟到其子类

如果是一个工厂，利用静态工厂模式，不需要工厂父类。

44.动态代理（反射）

1. java里的class文件加载分为两种情况，一种就是类型是编译器已知的，这种文件的.class文件在编译的时候，编译器会把.class文件打开检查，但是注意不是加载哦，第二种就是我们可能是从别的地方获取到了一个引用，然后动态的把这个未知类型的引用的对象的.class文件加载进jvm虚拟机里。
2. 在java.lang.Object 类中定义了getClass()方法，因此对于任意一个Java对象，都可以通过此方法获得对象的类型。Class类是Reflection API 中的核心类，它有以下方法**getName()：获得类的完整名字。getFields()：获得类的public类型的属性。getDeclaredFields()：获得类的所有属性。getMethods()：获得类的public类型的方法。getDeclaredMethods()：获得类的所有方法。**
3. getMethod(String name, Class[] parameterTypes)：获得类的特定方法，name参数指定方法的名字，parameterTypes 参数指定方法的参数类型。getConstructors()：获得类的public类型的构造方法。getConstructor(Class[] parameterTypes)：获得类的特定构造方法，parameterTypes 参数指定构造方法的参数类型。newInstance()：通过类的不带参数的构造方法创建这个类的一个对象。
4. 动态代理步骤

1. 创建一个实现接口InvocationHandler的类，它必须实现invoke方法

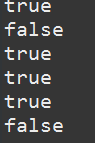
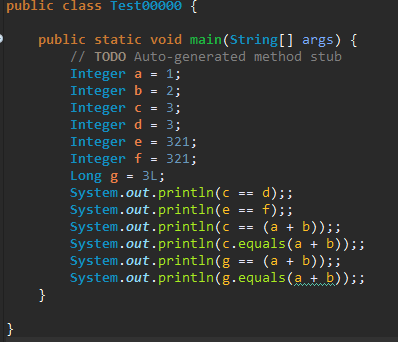
2. 创建被代理的类以及接口

3. 通过Proxy的静态方法newProxyInstance(ClassLoader loader, Class[] interfaces, InvocationHandler h) 创建一个代理

4. 通过代理调用方法

45.自动装箱拆箱

类的“==”运算在不遇到算数运算的情况下不会自动拆箱，以及equal()方法不处理数据转型的关系，



46.线程和volatile

**如果一个对象可以安全的被多个线程同时使用，那他就是线程安全的**

**都能得到正确的结果**

read load，store write

变量在工作内存中改变了之后必须把该改变同步回内存

**volatile具有两种特性：**

1. **保证此变量对所有线程的可见性**，这里的可见性是指当一条线程修改了这个变量的值，新值对于其他线程来说是可以立即得知的，而普通变量不能做到这一点。普通变量的值在线程间传递均需要通过主内存来完成。
2. **禁止指令重排序优化**

**线程调度方式：**协同式线程调度，抢占式线程调度**（java）**

**设置优先级**给某些线程多分配一点时间，另外的一些时间则可以分配一点，优先级越高的越容易被系统选择执行

**状态转换（五种状态）：**

**新建：创建后尚未启动的线程**

**运行：有可能正在执行，也有可能正在等待CPU为它分配执行时间**

**无限期等待：不会分配CPU执行时间，要等待被其他线程显式的唤醒。**

**没有设置Timeout参数的Object.wait()**

**没有设置Timeout参数的Thread.join()**

**LockSupport.park()方法**

**限期等待：也不会分配CPU执行时间，无需等待被其他线程唤醒，在一定时间之后他们会有系统自动唤醒。**

**Thread.sleep()**

**设置Timeout参数的Object.wait()**

**设置Timeout参数的Thread.join()**

**LockSupport.parkNanos();**

**LockSupport.parkUntil()**

**阻塞：**

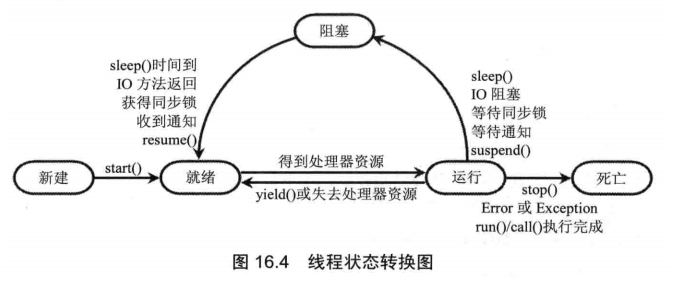
**结束：**

47.线程锁

<https://blog.csdn.net/zqz_zqz/article/details/70233767>

**CAS**

48.线程生命周期（状态）



49.Happens-before

在JVM中，如果一个操作执行的结果需要对另一个操作可见，那么这两个操作之间必须存在happens-before关系。

如果一个操作happens-before另一个操作，那么第一个操作的执行结果将对第二个操作可见，而且第一个操作的执行顺序排在第二个操作之前。

两个操作之间存在happens-before关系，并不意味着一定要按照happens-before原则制定的顺序来执行。如果重排序之后的执行结果与按照happens-before关系来执行的结果一致，那么这种重排序并不非法。

50.HashMap与ConcurrentHashMap

51.ThreadLocal

在每一个线程中对变量创建一个副本。

实际的通过ThreadLocal创建的副本是存储在每个线程自己的threadLocals中的；

为何threadLocals的类型ThreadLocalMap的键值为ThreadLocal对象，因为每个线程中可有多个threadLocal变量，就像上面代码中的longLocal和stringLocal；

在进行get之前，必须先set，否则会报空指针异常；

52.Math类中的取整方法

Math.ceil()向上取整

Math.floor()向下取整

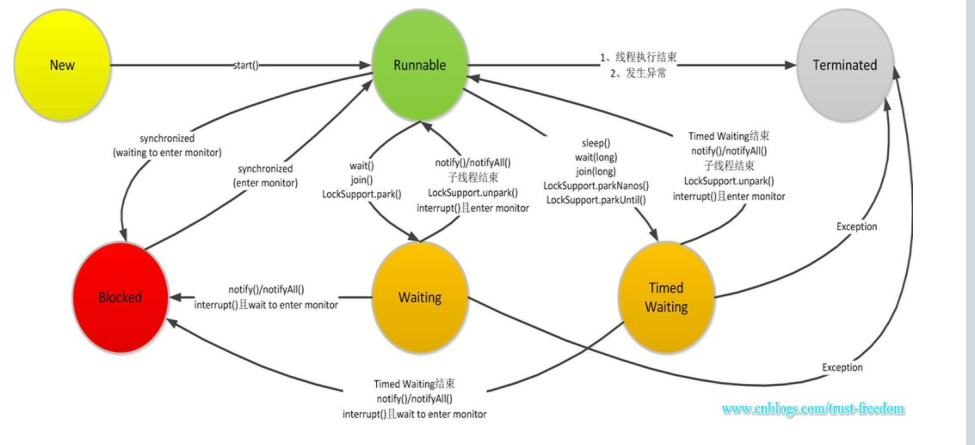
Math.round(x) = Math.floor(x+0.5);取最接近整数，如果遇到一样近，则取最大值

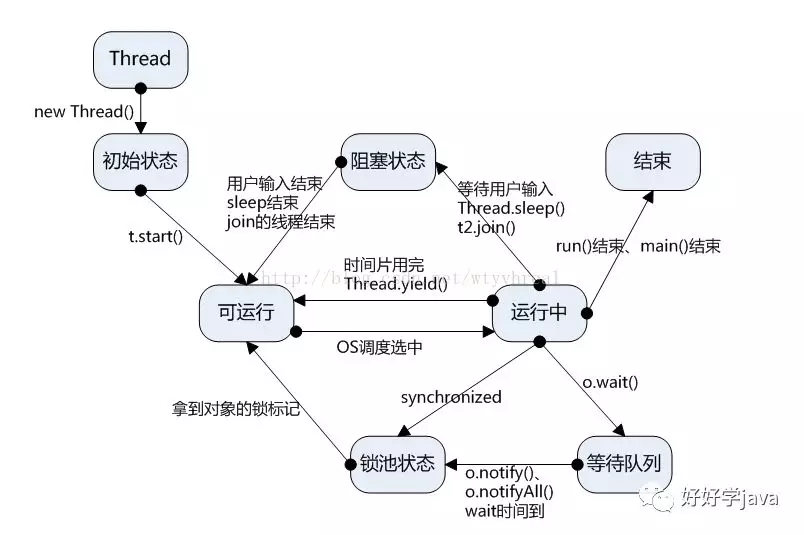
53.线程状态

五种状态（操作系统的说法）

New， Runnable， Running， Dead， Blocked

六种状态：





54.中断

**中断的协作特性所带来的一个好处是，它为安全地构造可取消活动提供更大的灵活性。我们很少希望一个活动立即停止；如果活动在正在进行更新的时候被取消，那么程序数据结构可能处于不一致状态。中断允许一个可取消活动来清理正在进行的工作，恢复不变量，通知其他活动它要被取消，然后才终止。**

当收到了中断请求后，如何结束该线程呢？

一种可行的方法是使用 **return 语句 而不是 break语句。。。。。哈哈。。。**

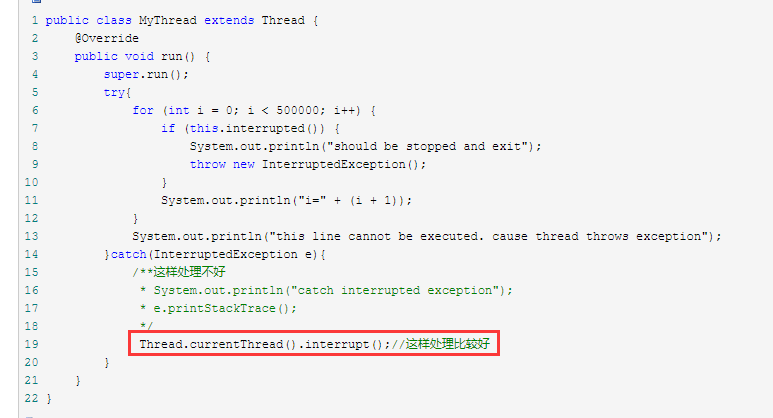
当然，一种更优雅的方式则是：抛出InterruptedException异常。

**interrupt()方法有两个作用，一个是将线程的中断状态置位(中断状态由false变成true)；另一个则是：让被中断的线程抛出InterruptedException异常。**

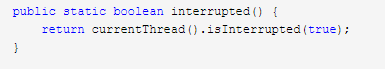
**对于那些阻塞方法(比如 wait() 和 sleep())而言，当另一个线程调用interrupt()中断该线程时，该线程会从阻塞状态退出并且抛出中断异常。这样，我们就可以捕捉到中断异常，并根据实际情况对该线程从阻塞方法中异常退出而进行一些处理。**

