# Object类

## 基本概念

所有对象的父类。

## 常用的方法

### （1）Boolean equals(Object obj)

//指示某个对象是否与此对象相等（判断对象的值是否相等）。

### （2）Class getClass()

//返回此对象的运行时类，返回动态时的某个类。

### （3）int hashCode()

//返回对象的哈希码值

### （4）void notify()

//唤醒在此对象监视器上等待的单个线程。

### （5）void notifyAll()

//唤醒在此对象监视器上等待的所有线程。

### （6）String toString()

//返回该对象的字符串表示。

### （7）void wait()

//在其他线程调用此对象的 notify() 方法或 notifyAll() 方法前，导致当前线程等待。

### （8）public final native void wait(long timeout) throws InterruptedException;

//在其他线程调用此对象的 notify() 方法或 notifyAll() 方法前，导致当前线程等待。

参数：等待的最大时间（单位毫秒）

# System类

## 方法

### public static Properties getProperties()

确定当前系统属性，首先，如果有安全管理器，则调用其checkPropertiesAccess方法时不带参数。 这可能会导致安全性异常。getProperty（String）方法使用的当前系统属性集将作为Properties对象返回。

# Character类

## 方法

### public static boolean isISOControl(int codePoint)

确定所引用的字符(Unicode代码点)是一家通过ISO控制字符。一个字符被认为是一个ISO控制字符，如果它的代码是在'u0000'到' u001F'或由'u009F'到'u007F'范围。

# Date类

## 基本概念

Date类用于表示日期和时间，它位于java.util包中。很多与日期时间有关的类都能够转换为long类型整数。

## 构造函数

### （1）Date()

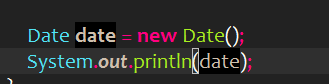
//无参的构造函数

### （2）Date(long date)

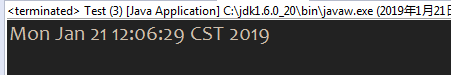
//传递一个长整型数字

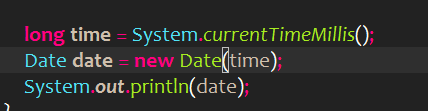
## 方法

## 代码示例

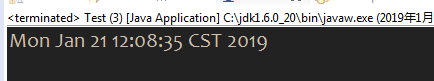


输出结果：





输出结果：



## 比较Date对象

### 方法

#### public boolean after(Date when)

用于测试日期对象是否在指定日期之后。

参数 when:是要比较的日期对象。

#### public boolean before(Date when)

用于测试日期对象是否在指定日期之前。

参数 when：是要比较的日期对象。

## 更改Date对象

getTime()和setTime()方法分别用于获取和设置Date对象的毫秒数。

## 日期的格式化输出

### 基本概念

使用String.format()方法可以通过特殊转换符作为参数实现对日期和时间的格式化，将Date类型转换为String类型。

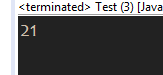
### 代码示例

Date now = new Date();

String time = String.format("%td",now);

System.out.println(time);

输出结果：



说明：输出这个月的第几天。

### 常见的日期时间转换符

%te 一个月中的某一天

%td 一个月中的第几天

%ta 指定语言环境中的星期几简称

%tb 指定语言环境中的月份简称

%tm 月份

%tc 包含全部日期和时间信息

%tY 4位年份

%td 获取当月的日期

### 时间格式化转换符

%tH 2位数字的24时值的小时

%tI 2位数字的12时值的小时

%tM 2位数字的分钟（00-59）

%tS 2位数字的秒数 （00-60）

%tL 3位数字的毫秒数（000-999）

%tp 语言环境中的上午或下午标记

### 常见的日期时间格式组合

%tF "年-月-日"格式（4位年份）

%tD "月/日/年"（2位年份）

%tc 全部日期和时间格式

%tr "时：分：秒PM(AM)"(12时制)

