|  |  |
| --- | --- |
| **构造方法摘要** | |
| [**Object**](file:///C:\Users\Innovation\Desktop\api\java\lang\Object.html#Object())() |  |

|  |  |
| --- | --- |
| **方法摘要** | |
| protected  [Object](file:///C:\Users\Innovation\Desktop\api\java\lang\Object.html) | [**clone**](file:///C:\Users\Innovation\Desktop\api\java\lang\Object.html#clone())()            创建并返回此对象的一个副本。 |
| boolean | [**equals**](file:///C:\Users\Innovation\Desktop\api\java\lang\Object.html#equals(java.lang.Object))([Object](file:///C:\Users\Innovation\Desktop\api\java\lang\Object.html) obj)            指示其他某个对象是否与此对象“相等”。 |
| protected  void | [**finalize**](file:///C:\Users\Innovation\Desktop\api\java\lang\Object.html#finalize())()            当垃圾回收器确定不存在对该对象的更多引用时，由对象的垃圾回收器调用此方法。 |
| [Class](file:///C:\Users\Innovation\Desktop\api\java\lang\Class.html)<?> | [**getClass**](file:///C:\Users\Innovation\Desktop\api\java\lang\Object.html#getClass())()            返回此 Object 的运行时类。 |
| int | [**hashCode**](file:///C:\Users\Innovation\Desktop\api\java\lang\Object.html#hashCode())()            返回该对象的哈希码值。 |
| void | [**notify**](file:///C:\Users\Innovation\Desktop\api\java\lang\Object.html#notify())()            唤醒在此对象监视器上等待的单个线程。 |
| void | [**notifyAll**](file:///C:\Users\Innovation\Desktop\api\java\lang\Object.html#notifyAll())()            唤醒在此对象监视器上等待的所有线程。 |
| [String](file:///C:\Users\Innovation\Desktop\api\java\lang\String.html) | [**toString**](file:///C:\Users\Innovation\Desktop\api\java\lang\Object.html#toString())()            返回该对象的字符串表示。 |
| void | [**wait**](file:///C:\Users\Innovation\Desktop\api\java\lang\Object.html#wait())()            在其他线程调用此对象的 [notify()](file:///C:\Users\Innovation\Desktop\api\java\lang\Object.html#notify()) 方法或 [notifyAll()](file:///C:\\Users\\Innovation\\Desktop\\api\\java\\lang\\Object.html" \l "notifyAll()) 方法前，导致当前线程等待。 |
| void | [**wait**](file:///C:\Users\Innovation\Desktop\api\java\lang\Object.html#wait(long))(long timeout)            在其他线程调用此对象的 [notify()](file:///C:\Users\Innovation\Desktop\api\java\lang\Object.html#notify()) 方法或 [notifyAll()](file:///C:\\Users\\Innovation\\Desktop\\api\\java\\lang\\Object.html" \l "notifyAll()) 方法，或者超过指定的时间量前，导致当前线程等待。 |
| void | [**wait**](file:///C:\Users\Innovation\Desktop\api\java\lang\Object.html#wait(long, int))(long timeout, int nanos)            在其他线程调用此对象的 [notify()](file:///C:\Users\Innovation\Desktop\api\java\lang\Object.html#notify()) 方法或 [notifyAll()](file:///C:\\Users\\Innovation\\Desktop\\api\\java\\lang\\Object.html" \l "notifyAll()) 方法，或者其他某个线程中断当前线程，或者已超过某个实际时间量前，导致当前线程等待。 |

|  |
| --- |
| **构造方法详细信息** |

**Object**

public **Object**()

|  |
| --- |
| **方法详细信息** |

**getClass**

public final [Class](file:///C:\Users\Innovation\Desktop\api\java\lang\Class.html)<?> **getClass**()

返回此 Object 的运行时类。返回的 Class 对象是由所表示类的 static synchronized 方法锁定的对象。

**实际结果类型是 Class<? extends |X|>，其中 |X| 表示清除表达式中的静态类型，该表达式调用 getClass。** 例如，以下代码片段中不需要强制转换：

Number n = 0;   
Class<? extends Number> c = n.getClass();