**比如说：线程A获得了锁进入了同步代码块中，但由于条件不足调用 wait() 方法阻塞了。这个时候，线程B执行 threadA.interrupt()请求中断线程A，此时线程A就会抛出InterruptedException，我们就可以在catch中捕获到这个异常并进行相应处理(比如进一步往上抛出)**

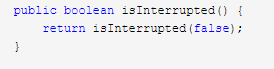
**处理方法：**



55.interrupted()和isInterrupted()方法



测试的是当前线程(current thread)的中断状态，且这个方法会清除中断状态。



isInterrupted()方法不会清除中断状态。

这两个方法都是调用的**isInterrupted(boolean ClearInterrupted)**,只不过一个带的参数是true，另一个带的参数是false。

**区别：**

1. 一个会清除中断标识位，另一个不会清除中断标识位。
2. interrupted()测试的是**当前的线程的中断状态**。而**isInterrupted()测试的是调用该方法的对象所表示的线程。**一个是静态方法（它测试的是当前线程的中断状态），一个是实例方法（它测试的是实例对象所表示的线程的中断状态）。

56.suspend()挂起resume()继续执行

这两个操作就好比播放器的暂停和恢复。

但这两个 API 是过期的，也就是不建议使用的。

不推荐使用 suspend() 去挂起线程的原因，是因为 suspend() 在导致线程暂停的同时，并不会去释放任何锁资源。其他线程都无法访问被它占用的锁。直到对应的线程执行 resume() 方法后，被挂起的线程才能继续，从而其它被阻塞在这个锁的线程才可以继续执行。

但是，如果 resume() 操作出现在 suspend() 之前执行，那么线程将一直处于挂起状态，同时一直占用锁，这就产生了死锁。而且，对于被挂起的线程，它的线程状态居然还是 Runnable。

57.栅栏和闭锁

栅栏类似于闭锁，它能阻塞一组线程直到某个事件发生。 栅栏与闭锁的关键区别在于，所有的线程必须同时到达栅栏位置，才能继续执行。可用于命令一组线程在同一个时刻开始执行某个任务，或者等待一组相关的操作结束。尤其适合计算并发执行某个任务的耗时。闭锁用于等待事件，而栅栏用于等待其他线程。所有线程必须同时到达栅栏位置才能继续执行下一步操作，且能够被重置以达到重复利用。而闭锁式一次性对象，一旦进入终止状态，就不能被重置。

闭锁(CountDownLatch)的await方法等待计数器为0；

栅栏(CyclicBarrier)的await方法等待所有的线程到达；

**闭锁场景：**

**场景：**有三个工人在为老板干活，这个老板有一个习惯，就是当三个工人把一天的活都干完了的时候，他就来检查所有工人所干的活

**栅栏场景：**

**场景：**接着上面的例子，还是这三个工人，不过这一次，这三个工人自由了，老板不用检查他们任务了，他们三个合作建桥，有三个桩，每人打一个，同时打完之后才能一起搭桥（搭桥需要三人一起合作）。也就是说三个人都打完桩之后才能继续工作。

58.数据库插入



59.信号量

用来控制同时访问某个特定资源的操作数量，或者同时执行某个指定操作的数量。计数信号量可以用来实现某种资源池，或者对容器施加边界。

60.FutureTask缓存计算过程

当一个计算的代价比较高，譬如比较耗时，或者耗资源，为了避免重复计算带来的浪费，当第一次计算后，通常会将结果缓存起来。比较常见的方式就是使用synchronized进行同步，但该方式带来的代价是被同步的代码只能被串行执行，如果有多个线程在排队对待计算结果，那么针对最后一个线程的计算时间可能比没有使用缓存的时间会更长。

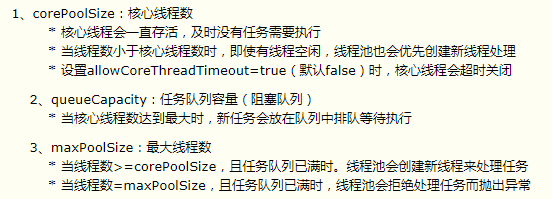
第二种方式是采用ConcurrentHashMap，但对于耗时比较长的计算过程来说，该方式也存在一个漏洞。如果在第一个线程正在计算的过程中，第二个线程开始获取结果，会发现缓存里没有缓存结果，因此第二个线程又启动了同样的计算，这样就导致重复计算，违背了缓存的初衷。计算过程越长，则出现这种重复计算的几率就会越大。

通过第二种方式的缺点分析，得知真正要缓存的应该是计算是否已被启动，而不是等待漫长的计算过程结束后，再缓存结果。一旦从缓存中得知某个计算过程已被其他线程启动，则当前线程不需要再重新启动计算，只需要阻塞等待计算结果的返回。FutureTask就是实现该功能的最佳选择。

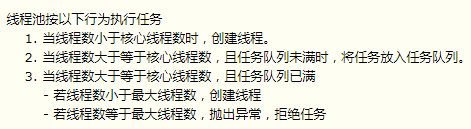
61.可重入锁

62.线程池ThreadPoolExecutor

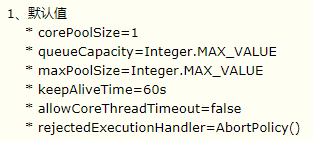
重要参数：



执行顺序：



如何设置参数：



静态方法：



63.利用反射修改私有属性的值

|  |
| --- |
| /\*\*  \* 测试实体类 属性都是私有的  \*/  public class Model {  private String a;  private String b;  @Override  public String toString() {  return "model [a=" + a + ", b=" + b + "]";  }    } |

|  |
| --- |
| //测试main方法  public static void main(String[] args) throws NoSuchFieldException, SecurityExcepton,  IllegalArgumentException, IllegalAccessException {           Model m = new Model();           Class**<?>** classType = m.getClass();           Field a = classType.getDeclaredField("a");           Field b = classType.getDeclaredField("b");           a.setAccessible(true); // 抑制Java对修饰符的检查           b.setAccessible(true); // 抑制Java对修饰符的检查           a.set(m, "测试aaa");           b.set(m, "测试bbb");           System.out.println(m);   } |

64. getDeclaredMethods和getMethods区别

getDeclaredMethod\*()获取的是类自身声明的所有方法，包含public、protected和private方法。getMethod\*()获取的是类的所有共有方法，这就包括自身的所有public方法，和从基类继承的、从接口实现的所有public方法。

65.线程和进程

* 进程是**资源分配的最小单位**，线程是**程序执行的最小单位。**
* 进程有自己的独立地址空间，每启动一个进程，系统就会为它分配地址空间，建立数据表来维护代码段、堆栈段和数据段，这种操作非常昂贵。而线程是共享进程中的数据的，使用相同的地址空间，因此CPU切换一个线程的花费远比进程要小很多，同时创建一个线程的开销也比进程要小很多。
* 线程之间的通信更方便，同一进程下的线程共享全局变量、静态变量等数据，而进程之间的通信需要以通信的方式（IPC)进行。不过如何处理好同步与互斥是编写多线程程序的难点。
* 但是多进程程序更健壮，多线程程序只要有一个线程死掉，整个进程也死掉了，而一个进程死掉并不会对另外一个进程造成影响，因为进程有自己独立的地址空间。

66.避免死锁

* 加锁顺序（线程按照一定的顺序加锁）
* 加锁时限（线程尝试获取锁的时候加上一定的时限，超过时限则放弃对该锁的请求，并释放自己占有的锁）
* 死锁检测

1. JavaIO和NIO
2. 网络socket

1. 网络编程基础知识

TCP/IP协议组：传输层；

IP地址：IPv4（32位二进制，192.168.199.1）；

端口：区分不同的应用程序，每个应用都有唯一的端口号，0-65535，（0-1023为系统保留）；

Socket：IP地址和端口号组成了所谓的Socket，Socket是网络上运行的程序之间双向通信链路的终结点。

常用协议的端口号：

http：80 ftp：21 telnet：23

java类

1. InetAddress
2. URL
3. Sockets：基于TCP的网络通信
4. Datagram:使用UDP协议

2. InetAddress

没有构造方法（Singlon模式）；

（1）获取本机的InetAddress实例

InetAddress address = InetAddress.getLocalHost();

Address.getHostName();

Address.getHostAddress();

Address.getAddress()字节数组形式的IP；

（2）根据主机名获取InetAddress

InnetAddress address2 = InetAddress.getByName(“”);

3. URL

URL：统一资源定位符；

URL由两部分组成：协议名称和资源名称，中间用冒号隔开；

在java.net包中，提供了URL类来表示URL；

URL imooc = new URL(“http://www.imooc.com”);(捕获异常)

URL url = new URL(imooc, “/index.html);

url.getProtocol(); 协议

url.getHost(); 主机

如果未指定端口号，则使用默认端口号，此时getPort()返回-1；

url.getPort(); 端口

url.getPath(); 文件路径

url.getFile() 文件名

**使用URL读取网页内容**

InputStream is = url.openStream() 通过URL的openStream方法获取URL对象所表示资源的字节输入流

InputStreamReader isr = new InputStreamReader(is, “utf-8”) 变为字符输入流

BufferReader br- new BufferReader(isr) 为字符输入流添加缓冲

4. TCP编程

客户端的Socket类

服务端的ServerSocket类

建立连接

开始通信：InputStream OutputStream

实现步骤；

1. 创建ServerSocket和Socket
2. 打开连接到Socket的输入输出流
3. 按照协议对Socket进行读写操作
4. 关闭输出输入、关闭Socket