%tT "时：分：秒"格式（24时值）

%tR "时：分"格式（24时值）

# SimpleDateFormat类

## 基本概念

SimpleDateFormat 是一个与语言环境相关的方式来格式化和分析日期的具体类，它允许进行格式化（日期 -> 文本）、分析（文本 -> 日期）和规范化，这个类继承了DateFormat类。

## 构造函数

### public SimpleDateFormat()

### public SimpleDateFormat(String pattern)

//pattern：描述了日期和时间的格式

## 方法

### public final String format(Date date)

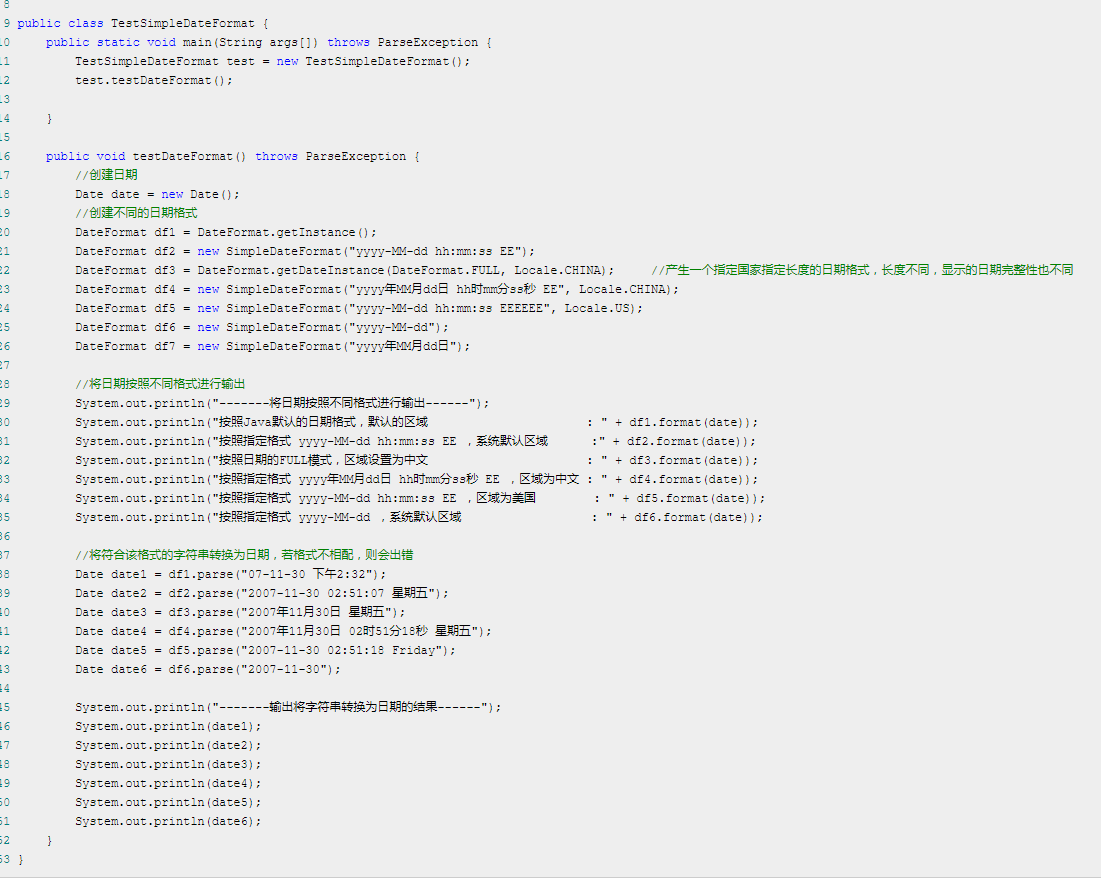
### public Date parse(String source)