**返回：**

表示此对象运行时类的 Class 对象。

**另请参见：**

[The Java Language Specification, Third Edition (15.8.2 Class Literals)](http://java.sun.com/docs/books/jls/)

**hashCode**

public int **hashCode**()

返回该对象的哈希码值。支持此方法是为了提高哈希表（例如 java.util.Hashtable 提供的哈希表）的性能。

hashCode 的常规协定是：

* 在 Java 应用程序执行期间，在对同一对象多次调用 hashCode 方法时，必须一致地返回相同的整数，前提是将对象进行 equals 比较时所用的信息没有被修改。从某一应用程序的一次执行到同一应用程序的另一次执行，该整数无需保持一致。
* 如果根据 equals(Object) 方法，两个对象是相等的，那么对这两个对象中的每个对象调用 hashCode 方法都必须生成相同的整数结果。
* 如果根据 [equals(java.lang.Object)](file:///C:\Users\Innovation\Desktop\api\java\lang\Object.html#equals(java.lang.Object)) 方法，两个对象不相等，那么对这两个对象中的任一对象上调用 hashCode 方法*不* 要求一定生成不同的整数结果。但是，程序员应该意识到，为不相等的对象生成不同整数结果可以提高哈希表的性能。

实际上，由 Object 类定义的 hashCode 方法确实会针对不同的对象返回不同的整数。（这一般是通过将该对象的内部地址转换成一个整数来实现的，但是 JavaTM 编程语言不需要这种实现技巧。）

**返回：**

此对象的一个哈希码值。

**另请参见：**

[equals(java.lang.Object)](file:///C:\Users\Innovation\Desktop\api\java\lang\Object.html#equals(java.lang.Object)), [Hashtable](file:///C:\Users\Innovation\Desktop\api\java\util\Hashtable.html)

**equals**

public boolean **equals**([Object](file:///C:\\Users\\Innovation\\Desktop\\api\\java\\lang\\Object.html" \o "java.lang 中的类) obj)

指示其他某个对象是否与此对象“相等”。

equals 方法在非空对象引用上实现相等关系：

* *自反性*：对于任何非空引用值 x，x.equals(x) 都应返回 true。
* *对称性*：对于任何非空引用值 x 和 y，当且仅当 y.equals(x) 返回 true 时，x.equals(y) 才应返回 true。
* *传递性*：对于任何非空引用值 x、y 和 z，如果 x.equals(y) 返回 true，并且 y.equals(z) 返回 true，那么 x.equals(z) 应返回 true。
* *一致性*：对于任何非空引用值 x 和 y，多次调用 x.equals(y) 始终返回 true 或始终返回 false，前提是对象上 equals 比较中所用的信息没有被修改。
* 对于任何非空引用值 x，x.equals(null) 都应返回 false。

Object 类的 equals 方法实现对象上差别可能性最大的相等关系；即，对于任何非空引用值 x 和 y，当且仅当 x 和 y 引用同一个对象时，此方法才返回 true（x == y 具有值 true）。

注意：当此方法被重写时，通常有必要重写 hashCode 方法，以维护 hashCode 方法的常规协定，该协定声明相等对象必须具有相等的哈希码。

**参数：**

obj - 要与之比较的引用对象。

**返回：**

如果此对象与 obj 参数相同，则返回 true；否则返回 false。

**另请参见：**

[hashCode()](file:///C:\Users\Innovation\Desktop\api\java\lang\Object.html#hashCode()), [Hashtable](file:///C:\Users\Innovation\Desktop\api\java\util\Hashtable.html)

**clone**

protected [Object](file:///C:\Users\Innovation\Desktop\api\java\lang\Object.html) **clone**()

throws [CloneNotSupportedException](file:///C:\Users\Innovation\Desktop\api\java\lang\CloneNotSupportedException.html)