服务器端：

1. 创建ServerSocket对象，绑定监听端口
2. 通过accept()方法监听客户端请求
3. 连接建立后，通过输入流读取客户端的请求信息
4. 通过输入流向客户端发送响应信息
5. 关闭相应资源

客户端：

1. 创建Socket对象，指明需要连接的服务器的地址和端口号
2. 连接建立后，通过输出流向服务器发送请求信息
3. 通过输入流获取服务器响应的信息
4. 关闭相关资源

多线程服务器实现服务器和多客户端之间的通信

基本步骤：

1. 服务器端创建ServerSocket，循环调用accept()等待客户端连接
2. 客户端创建一个socket并请求和服务器端连接；
3. 服务器端接收客户端请求，创建socket与该客户端建立专线连接；
4. 建立连接的两个socket在一个单独的线程上对话；
5. 服务器端继续等待新的连接；

5. UDP编程

UDP传输是不可靠、无序的，是以数据包为传输的载体；

先封装成数据报（Datagram）；

DatagramPacket：进行数据报包；

DatagramSocket：进行端到端通信的类；

6. 注意事项

多线程的优先级

是否关闭输出流和输入流

使用TCP通信传输对象

Socket编程传递文件

1. 数据库myspl（JDBC）

1.基础知识

数据库系统（database system）=数据库管理系统(DBMS)+数据库(database) + 管理员

DBS = DBMS + DB

JDBC包括两个包：java.sql, javax.sql

做JDBC的几个前提

导入java.sql包，有时导入javax.sql

导入jar驱动程序

简单案例

|  |
| --- |
| **import** java.sql.\*;  **public** **class** SimpleJDBC {  **public** **static** **void** main(String[] args) **throws** ClassNotFoundException, SQLException {  // **TODO** Auto-generated method stub  Class.*forName*("com.mysql.jdbc.Driver");  System.***out***.println("Driver loaded");  Connection connection = DriverManager.*getConnection*("jdbc:mysql://localhost/javabook", "scott", "tiger");  System.***out***.println("Database connected");  Statement statement = connection.createStatement();  ResultSet resultSet = statement.executeQuery("select \* from course");  **while** (resultSet.next())  System.***out***.println(resultSet.getString(1) + "\t" +  resultSet.getString(2) + "\t");  connection.close();  }  } |

1. 剑指offer读书笔记

1.

位运算：计算二进制中的数字：一个整数减去1，再与原整数做与运算，会把该整数最右边一个1变为0；

f(n) = f(n – 1) + f(n – 2) +……+f(1); f(n) = pow(2, n – 1)

用位与运算符代替求余运算符判断一个数是奇数还是偶数

4.

**全排列**

5.

链表用两个指针解决问题。

6.空间换取时间，时间换取空间

1. 数据结构和算法

1.深度优先用迭代或者递归。

2.广度优先用队列

3.拓扑排序（任务调度问题）

有向环存在，则不存在拓扑排序。

有向环的检测是排序的前提。

有向图的强连通性质。

4.最小生成树（无向图）

**切分定理**

**贪心算法**

**Prim算法：**延时实现、即时实现

Prim算法每一步都选择连接U和V-U的权值最小的边加入生成树。

**Kruskal算法：**

Kruskal算法每一步直接将权值最小的不成环的边加入生成树，我们借助并查集这一数据结构可以完美实现它。

5.最短路径（有向图）

Dijkstra算法：权重非负的最短路径。

无环加权有向图时：更快的算法；

Bellman-Ford：可以含有环，边的权重也可以是负值（Dijkstra不适合这种情况）。

6.字符串排序

低位优先的字符串排序

高位优先的字符串排序

三向字符串排序

7.单词查找树

单词查找树（R向单词查找树）

三向单词查找树

8.子字符串查找

KMP算法：两种编写方式：

1. 算法第四版：
2. 大众简单版：

Boyer-Moore算法：

Rabin-Karp指纹算法：

9.霍夫曼编码

前缀码

霍夫曼编码单词查找树

10.LZW压缩

11.后缀数组

12.最大流量问题，网络流问题

问题归约：

以下问题均可归约为线性规划问题：

**最大流量问题**

**最短路径问题**

**。。。。。。**

最大流最小切分定理

解法：

Ford-Fulkerson最大流算法：最大流-最小切分定理；

线性规划：单纯形法。

13.动态规划

**贪心和动归不是互斥的，而是包含的，贪心更快，但约束更强，适应范围更小。**

**动归和bfs的关系也是一样的。**

1. **背包问题**
2. **最长非降子序列的长度。**

第一种：排序，然后用LCS来解决：设序列X=<b1,b2,…,bn>是对序列L=<a1,a2,…,an>按递增排好序的序列。那么显然X与L的最长公共子序列即为L的最长递增子序列。这样就把求最长递增子序列的问题转化为求最长公共子序列问题LCS了。

第二种：时间复杂度O(N^2)的算法：

1. **最长公共子序列和最长公共字串**
2. **硬币找零算法**
3. **最长回文串算法**
4. **N皇后问题**
5. 记忆化的搜索就是动态规划
6. 动态规划算法通常基于一个递推公式及一个或多个初始状态。 当前子问题的解将由上一次子问题的解推出。使用动态规划来解题只需要多项式时间复杂度， 因此它比回溯法、暴力法等要快许多。
7. 分治与动态规划

**共同点**：二者都要求原问题具有最优子结构性质,都是将原问题分而治之,分解成若干个规模较小(小到很容易解决的程序)的子问题.然后将子问题的解合并,形成原问题的解.

**不同点**：分治法将分解后的子问题看成相互独立的，通过用递归来做。

动态规划将分解后的子问题理解为相互间有联系,有重叠部分，需要记忆，通常用迭代来做。

14.贪心算法

**参考《算法竞赛入门经典》**

贪心算法是指在对问题求解时，总是做出在当前看来是最好的选择。也就是说，不从整体最优上加以考虑，只做出在某种意义上的局部最优解。贪心算法不是对所有问题都能得到整体最优解，关键是贪心策略的选择，选择的贪心策略必须具备**无后效性**，即某个状态以前的过程不会影响以后的状态，只与当前状态有关。

1. java虚拟机

1.java程序执行流程

2.方法区

3.堆

4.java栈

5.GC

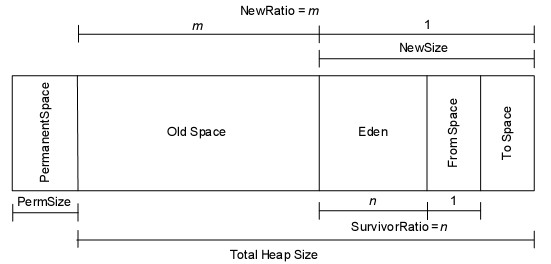
标记清除算法：

复制算法（新生代）：Eden空间和两个较小的survivor空间。Eden：Survivor=8:1

标记整理办法：（老年代）

**分代收集：**

**HotSpot虚拟机**



1. Serial收集器：复制算法的单线程收集器；
2. ParNew收集器：Serial收集器的多线程版本
3. Parallel收集器：新生代收集器，复制算法，并行的多线程收集器。
4. Serial Old收集器：Serial收集器的老年代版本，单线程，标记-清理算法。
5. Parallel Old收集器：Parallel收集器的老年代版本。使用多线程和标记-清理算法。
6. CMS收集器：最短回收停顿为目标的老年代收集器。标记-清理。
7. G1收集器：
8. JavaEE十三种技术

1. JDBC

JDBC定义了4种不同的驱动：

类型一：JDBC-ODBC桥

类型二：JDBC-native桥

类型三：

类型四：纯java驱动

2.EJB企业Java Beans

3.JSP

4.servlets

5.Java IDL/CORBA

6.java事务体系JTA/java事务服务JTS

7.JavaMail和JavaBeans激活架构

8.Java信使服务（JMS）

9.扩展标记语言（XML）

为什么要用web服务器 模拟一个web服务器

1. Tomcat

**Tomcat如何管理虚拟目录**

需求：不在webapps里的web怎么用tomcat管理。

配置虚拟目录：在server.xml的<host>节点间

<Context path =”/web2” docBase=”绝对路径”>

Path就是<http://xxx/web应用名/>资源名

Context的几个属性的说明:

daocBase

path

rebadable：如果设置为true，表示tomcat会自动更新，开销很大，建议在开发的时候，可以设置为true，但是一旦发布了，则应该设置为false

unpackWAR：举例说明

**在一台服务器中配置网站的域名：**

如何配置自己的主机名 ：

在实际中不可能使用http//localhost:8080/web应用/资源名 的方式去访问网站，实际中使用<http://www.sina.com>

看看ie浏览器访问一个web站点的流程：

浏览器

<http://localhost:8088/web3/hello2.html>

1. 解析主机名即：localhost, 先查找本机文件hosts，查询localhost对应的ip，如果找到了就不向域名服务器发送请求，直接访问查找到的ip。
2. 如果查不到，查询DNS服务器，如果没有查询到localhost的ip，直接报错。查到的话就访问查找到的ip。

实现的步骤如下：

(1)在host文件中添加：ip [www.wkk.com](http://www.wkk.com)