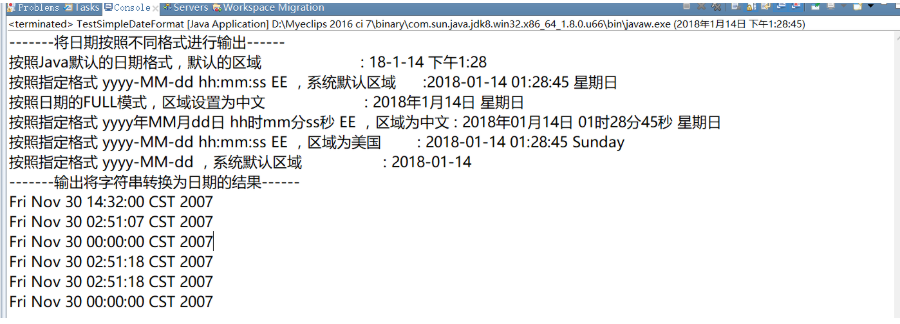
//参数：source 应该解析其开头的字符串

## 模式字母



## 代码示例





# Calendar类

## 基本概念

Java中提供了Calendar这个专门用于对日期进行操作的类

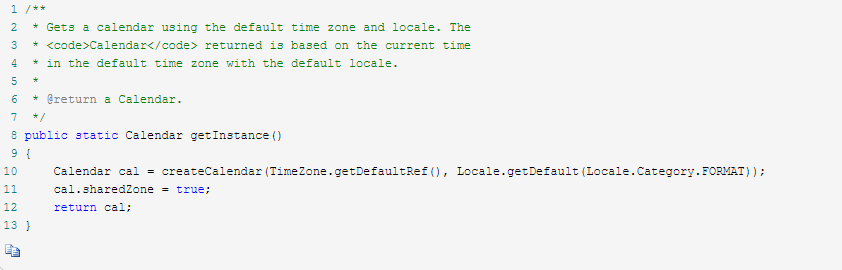
## 源码定义



该类被abstract所修饰，说明不能通过new的方式来获得实例，对此，Calendar提供了一个类方法getInstance，以获得此类型的一个通用的对象，getInstance方法返回一个Calendar对象（该对象为Calendar的子类对象），其日历字段已由当前日期和时间初始化：

Calendar rightNow = Calendar.getInstance();

为什么说返回的是Calendar的子类对象呢，因为每个国家地区都有自己的一套日历算法，比如西方国家的第一个星期大部分为星期日，而中国则为星期一，我们来看看getInstance方法获取实例的源码