创建并返回此对象的一个副本。“副本”的准确含义可能依赖于对象的类。这样做的目的是，对于任何对象 x，表达式：

x.clone() != x

为 true，表达式：

x.clone().getClass() == x.getClass()

也为 true，但这些并非必须要满足的要求。一般情况下：

x.clone().equals(x)

为 true，但这并非必须要满足的要求。

按照惯例，返回的对象应该通过调用 super.clone 获得。如果一个类及其所有的超类（Object 除外）都遵守此约定，则 x.clone().getClass() == x.getClass()。

按照惯例，此方法返回的对象应该独立于该对象（正被复制的对象）。要获得此独立性，在 super.clone 返回对象之前，有必要对该对象的一个或多个字段进行修改。这通常意味着要复制包含正在被复制对象的内部“深层结构”的所有可变对象，并使用对副本的引用替换对这些对象的引用。如果一个类只包含基本字段或对不变对象的引用，那么通常不需要修改 super.clone 返回的对象中的字段。

Object 类的 clone 方法执行特定的复制操作。首先，如果此对象的类不能实现接口 Cloneable，则会抛出 CloneNotSupportedException。注意，所有的数组都被视为实现接口 Cloneable。否则，此方法会创建此对象的类的一个新实例，并像通过分配那样，严格使用此对象相应字段的内容初始化该对象的所有字段；这些字段的内容没有被自我复制。所以，此方法执行的是该对象的“浅表复制”，而不“深层复制”操作。

Object 类本身不实现接口 Cloneable，所以在类为 Object 的对象上调用 clone 方法将会导致在运行时抛出异常。

**返回：**

此实例的一个副本。

**抛出：**

[CloneNotSupportedException](file:///C:\Users\Innovation\Desktop\api\java\lang\CloneNotSupportedException.html) - 如果对象的类不支持 Cloneable 接口，则重写 clone 方法的子类也会抛出此异常，以指示无法复制某个实例。

**另请参见：**

[Cloneable](file:///C:\Users\Innovation\Desktop\api\java\lang\Cloneable.html)

**toString**

public [String](file:///C:\Users\Innovation\Desktop\api\java\lang\String.html) **toString**()

返回该对象的字符串表示。通常，toString 方法会返回一个“以文本方式表示”此对象的字符串。结果应是一个简明但易于读懂的信息表达式。建议所有子类都重写此方法。

Object 类的 toString 方法返回一个字符串，该字符串由类名（对象是该类的一个实例）、at 标记符“@”和此对象哈希码的无符号十六进制表示组成。换句话说，该方法返回一个字符串，它的值等于：

getClass().getName() + '@' + Integer.toHexString(hashCode())

**返回：**

该对象的字符串表示形式。

**notify**

public final void **notify**()

唤醒在此对象监视器上等待的单个线程。如果所有线程都在此对象上等待，则会选择唤醒其中一个线程。选择是任意性的，并在对实现做出决定时发生。线程通过调用其中一个 wait 方法，在对象的监视器上等待。

直到当前线程放弃此对象上的锁定，才能继续执行被唤醒的线程。被唤醒的线程将以常规方式与在该对象上主动同步的其他所有线程进行竞争；例如，唤醒的线程在作为锁定此对象的下一个线程方面没有可靠的特权或劣势。

此方法只应由作为此对象监视器的所有者的线程来调用。通过以下三种方法之一，线程可以成为此对象监视器的所有者：

* 通过执行此对象的同步实例方法。
* 通过执行在此对象上进行同步的 synchronized 语句的正文。
* 对于 Class 类型的对象，可以通过执行该类的同步静态方法。

一次只能有一个线程拥有对象的监视器。

**抛出：**

[IllegalMonitorStateException](file:///C:\Users\Innovation\Desktop\api\java\lang\IllegalMonitorStateException.html) - 如果当前线程不是此对象监视器的所有者。

**另请参见：**

[notifyAll()](file:///C:\Users\Innovation\Desktop\api\java\lang\Object.html#notifyAll()), [wait()](file:///C:\Users\Innovation\Desktop\api\java\lang\Object.html#wait())