(2)在tomcat的server.xml文件添加主机名

<Host name=“网址” appBase=“路径”>

(3)在WEB-INF.xml中的首页面改变，如果端口也不想带，就可以把tomcat的端口设置为80。

(4)重启生效。

一个web服务器配置多个网站：

在host的文件中配置，

Catalina服务就是Tomcat服务

**Tomcat体系结构**

Tomcat

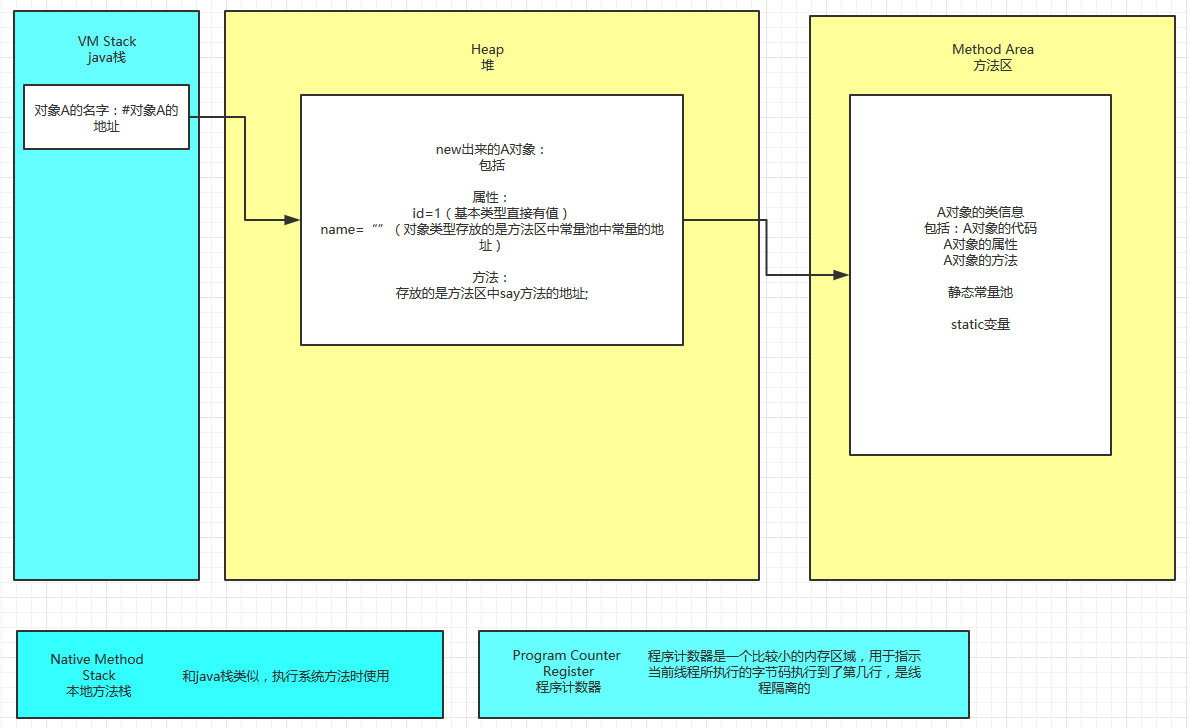
Service

Engine引擎

Host(可以有多个主机)

Context(应用)

1. Spring
2. 刷题小知识
3. JVM内存



1. JDBC Statement

1.Statement、PreparedStatement和CallableStatement都是接口(interface)。

2.Statement继承自Wrapper、PreparedStatement继承自Statement、CallableStatement继承自PreparedStatement。

3.

Statement接口提供了执行语句和获取结果的基本方法；

PreparedStatement接口添加了处理 IN 参数的方法；

CallableStatement接口添加了处理 OUT 参数的方法。

4.

a.Statement:

普通的不带参的查询SQL；支持批量更新,批量删除;

b.PreparedStatement:

可变参数的SQL,编译一次,执行多次,效率高;

安全性好，有效防止Sql注入等问题;

支持批量更新,批量删除;

c.CallableStatement:

继承自PreparedStatement,支持带参数的SQL操作;

支持调用存储过程,提供了对输出和输入/输出参数(INOUT)的支持;

Statement每次执行sql语句，数据库都要执行sql语句的编译 ，

最好用于仅执行一次查询并返回结果的情形，效率高于PreparedStatement。

PreparedStatement是预编译的，使用PreparedStatement有几个好处

1. 在执行可变参数的一条SQL时，PreparedStatement比Statement的效率高，因为DBMS预编译一条SQL当然会比多次编译一条SQL的效率要高。

2. 安全性好，有效防止Sql注入等问题。

3. 对于多次重复执行的语句，使用PreparedStament效率会更高一点，并且在这种情况下也比较适合使用batch；

4. 代码的可读性和可维护性。

1. Servlet和CGI

Servlet 与 CGI 的比较

和CGI程序一样，Servlet可以响应用户的指令(提交一个FORM等等)，也可以象CGI程序一样，收集用户表单的信息并给予动态反馈(简单的注册信息录入和检查错误)。

然而，Servlet的机制并不仅仅是这样简单的与用户表单进行交互。传统技术中，动态的网页建立和显示都是通过CGI来实现的，但是，有了Servlet,您可以大胆的放弃所有CGI(perl?php?甚至asp!)，利用Servlet代替CGI,进行程序编写。

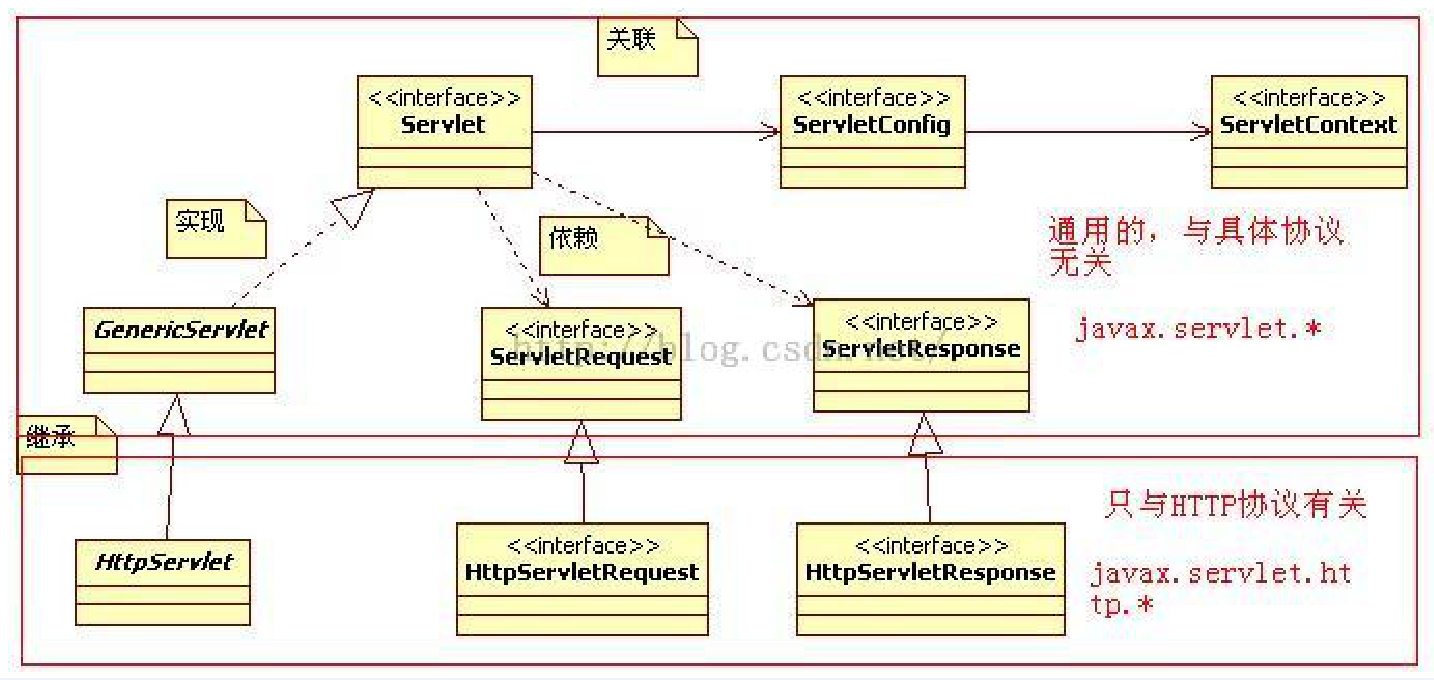
对比一：当用户浏览器发出一个Http/CGI的请求，或者说 调用一个CGI程序的时候，服务器端就要新启用一个进程 (而且是每次都要调用)，调用CGI程序越多(特别是访问量高的时候)，就要消耗系统越多的处理时间，只剩下越来越少的系统资源，对于用户来说，只能是漫长的等待服务器端的返回页面了，这对于电子商务激烈发展的今天来说，不能不说是一种技术上的遗憾。

而Servlet充分发挥了服务器端的资源并高效的利用。每次调用Servlet时并不是新启用一个进程 ，而是在一个Web服务器的进程敏感词享和分离线程，而线程最大的好处在于可以共享一个数据源，使系统资源被有效利用。

对比二：传统的CGI程序，不具备平台无关性特征，系统环境发生变化，CGI程序就要瘫痪，而Servlet具备Java的平台无关性，在系统开发过程中保持了系统的可扩展性、高效性。

对比三：传统技术中，一般大都为二层的系统架构，即Web服务器+数据库服务器，导致网站访问量大的时候，无法克服CGI程序与数据库建立连接时速度慢的瓶颈，从而死机、数据库死锁现象频繁发生。而我们的Servlet有连接池的概念，它可以利用多线程的优点，在系统缓存中事先建立好若干与数据库的连接，到时候若想和数据库打交道可以随时跟系统"要"一个连接即可，反应速度可想而知。

4 . Servlet service



doget/dopost与Http协议有关，是在 javax.servlet.http.HttpServlet 中实现的。

GenericServlet 抽象类 给出了设计 servlet 的一些骨架，定义了 servlet 生命周期，还有一些得到名字、配置、初始化参数的方法，其设计的是和应用层协议无关的。

5． Servlet生命周期

Servlet的生命周期分为5个阶段：加载、创建、初始化、处理客户请求、卸载。

(1)加载：容器通过类加载器使用servlet类对应的文件加载servlet

(2)创建：通过调用servlet构造函数创建一个servlet对象

(3)初始化：调用init方法初始化

(4)处理客户请求：每当有一个客户请求，容器会创建一个线程来处理客户请求

(5)卸载：调用destroy方法让servlet自己释放其占用的资源

实例化和初始化不同

init方法： 是在servlet实例创建时调用的方法，用于创建或打开任何与servlet相的资源和初始化servlet的状态，Servlet规范保证调用init方法前不会处理任何请求

service方法：是servlet真正处理客户端传过来的请求的方法，由web容器调用， 根据HTTP请求方法（GET、POST等），将请求分发到doGet、doPost等方法

destory方法：是在servlet实例被销毁时由web容器调用。Servlet规范确保在destroy方法调用之 前所有请求的处理均完成，需要覆盖destroy方法的情况：释放任何在init方法中 打开的与servlet相关的资源存储servlet的状态