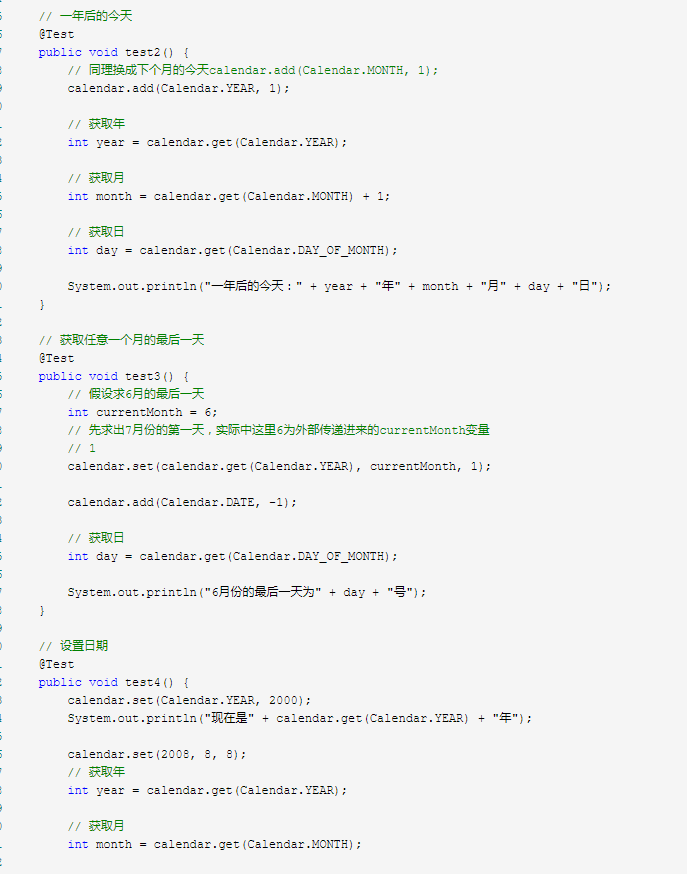
其中createCalendar方法就是根据不同国家地区返回对应的日期子类

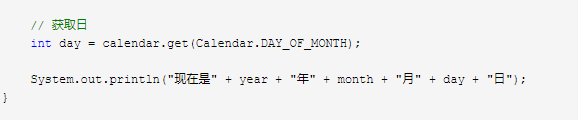
为了更加便捷的对日期进行操作，Calendar类对YEAR、MONTH、DAY\_OF\_MONTH、HOUR等日历字段之间的转换提供了一些方法，并为操作日历字段（例如获得下星期的日期）提供了一些方法。瞬间可用毫秒值来表示，它是距历元（即格林威治标准时间 1970 年 1 月 1 日的 00:00:00.000，格里高利历）的偏移量。

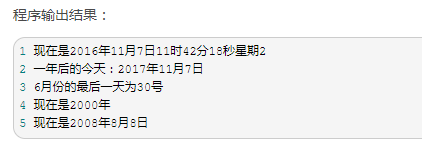
Calendar类中也有before，after，compareTo等方法，用法与[Date](http://www.cnblogs.com/huangminwen/p/5994927.html)类的类似，只是现在推荐用Calendar类操作日期。

## 代码示例









## 从date对象中获取月份

Calendar cal = Calendar.getInstance();

cal.setTime(new Date());

int month = cal.get(Calendar.MONTH);

System.out.println(month);

## 字段常量

public final static int YEAR = 1;

//年

public final static int MONTH = 2;

//月

public final static int WEEK\_OF\_YEAR = 3;

//一年中的某个星期

public final static int WEEK\_OF\_MONTH = 4;

//一月中的某个星期

public final static int DATE = 5;

//天

# GregorianCalendar类

## 基本概念

Calendar 类是一个抽象类，它为特定瞬间与一组诸如 YEAR、MONTH、DAY\_OF\_MONTH、HOUR 等日历字段之间的转换提供了一些方法，并为操作日历字段（例如获得下星期的日期）提供了一些方法。瞬间可用毫秒值来表示。GregorianCalendar 是 Calendar 的一个具体子类，提供了世界上大多数国家/地区使用的标准日历系统，注意月份的表示，一月是0，二月是1，以此类推，12月是11。因为大多数人习惯于使用单词而不是使用数字来表示月份，这样程序也许更易读，父类Calendar使用常量来表示月份：JANUARY, FEBRUARY,等