**notifyAll**

public final void **notifyAll**()

唤醒在此对象监视器上等待的所有线程。线程通过调用其中一个 wait 方法，在对象的监视器上等待。

直到当前线程放弃此对象上的锁定，才能继续执行被唤醒的线程。被唤醒的线程将以常规方式与在该对象上主动同步的其他所有线程进行竞争；例如，唤醒的线程在作为锁定此对象的下一个线程方面没有可靠的特权或劣势。

此方法只应由作为此对象监视器的所有者的线程来调用。有关线程能够成为监视器所有者的方法的描述，请参阅 notify 方法。

**抛出：**

[IllegalMonitorStateException](file:///C:\Users\Innovation\Desktop\api\java\lang\IllegalMonitorStateException.html) - 如果当前线程不是此对象监视器的所有者。

**另请参见：**

[notify()](file:///C:\Users\Innovation\Desktop\api\java\lang\Object.html#notify()), [wait()](file:///C:\Users\Innovation\Desktop\api\java\lang\Object.html#wait())

**wait**

public final void **wait**(long timeout)

throws [InterruptedException](file:///C:\Users\Innovation\Desktop\api\java\lang\InterruptedException.html)

在其他线程调用此对象的 [notify()](file:///C:\Users\Innovation\Desktop\api\java\lang\Object.html#notify()) 方法或 [notifyAll()](file:///C:\\Users\\Innovation\\Desktop\\api\\java\\lang\\Object.html" \l "notifyAll()) 方法，或者超过指定的时间量前，导致当前线程等待。

当前线程必须拥有此对象监视器。

此方法导致当前线程（称之为 *T*）将其自身放置在对象的等待集中，然后放弃此对象上的所有同步要求。出于线程调度目的，在发生以下四种情况之一前，线程 *T* 被禁用，且处于休眠状态：

* 其他某个线程调用此对象的 notify 方法，并且线程 *T* 碰巧被任选为被唤醒的线程。
* 其他某个线程调用此对象的 notifyAll 方法。
* 其他某个线程[中断](file:///C:\Users\Innovation\Desktop\api\java\lang\Thread.html#interrupt())线程 *T*。
* 大约已经到达指定的实际时间。但是，如果 timeout 为零，则不考虑实际时间，在获得通知前该线程将一直等待。

然后，从对象的等待集中删除线程 *T*，并重新进行线程调度。然后，该线程以常规方式与其他线程竞争，以获得在该对象上同步的权利；一旦获得对该对象的控制权，该对象上的所有其同步声明都将被恢复到以前的状态，这就是调用 wait 方法时的情况。然后，线程 *T* 从 wait 方法的调用中返回。所以，从 wait 方法返回时，该对象和线程 T 的同步状态与调用 wait 方法时的情况完全相同。

在没有被通知、中断或超时的情况下，线程还可以唤醒一个所谓的*虚假唤醒* (spurious wakeup)。虽然这种情况在实践中很少发生，但是应用程序必须通过以下方式防止其发生，即对应该导致该线程被提醒的条件进行测试，如果不满足该条件，则继续等待。换句话说，等待应总是发生在循环中，如下面的示例：

synchronized (obj) {

while (<condition does not hold>)

obj.wait(timeout);

... // Perform action appropriate to condition

}

（有关这一主题的更多信息，请参阅 Doug Lea 撰写的 Concurrent Programming in Java (Second Edition) (Addison-Wesley, 2000) 中的第 3.2.3 节或 Joshua Bloch 撰写的 Effective Java Programming Language Guide (Addison-Wesley, 2001) 中的第 50 项。

如果当前线程在等待之前或在等待时被任何线程[中断](file:///C:\Users\Innovation\Desktop\api\java\lang\Thread.html#interrupt())，则会抛出 InterruptedException。在按上述形式恢复此对象的锁定状态时才会抛出此异常。