6. 加载驱动程序

加载驱动方法

1.Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");

2. DriverManager.registerDriver(new com.mysql.jdbc.Driver());

3.System.setProperty("jdbc.drivers", "com.mysql.jdbc.Driver");

7. sleep和wait的区别

① 这两个方法来自不同的类分别是，sleep来自Thread类，和wait来自Object类。

sleep是Thread的静态类方法， 谁调用的谁去睡觉，即使在a线程里调用b的sleep方法，实际上还是a去睡觉， 要让b线程睡觉要在b的代码中调用sleep。

② 锁:  **最主要是sleep方法没有释放锁，而wait方法释放了锁**，使得其他线程可以使用同步控制块或者方法。

sleep不出让系统资源；wait是进入线程等待池等待，出让系统资源，其他线程可以占用CPU。一般wait不会加时间限制，因为如果wait线程的运行资源不够，再出来也没用，要等待其他线程调用notify/notifyAll唤醒等待池中的所有线程，才会进入就绪队列等待OS分配系统资源。sleep(milliseconds)可以用时间指定使它自动唤醒过来，如果时间不到只能调用interrupt()强行打断。

Thread.sleep(0)的作用是“触发操作系统立刻重新进行一次CPU竞争”。

③ 使用范围：wait，notify和notifyAll只能在同步控制方法或者同步控制块里面使用，而sleep可以在任何地方使用。

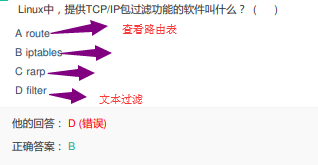
   synchronized(x){

      x.notify()

     //或者wait()

   }

8.



Arp协议：IP地址->MAC地址

Rarp协议：MAC地址->IP地址

9.Tomcat性能调优

**tomcat的3种运行模式**

**bio：**

默认的模式,性能非常低下,没有经过任何优化处理和支持.

**nio：**

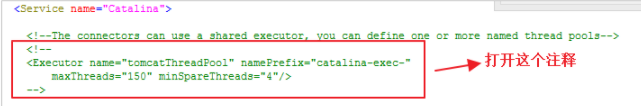
nio(new I/O)，是Java SE 1.4及后续版本提供的一种新的I/O操作方式(即java.nio包及其子包)。Java nio是一个基于缓冲区、并能提供非阻塞I/O操作的Java API，因此nio也被看成是non-blocking I/O的缩写。它拥有比传统I/O操作(bio)更好的并发运行性能。

**apr：**

安装起来最困难,但是从操作系统级别来解决异步的IO问题,大幅度的提高性能.

**执行器优化(线程池)**

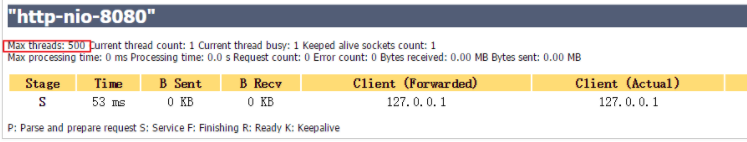
在tomcat中每一个用户请求都是一个线程，所以可以使用线程池提高性能。



在Connector中指定使用共享线程池



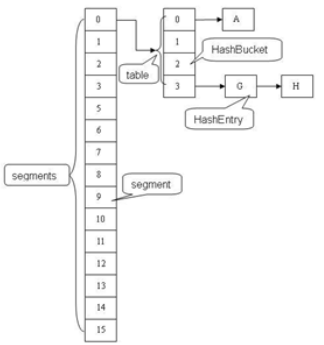
Tomcat控制台



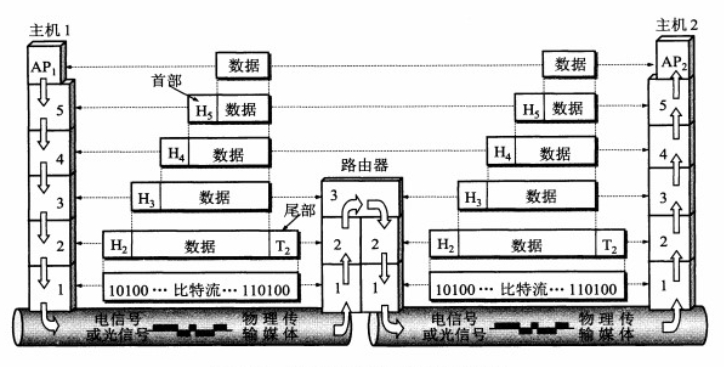
**禁用AJP连接器**

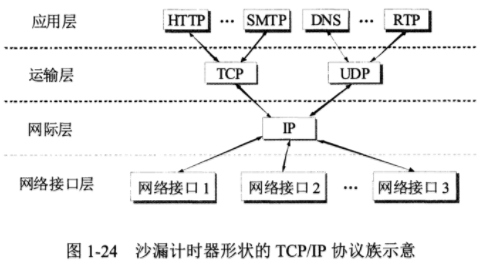
AJPv13协议是面向包的。WEB服务器和Servlet容器通过TCP连接来交互；为了节省SOCKET创建的昂贵代价，WEB服务器会尝试维护一个永久TCP连接到servlet容器，并且在多个请求和响应周期过程会重用连接。

10.ConcurrentHashMap

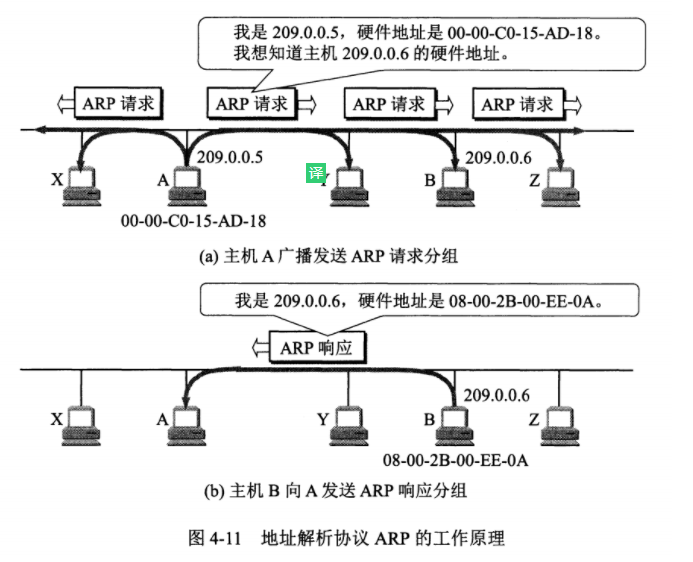


11.五层协议

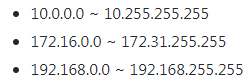




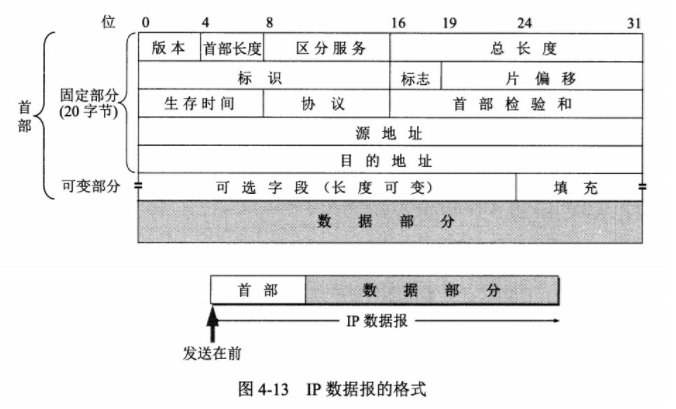
12.ARP(地址解析协议)



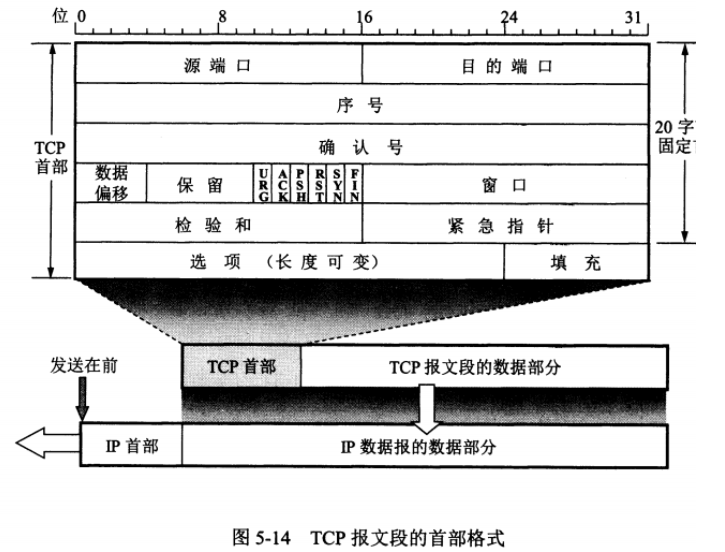
13.虚拟专用网VPN



14.IP数据报首部和TCP首部



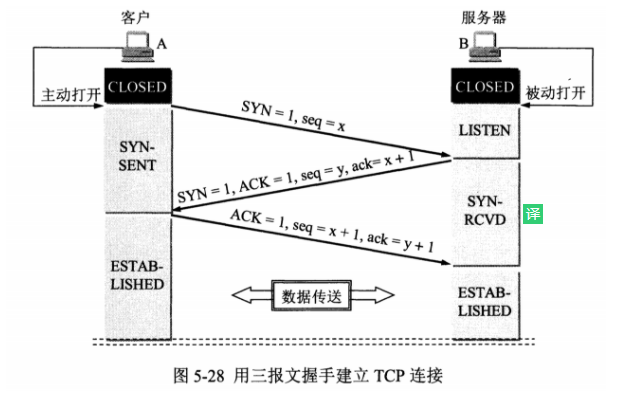
* **版本** : 有 4（IPv4）和 6（IPv6）两个值；
* **首部长度** : 占 4 位，因此最大值为 15。值为 1 表示的是 1 个 32 位字的长度，也就是 4 字节。因为首部固定长度为 20 字节，因此该值最小为 5。如果可选字段的长度不是 4 字节的整数倍，就用尾部的填充部分来填充。
* **区分服务** : 用来获得更好的服务，一般情况下不使用。
* **总长度** : 包括首部长度和数据部分长度。
* **生存时间** ：TTL，它的存在是为了防止无法交付的数据报在互联网中不断兜圈子。以路由器跳数为单位，当 TTL 为 0 时就丢弃数据报。
* **协议** ：指出携带的数据应该上交给哪个协议进行处理，例如 ICMP、TCP、UDP 等。
* **首部检验和** ：因为数据报每经过一个路由器，都要重新计算检验和，因此检验和不包含数据部分可以减少计算的工作量。
* **标识** : 在数据报长度过长从而发生分片的情况下，相同数据报的不同分片具有相同的标识符。
* **片偏移** : 和标识符一起，用于发生分片的情况。片偏移的单位为 8 字节。