## 方法

### public final void setTime(Date date)

//用给定的日期设置此日历的时间

### public void add(int field, int amount)

//根据日历的规则将指定的(已签名的)时间添加到给定的日历字段。field表示日历字段，amount表示要添加到字段中的日期或时间的数量。

# Math类

## 基本概念

包含了所有用于数学运算的函数方法，这些方法都是静态的，所以每个方法只要使用"Math.数学方法"就可以调用。

## 主要方法

### （1）double abs(double a)

返回double值的绝对值

### （2）float abs(float a)

返回float值的绝对值

### （3）int abs(int a)

返回Int值的绝对值

### （4）long abs(long a)

返回long值的绝对值

### （5）double cbrt(double a)

返回double值的立方根。

### （6）double max(double a,double b)

返回两个double值中较大的一个。

### （7）float max(float a,float b)

返回两个float值中较大的一个。

### （8）long max(long a,long b)

返回两个long值中较大的一个。

### （9）int max(int a ,int b)

返回两个int值中较大的一个。

### （10）double random()

返回带正号的double值，该值大于等于0.0且小于1.0，返回一个随机数。

### （11）long round(double a)

返回最接近参数的long。

### （12）double ceil(double a)

返回大于等于a的最小整数。

### （13）double floor(double a)

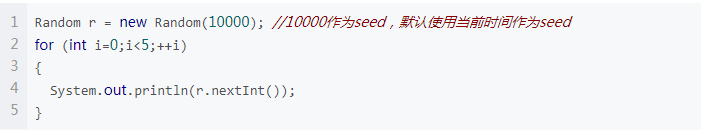
返回小于等于a的最大整数。

# Java伪随机数安全性

## 基本知识

在实际应用中往往使用伪随机数就足够了。这些数列看似是随机的数，实际上它们是通过一个固定的、可以重复的计算方法产生的。计算机产生的随机数有很长的周期性。它们不真正地随机，因为它们实际上是可以计算出来的，但是它们具有类似于随机数的统计特征。这样的发生器叫做伪随机数发生器。

伪随机数又有强弱之分。强伪随机数一般是指相对难以猜解的随机数，比如服务器占用的内存数量作为随机数；而弱伪随机数是指相对容易猜解的随机数呢，典型的例子是当前的时间戳等。C、C++、Java等程序语言和软件中都有对应的随机数生成函数或者类。以我们最常用的Java语言为例，强伪随机数RNG实现java.security.SecureRandom类，该类使用临时文件夹中大小，线程休眠时间等的值作为随机数种子；而弱伪随机数实现PRNG java.util.Random类，默认使用当前时间作为种子，并且采用线性同余法计算下一个随机数。



以上这段代码，无论你怎么跑都会打印出以下结果：



这是一个稳定的结果。这就是由于线性同余法带来的后果。那么，在我们的程序，如果使用Random类生成一个随机数，事实上很容易通过上一个产生的随机数来推断下一个随机数。

接下来，我们来分析一些常用的随机数应用场景，并且分析一下出错的原因。

很多账号体系都有一个找回密码功能，找回密码时给手机发送的验证码，给邮箱发送的验证码或者重置密码链接，以上种种都使用了伪随机数。

下面以某网站通过邮箱重置密码链接找回密码为例，通过页面操作之后，会在密保邮箱中发现以下重置密码的链接：

http://www.xxx.com/findpwd/setpwdfromemail?vc=2ABB36620A927644607491393EF0D5EF&u=xxx%40gmail.com

通过分析，我们发现，vc=2ABB36620A927644607491393EF0D5EF是一串md值，解开之后值是1339744000，是个unix时间戳！那么可以猜测，用户取回密码时产生一个时间戳与帐号绑定，那么修改这个用户密码只需知道这个时间戳就可以。况且，一般服务器时间都是跟标准时间同步，也就是说unix时间戳是可以预测的。我们可以通过暴力破解遍历当前标准时间+一个网络延迟来进行暴力破解。

伪随机数的应用里，验证码是另外一种典型应用。对于安全而言，验证码是一个非常有效的保护机制和人机区分机制，可以保障口令不被暴力破解，可以防止刷票，刷屏，重复提交恶意数据等。除了作为验证码之外，类似的应用还存在于一些活动的优惠券或者兑换码，如果兑换码设计不当，很容易被破解而破坏活动的公平性。

## 总结

1、不要使用时间戳作为随机数。

2、保证不同用处的随机数使用不同的种子

3、对于安全性要求高的随机数，使用强伪随机数产生

# Random类

## 基本概念

在Random类中提供了获取各种数据类型随机数的方法。

## 方法

### nextInt()

返回下一个伪随机数，它是此随机数生成器的序列中均匀分布的 int 值。（不是0到1的数）

### nextInt(int n)

返回一个伪随机数，它是取自此随机数生成器序列的、在 0（包括）和指定值（不包括）之间均匀分布的 int 值。(常用这个方法)