注意，由于 wait 方法将当前线程放入了对象的等待集中，所以它只能解除此对象的锁定；可以同步当前线程的任何其他对象在线程等待时仍处于锁定状态。

此方法只应由作为此对象监视器的所有者的线程来调用。有关线程能够成为监视器所有者的方法的描述，请参阅 notify 方法。

**参数：**

timeout - 要等待的最长时间（以毫秒为单位）。

**抛出：**

[IllegalArgumentException](file:///C:\Users\Innovation\Desktop\api\java\lang\IllegalArgumentException.html) - 如果超时值为负。

[IllegalMonitorStateException](file:///C:\Users\Innovation\Desktop\api\java\lang\IllegalMonitorStateException.html) - 如果当前线程不是此对象监视器的所有者。

[InterruptedException](file:///C:\Users\Innovation\Desktop\api\java\lang\InterruptedException.html) - 如果在当前线程等待通知之前或者正在等待通知时，任何线程中断了当前线程。在抛出此异常时，当前线程的*中断状态* 被清除。

**另请参见：**

[notify()](file:///C:\Users\Innovation\Desktop\api\java\lang\Object.html#notify()), [notifyAll()](file:///C:\\Users\\Innovation\\Desktop\\api\\java\\lang\\Object.html" \l "notifyAll())

**wait**

public final void **wait**(long timeout,

int nanos)

throws [InterruptedException](file:///C:\Users\Innovation\Desktop\api\java\lang\InterruptedException.html)

在其他线程调用此对象的 [notify()](file:///C:\Users\Innovation\Desktop\api\java\lang\Object.html#notify()) 方法或 [notifyAll()](file:///C:\\Users\\Innovation\\Desktop\\api\\java\\lang\\Object.html" \l "notifyAll()) 方法，或者其他某个线程中断当前线程，或者已超过某个实际时间量前，导致当前线程等待。

此方法类似于一个参数的 wait 方法，但它允许更好地控制在放弃之前等待通知的时间量。用毫微秒度量的实际时间量可以通过以下公式计算出来：

1000000\*timeout+nanos

在其他所有方面，此方法执行的操作与带有一个参数的 [wait(long)](file:///C:\Users\Innovation\Desktop\api\java\lang\Object.html#wait(long)) 方法相同。需要特别指出的是，wait(0, 0) 与 wait(0) 相同。

当前线程必须拥有此对象监视器。该线程发布对此监视器的所有权，并等待下面两个条件之一发生：

* 其他线程通过调用 notify 方法，或 notifyAll 方法通知在此对象的监视器上等待的线程醒来。
* timeout 毫秒值与 nanos 毫微秒参数值之和指定的超时时间已用完。

然后，该线程等到重新获得对监视器的所有权后才能继续执行。

对于某一个参数的版本，实现中断和虚假唤醒是有可能的，并且此方法应始终在循环中使用：

synchronized (obj) {

while (<condition does not hold>)

obj.wait(timeout, nanos);

... // Perform action appropriate to condition

}

此方法只应由作为此对象监视器的所有者的线程来调用。有关线程能够成为监视器所有者的方法的描述，请参阅 notify 方法。

**参数：**

timeout - 要等待的最长时间（以毫秒为单位）。

nanos - 额外时间（以毫微秒为单位，范围是 0-999999）。

**抛出：**

[IllegalArgumentException](file:///C:\Users\Innovation\Desktop\api\java\lang\IllegalArgumentException.html) - 如果超时值是负数，或者毫微秒值不在 0-999999 范围内。

[IllegalMonitorStateException](file:///C:\Users\Innovation\Desktop\api\java\lang\IllegalMonitorStateException.html) - 如果当前线程不是此对象监视器的所有者。

[InterruptedException](file:///C:\Users\Innovation\Desktop\api\java\lang\InterruptedException.html) - 如果在当前线程等待通知之前或者正在等待通知时，任何线程中断了当前线程。在抛出此异常时，当前线程的*中断状态* 被清除。