序号 ：用于对字节流进行编号，例如序号为 301，表示第一个字节的编号为 301，如果携带的数据长度为 100 字节，那么下一个报文段的序号应为 401。

* **确认号** ：期望收到的下一个报文段的序号。例如 B 正确收到 A 发送来的一个报文段，序号为 501，携带的数据长度为 200 字节，因此 B 期望下一个报文段的序号为 701，B 发送给 A 的确认报文段中确认号就为 701。
* **数据偏移** ：指的是数据部分距离报文段起始处的偏移量，实际上指的是首部的长度。
* **确认 ACK** ：当 ACK=1 时确认号字段有效，否则无效。TCP 规定，在连接建立后所有传送的报文段都必须把 ACK 置 1。
* **同步 SYN** ：在连接建立时用来同步序号。当 SYN=1，ACK=0 时表示这是一个连接请求报文段。若对方同意建立连接，则响应报文中 SYN=1，ACK=1。
* **终止 FIN** ：用来释放一个连接，当 FIN=1 时，表示此报文段的发送方的数据已发送完毕，并要求释放连接。
* **窗口** ：窗口值作为接收方让发送方设置其发送窗口的依据。之所以要有这个限制，是因为接收方的数据缓存空间是有限的。

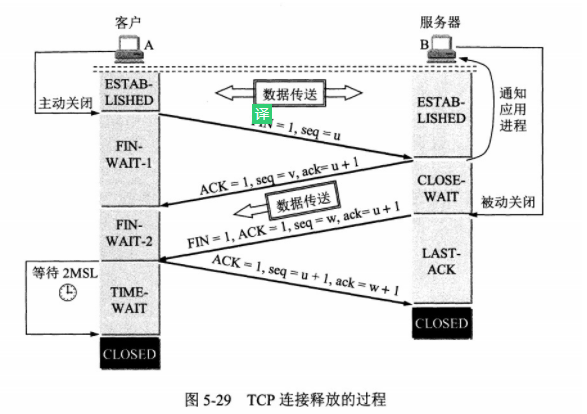
15.TCP的三次握手四次挥手



Ps：seq是数据包本身的序列号；ack是期望对方继续发送的那个数据包的序列号。

* 首先 B 处于 LISTEN（监听）状态，等待客户的连接请求。
* A 向 B 发送连接请求报文段，SYN=1，ACK=0，选择一个初始的序号 x。
* B 收到连接请求报文段，如果同意建立连接，则向 A 发送连接确认报文段，SYN=1，ACK=1，确认号为 x+1，同时也选择一个初始的序号 y。
* A 收到 B 的连接确认报文段后，还要向 B 发出确认，确认号为 y+1，序号为 x+1。
* B 收到 A 的确认后，连接建立。

**简单来说TCP三次握手后有个accept队列，进到这个队列才能从Listen变成accept，默认backlog 值是50，很容易就满了。满了之后握手第三步的时候server就忽略了client发过来的ack包（隔一段时间server重发握手第二步的syn+ack包给client），如果这个连接一直排不上队就异常了。**

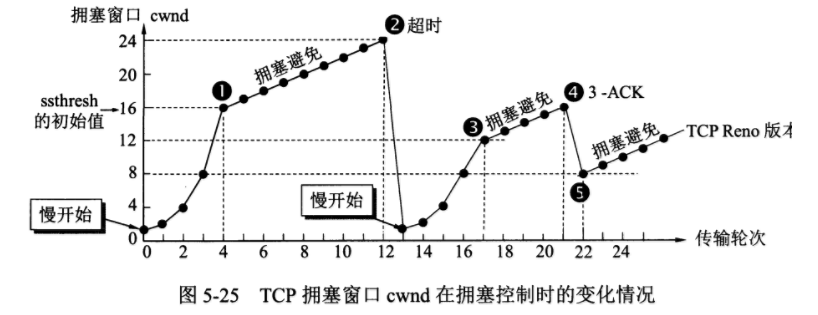


* A 发送连接释放报文段，FIN=1。
* B 收到之后发出确认，此时 TCP 属于半关闭状态，B 能向 A 发送数据但是 A 不能向 B 发送数据。
* 当 B 不再需要连接时，发送连接释放请求报文段，FIN=1。
* A 收到后发出确认，进入 TIME-WAIT 状态，等待 2 MSL 时间后释放连接。
* B 收到 A 的确认后释放连接。

16.TCP拥塞控制

TCP 主要通过四种算法来进行拥塞控制：**慢开始、拥塞避免、快重传、快恢复。**

发送方需要维护一个叫做拥塞窗口（cwnd）的状态变量，注意拥塞窗口与发送方窗口的区别：拥塞窗口只是一个**状态变量**，实际决定发送方能发送多少数据的是发送方窗口。



17.Web页面请求过程

1. **DHCP 配置主机信息**

* 假设主机最开始没有 IP 地址以及其它信息，那么就需要先使用 DHCP 来获取。
* 主机生成一个 DHCP 请求报文，并将这个报文放入具有目的端口 67 和源端口 68 的 UDP 报文段中。
* 该报文段则被放入在一个具有广播 IP 目的地址(255.255.255.255) 和源 IP 地址（0.0.0.0）的 IP 数据报中。
* 该数据报则被放置在 MAC 帧中，该帧具有目的地址 FF:FF:FF:FF:FF:FF，将广播到与交换机连接的所有设备。
* 连接在交换机的 DHCP 服务器收到广播帧之后，不断地向上分解得到 IP 数据报、UDP 报文段、DHCP 请求报文，之后生成 DHCP ACK 报文，该报文包含以下信息：IP 地址、DNS 服务器的 IP 地址、默认网关路由器的 IP 地址和子网掩码。该报文被放入 UDP 报文段中，UDP 报文段有被放入 IP 数据报中，最后放入 MAC 帧中。
* 该帧的目的地址是请求主机的 MAC 地址，因为交换机具有自学习能力，之前主机发送了广播帧之后就记录了 MAC 地址到其转发接口的交换表项，因此现在交换机就可以直接知道应该向哪个接口发送该帧。
* 主机收到该帧后，不断分解得到 DHCP 报文。之后就配置它的 IP 地址、子网掩码和 DNS 服务器的 IP 地址，并在其 IP 转发表中安装默认网关。

1. **ARP 解析 MAC 地址**

* 主机通过浏览器生成一个 TCP 套接字，套接字向 HTTP 服务器发送 HTTP 请求。为了生成该套接字，主机需要知道网站的域名对应的 IP 地址。
* 主机生成一个 DNS 查询报文，该报文具有 53 号端口，因为 DNS 服务器的端口号是 53。
* 该 DNS 查询报文被放入目的地址为 DNS 服务器 IP 地址的 IP 数据报中。
* 该 IP 数据报被放入一个以太网帧中，该帧将发送到网关路由器。
* DHCP 过程只知道网关路由器的 IP 地址，为了获取网关路由器的 MAC 地址，需要使用 ARP 协议。
* 主机生成一个包含目的地址为网关路由器 IP 地址的 ARP 查询报文，将该 ARP 查询报文放入一个具有广播目的地址（FF:FF:FF:FF:FF:FF）的以太网帧中，并向交换机发送该以太网帧，交换机将该帧转发给所有的连接设备，包括网关路由器。
* 网关路由器接收到该帧后，不断向上分解得到 ARP 报文，发现其中的 IP 地址与其接口的 IP 地址匹配，因此就发送一个 ARP 回答报文，包含了它的 MAC 地址，发回给主机。

1. **DNS 解析域名**

* 知道了网关路由器的 MAC 地址之后，就可以继续 DNS 的解析过程了。
* 网关路由器接收到包含 DNS 查询报文的以太网帧后，抽取出 IP 数据报，并根据转发表决定该 IP 数据报应该转发的路由器。
* 因为路由器具有内部网关协议（RIP、OSPF）和外部网关协议（BGP）这两种路由选择协议，因此路由表中已经配置了网关路由器到达 DNS 服务器的路由表项。
* 到达 DNS 服务器之后，DNS 服务器抽取出 DNS 查询报文，并在 DNS 数据库中查找待解析的域名。
* 找到 DNS 记录之后，发送 DNS 回答报文，将该回答报文放入 UDP 报文段中，然后放入 IP 数据报中，通过路由器反向转发回网关路由器，并经过以太网交换机到达主机。

1. **HTTP 请求页面**

* 有了 HTTP 服务器的 IP 地址之后，主机就能够生成 TCP 套接字，该套接字将用于向 Web 服务器发送 HTTP GET 报文。
* 在生成 TCP 套接字之前，必须先与 HTTP 服务器进行三次握手来建立连接。生成一个具有目的端口 80 的 TCP SYN 报文段，并向 HTTP 服务器发送该报文段。
* HTTP 服务器收到该报文段之后，生成 TCP SYN ACK 报文段，发回给主机。
* 连接建立之后，浏览器生成 HTTP GET 报文，并交付给 HTTP 服务器。
* HTTP 服务器从 TCP 套接字读取 HTTP GET 报文，生成一个 HTTP 响应报文，将 Web 页面内容放入报文主体中，发回给主机。
* 浏览器收到 HTTP 响应报文后，抽取出 Web 页面内容，之后进行渲染，显示 Web 页面。