## 代码示例

生成8位数字的随机数

public static String generateUID(){

Random random = new Random();

String result="";

for(int i=0;i<8;i++){

result += (random.nextInt(10));

}

return result;

}

# SecureRandom类

## 基本概念

此类提供加密的强随机数生成器 (RNG)。许多实现都是伪随机数生成器 (PRNG) 形式，这意味着它们将使用确定的算法根据实际的随机种子生成伪随机序列。其他实现可以生成实际的随机数，而另一些实现则可能结合使用这两项技术。

与 Java Security 中其他基于算法的类一样，SecureRandom 也提供了与实现无关的算法，因此，调用方（应用程序代码）会请求特定的 RNG 算法并将它传回到该算法的 SecureRandom 对象中。如果需要，还可以通过特定的提供程序请求特定的算法。请参见 getInstance 方法。

因此，有以下两种请求 SecureRandom 对象的方法：仅指定算法名称，或者既指定算法名称又指定包提供程序。

（1）如果仅指定算法名称，如下所示：

SecureRandom random = SecureRandom.getInstance("SHA1PRNG");

系统将确定环境中是否有所请求的算法实现，是否有多个，是否有首选实现

（2）如果既指定了算法名称又指定了包提供程序，如下所示：

SecureRandom random = SecureRandom.getInstance("SHA1PRNG", "SUN");

系统将确定在所请求的包中是否有算法实现；如果没有，则抛出异常。

## 构造方法

### SecureRandom()

通过使用此构造方法，调用方可获取一个 SecureRandom 对象，该对象包含的实现来自具有 SecureRandom 实现的最高优先级已安装提供程序。

### [SecureRandom](http://gceclub.sun.com.cn/Java_Docs/html/zh_CN/api/java/security/SecureRandom.html#SecureRandom%28byte%5B%5D%29)(byte[] seed)

通过使用此构造方法，调用方可获取一个 SecureRandom 对象，该对象包含的实现来自具有 SecureRandom 实现的最高优先级已安装提供程序。

# DecimalFormat类

## 基本概念

DecimalFormat是NumberFormat的一个子类，用于格式化十进制数字，通过该类可以为要输出的数字加上单位或控制数字的精度。一般情况下可以实例化Decimal对象时传递数字格式，也可以通过DecimalFormat类的applyPattern()方法来实现数字格式化。

## DecimalFormat类中的特殊字符含义

0 代表阿拉伯数字

# 代表阿拉伯数字

. 代表小数分隔符或货币小数分隔符

- 代表负号

, 代表分组分隔符

E 代表分隔科学计数法中的尾数和指数。

% 代表百分数

\u2030 代表千分数

# ResourceBundle类

## 基本概念

这个类用于读取属性文件.properties的内容。

## 常用的方法

### public static final ResourceBundle getBundle(String baseName)

//返回当前指定的属性文件

### public final String getString(String key)

//返回当前属性文件中指定的字符串的值

## 代码示例

创建config.properties属性文件

name=jinshihui

age=24

String str = ResourceBundle.getBundle("config").getString("name");//jinshihui

String str1 = ResourceBundle.getBundle("config").getString("age");//24

# Arrays类

## 基本概念

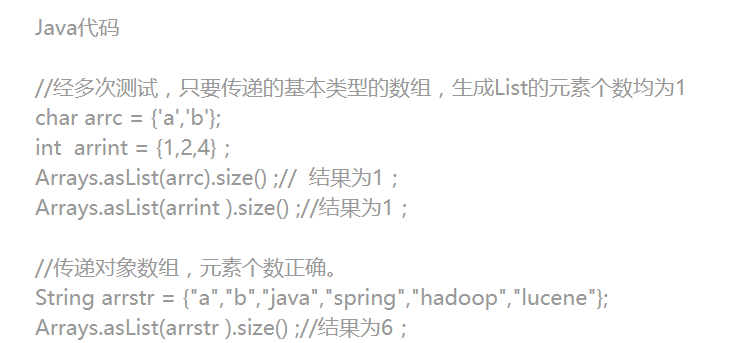
[java](http://lib.csdn.net/base/java).util.Arrays类能方便地操作数组，它提供的所有方法都是静态的。具有以下功能：