**wait**

public final void **wait**()

throws [InterruptedException](file:///C:\Users\Innovation\Desktop\api\java\lang\InterruptedException.html)

在其他线程调用此对象的 [notify()](file:///C:\Users\Innovation\Desktop\api\java\lang\Object.html#notify()) 方法或 [notifyAll()](file:///C:\\Users\\Innovation\\Desktop\\api\\java\\lang\\Object.html" \l "notifyAll()) 方法前，导致当前线程等待。换句话说，此方法的行为就好像它仅执行 wait(0) 调用一样。

当前线程必须拥有此对象监视器。该线程发布对此监视器的所有权并等待，直到其他线程通过调用 notify 方法，或 notifyAll 方法通知在此对象的监视器上等待的线程醒来。然后该线程将等到重新获得对监视器的所有权后才能继续执行。

对于某一个参数的版本，实现中断和虚假唤醒是可能的，而且此方法应始终在循环中使用：

synchronized (obj) {

while (<condition does not hold>)

obj.wait();

... // Perform action appropriate to condition

}

此方法只应由作为此对象监视器的所有者的线程来调用。有关线程能够成为监视器所有者的方法的描述，请参阅 notify 方法。

**抛出：**

[IllegalMonitorStateException](file:///C:\Users\Innovation\Desktop\api\java\lang\IllegalMonitorStateException.html) - 如果当前线程不是此对象监视器的所有者。

[InterruptedException](file:///C:\Users\Innovation\Desktop\api\java\lang\InterruptedException.html) - 如果在当前线程等待通知之前或者正在等待通知时，任何线程中断了当前线程。在抛出此异常时，当前线程的*中断状态* 被清除。

**另请参见：**

[notify()](file:///C:\Users\Innovation\Desktop\api\java\lang\Object.html#notify()), [notifyAll()](file:///C:\\Users\\Innovation\\Desktop\\api\\java\\lang\\Object.html" \l "notifyAll())

**finalize**

protected void **finalize**()

throws [Throwable](file:///C:\Users\Innovation\Desktop\api\java\lang\Throwable.html)

当垃圾回收器确定不存在对该对象的更多引用时，由对象的垃圾回收器调用此方法。子类重写 finalize 方法，以配置系统资源或执行其他清除。

finalize 的常规协定是：当 JavaTM 虚拟机已确定尚未终止的任何线程无法再通过任何方法访问此对象时，将调用此方法，除非由于准备终止的其他某个对象或类的终结操作执行了某个操作。finalize 方法可以采取任何操作，其中包括再次使此对象对其他线程可用；不过，finalize 的主要目的是在不可撤消地丢弃对象之前执行清除操作。例如，表示输入/输出连接的对象的 finalize 方法可执行显式 I/O 事务，以便在永久丢弃对象之前中断连接。

Object 类的 finalize 方法执行非特殊性操作；它仅执行一些常规返回。Object 的子类可以重写此定义。

Java 编程语言不保证哪个线程将调用某个给定对象的 finalize 方法。但可以保证在调用 finalize 时，调用 finalize 的线程将不会持有任何用户可见的同步锁定。如果 finalize 方法抛出未捕获的异常，那么该异常将被忽略，并且该对象的终结操作将终止。

在启用某个对象的 finalize 方法后，将不会执行进一步操作，直到 Java 虚拟机再次确定尚未终止的任何线程无法再通过任何方法访问此对象，其中包括由准备终止的其他对象或类执行的可能操作，在执行该操作时，对象可能被丢弃。

对于任何给定对象，Java 虚拟机最多只调用一次 finalize 方法。

finalize 方法抛出的任何异常都会导致此对象的终结操作停止，但可以通过其他方法忽略它。

**抛出：**

[Throwable](file:///C:\Users\Innovation\Desktop\api\java\lang\Throwable.html) - 此方法抛出的 Exception

版权所有 2008 Sun Microsystems, Inc. 保留所有权利。请遵守[GNU General Public License, version 2 only](http://openjdk.java.net/legal/gplv2+ce.html)。