18.位运算求二进制1或0的个数和求两个数的平均数

1的个数：

|  |
| --- |
| int num = 9999;  int count = 0;  while (num > 0) {  count++;  num = num & (num - 1);//使最后一个1及1之前变为0；  } |

0的个数

|  |
| --- |
| int num = 9999;  int count = 0;  while (num > 0) {  count++;  num = num || (num + 1);//使最后一个0及0之前变为1；  } |

一个数是不是2的N次方

|  |
| --- |
| int num = 9999;  if (num & (num – 1) == 0) 是2的N次方 |

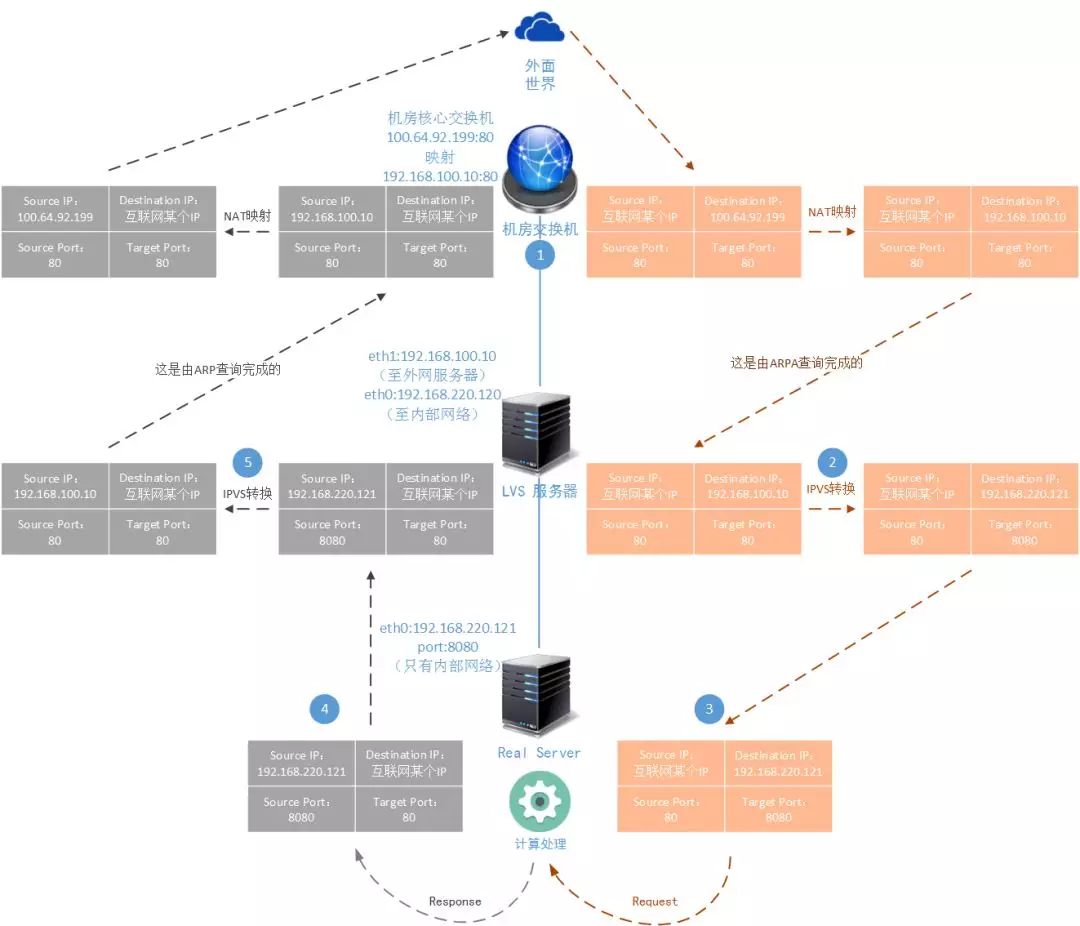
两个数的平均数

|  |
| --- |
| int x, y;  return (x & y) + ((x^y)>>1);//第一部分取相同位数的和的一半，后一部分不同位除2 |

19.LVS

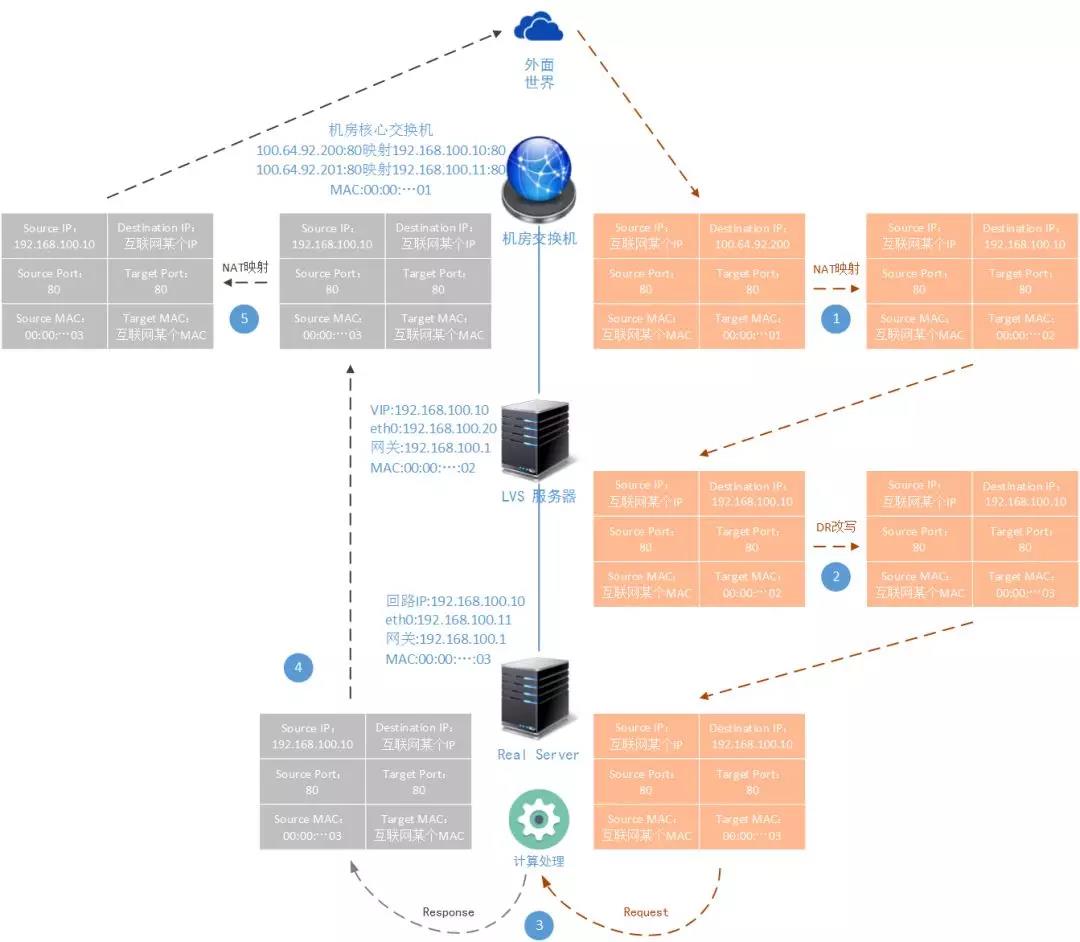
LVS-NAT模式

改写IP报文和TCP报文



LVS-DR工作模式

只改写链路层的报文封装



20.如何写一个不可变类

* 将类声明为final，所以它不能被继承
* 将所有的成员声明为私有的，这样就不允许直接访问这些成员
* 对变量不要提供setter方法
* 将所有可变的成员声明为final，这样只能对它们赋值一次
* 通过构造器初始化所有成员，进行深拷贝(deep copy)
* 在getter方法中，不要直接返回对象本身，而是克隆对象，并返回对象的拷贝

21.反射

* Class类
* 方法的反射
* 成员变量的反射
* 构造函数的反射
* Class类的动态加载类

22.闭锁与栅栏

* 闭锁：所有的线程中调用闭锁的方法await(),直到等到countDown调用使count为0，所有线程继续执行；
* 栅栏：建立一个栅栏对象，初始化一个线程数，线程中的栅栏调用await()，直到调用await的数量等于初始化的线程数，则所有await()的线程一起启动；

23.非阻塞无锁算法(CAS)

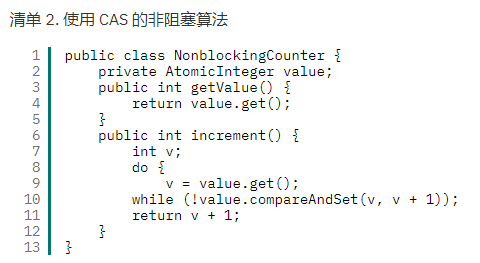
CAS比较与交换的伪代码可以表示为

|  |
| --- |
| do{  备份旧数据；  基于旧数据构造新数据；  }while(!CAS( 内存地址，备份的旧数据，新数据 )) |

就是指当两者进行比较时，如果相等，则证明共享数据没有被修改，替换成新值，然后继续往下运行；如果不相等，说明共享数据已经被修改，放弃已经所做的操作，然后重新执行刚才的操作。容易看出 CAS 操作是基于共享数据不会被修改的假设，采用了类似于数据库的 commit-retry 的模式。当同步冲突出现的机会很少时，这种假设能带来较大的性能提升

使用CAS的非阻塞算法

|  |
| --- |
| public class NonblockingCounter {  private AtomicInteger value;  public int getValue() {  return value.get();  }  public int increment() {  int v;  do {  v = value.get();  while (!value.compareAndSet(v, v + 1));  return v + 1;  }  } |



24.线程池的原理

假设一台服务器完成一项任务的时间为T

T1 创建线程的时间

T2 在线程中执行任务的时间，包括线程间同步所需时间

T3 线程销毁的时间

显然T ＝ T1＋T2＋T3。注意这是一个极度简化的假设。

可以看出T1，T3是多线程本身附加的开销，用户希望减少T1，T3所用的时间，从而减少T的时间。但一些线程的使用者并没有注意到这一点，所以在应用程序中频繁的创建或销毁线程，这导致T1和T3在T中占有非常大的比例。