1. 给数组赋值，通过fill方法。
2. 给数组排序，通过sort方法，升序排序。
3. 比较数组，通过equals方法比较数组中元素的值是否相等。
4. 查找数组元素，通过binarySearch方法能对排序好的数组进行二分查找法操作。

## 方法

### static List asList()

//将数组转换为List集合，这个数组类型必须是引用类型，不能是8种基本数据类型。这个比较重要。



### int binarySearch(Object[] a,Object key)

使用二分搜索法来搜索指定数组，以获得指定对象。

参数：a 要搜索的数组，key要搜索的值。

返回：如果它包含在数组中，则返回搜索键的索引；否则返回 (-(插入点) - 1)。

示例：

**int**[] a= {1,2,3,4};

**int** num = Arrays.*binarySearch*(a, 3);

System.***out***.println(num);//2

### T[] copyOf(T[] original,int newLength)

//复制指定的数组，截取或用 null 填充（如有必要），以使副本具有指定的长度。

参数：original，要复制的数组，newlength，要返回的副本的长度。

返回：原数组的副本，截取或用 null 填充以获得指定的长度

示例：

**int**[] a= {1,2,3,4};

**int**[] num = Arrays.*copyOf*(a, 3);

System.***out***.println(Arrays.*toString*(num));

### boolean[] copyOfRange(boolean[] original,int from,int to)

//将指定数组的指定范围复制到一个新数组。该范围的初始索引 (from) 必须位于 0 和 original.length（包括）之间

参数：original,要复制的数组，from 要复制的范围的初始索引（包括），to，要复制的最后索引（不包括））（此索引可能位于数组范围之外）

返回：包含取自原数组指定范围的新数组，截取或用 false 元素填充以获得所需长度。

示例：

**int**[] a = {1,2,3,4};

**int**[] b = Arrays.*copyOfRange*(a, 0,3);

System.***out***.println(Arrays.*toString*(b));

### boolean deepEquals(Object[] a,Object[]b)

//如果两个指定数组彼此是深层相等 的，则返回 true。与 [equals(Object[],Object[])](mk:@MSITStore:C:\Users\jinshihui\Desktop\我的编程知识总结\Java\JDK_API_1_6_zh_CN.CHM::/java/util/Arrays.html#equals(java.lang.Object[], java.lang.Object[])) 方法不同，此方法适用于任意深度的嵌套数组。

### boolean equals(Object[] a,Object[]b)

//如果两个指定的 Objects 数组彼此相等，则返回 true。如果两个数组包含相同数量的元素，并且两个数组中的所有相应元素对都是相等的，则认为这两个数组是相等的。

## 总结

1、deepEquals用于判定两个指定数组彼此是否深层相等，此方法适用于任意深度的嵌套数组。

 2、equals用于判定两个数组是否相等，如果两个数组以相同顺序包含相同元素，则返回true，否则返回false。

3、如果两个数组使用equals返回true，则使用deepEquals也返回true，也就是说在比较的两个数组均为一维数组的前提下，equals和deepEquals的比较结果没有差别。

4、如果要比较多维数组，则需要使用deepEquals方法。

# Comparator接口

## 基本概念

该接口代表一个比较器，比较器具有可比性，javase数组工具类和集合工具类中提供的sort方法就是使用Comparator接口来处理排序的，下面是javase一些使用到Comparator接口的地方：