线程池技术正是关注如何缩短或调整T1，T3时间的技术，从而提高服务器程序性能的。它把T1，T3分别安排在服务器程序的启动和结束的时间段或者一些空闲的时间段，这样在服务器程序处理客户请求时，不会有T1，T3的开销了，线程池不仅调整T1，T3产生的时间，而且它还显著减少了创建线程的数目

**一个任务通过execute(Runnable)方法被添加到线程池，任务必须是一个 Runnable类型的对象，任务的执行方法就是调用Runnable类型对象的run()方法。当一个任务通过execute(Runnable)方法欲添加到线程池时，会做一下几步：**

* 如果此时线程池中的数量小于corePoolSize，即使线程池中的线程都处于空闲状态，也要创建新的线程来处理被添加的任务。
* 如果此时线程池中的数量大于等于corePoolSize，但是缓冲队列 workQueue未满，那么任务被放入缓冲队列。
* 如果此时线程池中的数量大于corePoolSize，缓冲队列workQueue满，并且线程池中的数量小于maximumPoolSize，建新的线程来处理添加的任务。
* 如果此时线程池中的数量大于corePoolSize，缓冲队列workQueue满，并且线程池中的数量等于maximumPoolSize，那么通过 handler所指定的策略来处理此任务。也就是处理任务的优先级为：核心线程corePoolSize、任务队列workQueue、最大线程maximumPoolSize，如果三者都满了，使用handler处理被拒绝的任务。
* 当线程池中的线程数量大于corePoolSize时，如果某线程空闲时间超过keepAliveTime，线程将被终止。这样，线程池可以动态的调整池中的线程数**。**

|  |
| --- |
| public static ExecutorService newSingleThreadExecutor()  **(1, 1, BlockQueue(Integer.MAX\_VALUE))** |
| public static ExecutorService newCachedThreadPool()  **(0, Integer.MAX\_VALUE, 60s, SynchronousQueue(1个元素))** |
| public static ExecutorService newFixedThreadPool(int nThreads)  **(n, n, BlockQueue(Integer.MAX\_VALUE))** |

25.静态代理和动态代理

* Java静态代理：

代理类实现和目标类相同的接口，目标类对象为代理类的一个属性，实现接口的方法时，调用目标类对象的相应的方法并且，在方法前后增强处理；

实际上：代理对象的方法=增强处理+被代理对象的方法

* 动态代理：

JDK和CGLIB生成动态代理的区别：

JDK动态代理只能针对实现了接口的类生成代理，此代理对象和目标对象实现同一个接口，目标对象作为代理对象的一个属性，在具体接口实现中，可以在调用目标对象相应方法前后加上其他业务处理逻辑；

CGLIB是针对类实现的代理，主要是对指定的类生成一个子类，覆盖其中的方法

**SpringAOP应用场景：**

性能检测，访问控制，日志管理，事务

默认的策略是如果目标类实现接口，则使用JDK动态代理，如果目标对象没有实现接口，则默认会采用CGLIB代理；

26.悲观锁和乐观锁

* 悲观锁：假设会发生并发冲突
* 乐观锁：假设不会发生并发冲突

27.Synchronized和ReetrantLock的区别

* 在资源竞争不是很激烈的情况下，Synchronized的性能要优于ReetrantLock，但是在资源竞争很激烈的情况下，Synchronized的性能会下降几十倍，但是ReetrantLock的性能能维持常态；
* ReentrantLock获取锁定与三种方式：

1. lock(), 如果获取了锁立即返回，如果别的线程持有锁，当前线程则一直处于休眠状态，直到获取锁
2. tryLock(), 如果获取了锁立即返回true，如果别的线程正持有锁，立即返回false；
3. tryLock(long timeout,TimeUnit unit)， 如果获取了锁定立即返回true，如果别的线程正持有锁，会等待参数给定的时间，在等待的过程中，如果获取了锁定，就返回true，如果等待超时，返回false；
4. lockInterruptibly:如果获取了锁定立即返回，如果没有获取锁定，当前线程处于休眠状态，直到或者锁定，或者当前线程被别的线程中断

* synchronized是在JVM层面上实现的，不但可以通过一些监控工具监控synchronized的锁定，而且在代码执行时出现异常，JVM会自动释放锁定，但是使用Lock则不行，lock是通过代码实现的，要保证锁定一定会被释放，就必须将unLock()放到finally{}中

28.null+String

|  |
| --- |
| package com.test2;  public class TestString2 {  public static void main(String[] args) {  String str1 = null;  String str2 = "abc";  System.out.println((str1 + str2));  }  } |
| Output: **nullabc** |

29.ClassNotFoundException和NoClassDefFoundError的区别

* NoClassDefFoundError表示该类在编译阶段可以找到，但是在运行阶段找不到；是error,是不受检查的异常；是连接错误，发生在JVM加载流程的链接阶段，当解析引用的时候找不到对应的类就会抛出改异常；
* ClassNotFoundException一般发生在通过反射或者ClassLoader依据类名加载类时类不存在；是受检异常，需要进行异常捕获，否则会导致编译异常。发生在类加载流程的加载阶段；

[30.@Autowired](mailto:30.@Autowired)和@Resource的区别

* Autowired是根据类型装配的
* Resource是根据id装配的

31.SpringMVC运行原理

* 客户端请求提交到DispatcherServlet
* 由DispatcherServlet控制器查询处理器映射器(HandlerMapping)，找到并分发到指定的Controller中。
* 调用处理器适配器，通过处理器适配器调用Controller
* Controller处理业务逻辑，返回一个逻辑视图modelAndView
* 视图解析器解析逻辑视图，并将视图返回到客户端

32.一个Http请求

* DNS域名解析
* 发起TCP三次握手
* 建立TCP连接后发起http请求
* 服务器响应http请求，浏览器得到html代码
* 浏览器解析html代码，并请求html代码中的资源
* 浏览器对页面进行渲染呈现给用户

33.主键、外键、索引

* 定义：

1. 主键 -- 唯一标识一条记录，不能有重复的，不允许为空
2. 外键 -- 表的外键是另一表的主键 , 外键可以有重复的 , 可以是空值
3. 索引 -- 该字段没有重复值，但可以有一个空值

* 作用：

1. 主键 -- 用来保证数据完整性
2. 外键 -- 用来和其他表建立联系用的
3. 索引 -- 是提高查询排序的速度

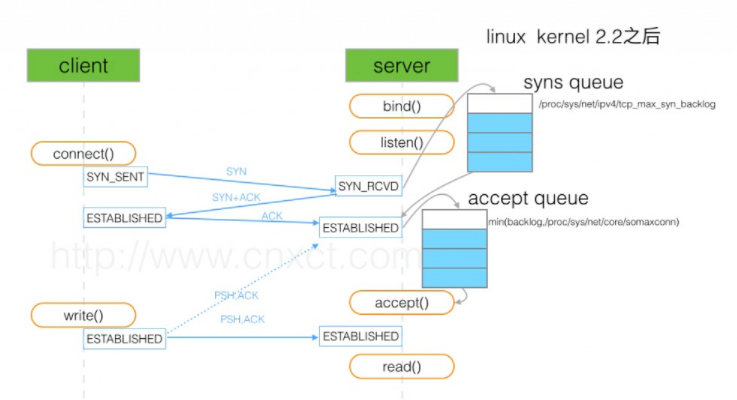
* 个数：

1. 主键 -- 主键只能有一个
2. 外键 -- 一个表可以有多个外键
3. 索引 -- 一个表可以有多个唯一索引

34.中间件

设计存储海量数据的存储系统：设计一个叫“中间层”的一个逻辑层，在这个层，将数据库的海量数据抓出来，做成缓存，运行在服务器的内存中，同理，当有新的数据到来，也先做成缓存，再想办法，持久化到数据库中，这是一个简单的思路。主要的步骤是负载均衡，将不同用户的请求分发到不同的处理节点上，然后先存入缓存，定时向主数据库更新数据。读写的过程采用类似乐观锁的机制，可以一直读（在写数据的时候也可以），但是每次读的时候会有个版本的标记，如果本次读的版本低于缓存的版本，会重新读数据，这样的情况并不多，可以忍受。

35.TCP握手的全连接队列和半连接队列



简单来说TCP三次握手后有个accept队列，进到这个队列才能从Listen变成accept，默认backlog 值是50，很容易就满了。满了之后握手第三步的时候server就忽略了client发过来的ack包（隔一段时间server重发握手第二步的syn+ack包给client），如果这个连接一直排不上队就异常了。

如上图所示，这里有两个队列：syns queue(半连接队列）；accept queue（全连接队列）

三次握手中，在第一步server收到client的syn后，把相关信息放到半连接队列中，同时回复syn+ack给client（第二步）；

比如**syn floods 攻击就是针对半连接队列的**，攻击方不停地建连接，但是建连接的时候只做第一步，第二步中攻击方收到server的syn+ack后故意扔掉什么也不做，导致server上这个队列满其它正常请求无法进来

第三步的时候server收到client的ack，如果这时全连接队列没满，那么从半连接队列拿出相关信息放入到全连接队列中，否则按tcp\_abort\_on\_overflow指示的执行。

这时如果全连接队列满了并且tcp\_abort\_on\_overflow是0的话，server过一段时间再次发送syn+ack给client（也就是重新走握手的第二步），如果client超时等待比较短，就很容易异常了。

在我们的os中retry 第二步的默认次数是2（centos默认是5次）：

net.ipv4.tcp\_synack\_retries = 2

36.常见负载均衡算法

* 静态负载均衡算法

1. 轮询
2. 比率
3. 优先权

* 动态负载均衡算法

1. 最小连接数
2. 最快响应速度
3. 观察方法
4. 预测法
5. 动态性能分配
6. 动态服务器补充
7. 服务质量
8. 服务类型
9. 规则模式

37.Error和Exception

相同点：都继承于Throwable

不同点：

Exception：

1. 可以被检查或者不可被检查
2. 由一个程序员导致的错误