对任意类型集合对象进行整体排序，排序时将此接口的实现传递给Collections.sort方法或者Arrays.sort方法排序。

## 使用场景

1. 排序，需要比较两个对象谁排在前谁排在后（排序也可以让类实现Comparable接口，实现后该类的实例也具有排序能力）

2、分组，需要比较两个对象是否属于同一组

## 方法

### int compare(T o1, T o2);

参数：

o1 第一个比较的对象

o2 第二个比较的对象

返回：正数，零，负数各代表大于，等于，小于。

### boolean equals(Object obj);

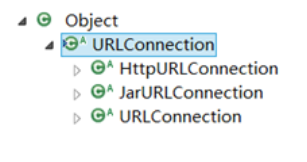
//比较

# URLConnection类

## 基本概念

这是一个抽象类，表示指向URL指定资源的活动连接，这个类在java.net包下，URLConnection类本身依赖于Socket类实现网络连接。一般认为，URLConnection类提供了比Socket类更易于使用、更高级的网络连接抽象。

URLConnection继承树：



URLConnection类声明为抽象类，除了connect()方法，其他方法都已经实现，URLConnection的三个实现子类都位于sun.net.www包下。

## 构造方法

### protected URLConnection(URL url)

//构造一个指定的URL连接。

## 方法

### abstract public void connect() throws IOException;

//打开到此 URL 引用的资源的通信链接（如果尚未建立这样的连接）

当派生URLConnection子类时，通常会覆盖URLConnection的其他方法，使其返回有意义的值。connect()方法由子类实现本地与服务器的连接方式。一般使用TCP socket，但也可以使用其他机制来建立。

### String getContentType()

//返回 Content-type 头字段的值，即数据的MIME内容类型。若类型不可用，则返回null。除了HTTP协议，极少协议会使用MIME首部。若内容类型是文本。则Content-type首部可能会包含一个标识内容编码方式的字符集。

例：Content-type:text/html; charset=UTF-8

### int getContentLength()

//返回 Content-length 头字段的值，该方法获取内容的字节数。许多服务器只有在传输二进制文件才发送Content-length首部，在传输文本文件时并不发送。若没有该首部，则返回-1。若需要知道读取的具体字节数，或需要预先知道创建足够大的缓冲区来保存数据时，可以使用该方法。

### String getContentEncoding()

//返回 Content-encoding 头字段的值，获取内容的编码方式。若内容无编码，则该方法返回null. 注意：Content-encoding(内容编码)与字符编码不同。字符编码方式由Content-type首部或稳定内容的信息确定，它指出如何使用字节指定字符。内容编码方式则指出字节如何编码其他字节。

### String getHeaderField(String name)

//返回指定头字段的值

### Map<String,List<String>> getHeaderFields()

//返回头字段不可修改的Map。

## 配置连接

下面都是属性

### protected  URL url

该字段指定URLConnection连接的URL。构造函数会在创建URLConnection时设置，此后就不能再改变。

### protected  boolean connected

当连接打开时，该字段为true，若连接关闭，则为false。没有直接读取或改变connected值的方法。任何导致URLConnection连接的方法都会将此变量设为true，包括connect()，getInputStream()，和getOutputStream()。任何导致URLConnection断开的方法都会将此字段设为false。java.net.URLConnection没有断开方法，但一些子类，如java.net.HttpURLConnection中存在这样的方法。

### protected  boolean allowUserInteraction

该字段表示是否允许与用户交互，默认为false，即不允许交互。该值只能在URLConnection连接前设置。例：Web浏览器可能需要访问用户名和口令。

以上字段定义了客户端如何向服务器作出请求。这些字段只能在URLConnection连接前修改，否则抛出IllegalStateException异常。

## 关于超时

Java 1.5增加了如下4个方法，允许查询和修改连接的超时值。即底层socket在抛出SocketTimeoutException异常前等待远端响应时间。

### int getReadTimeout()

//获取超时值，单位为毫秒

### void setReadTimeout(int timeout)

//设置连接的超时值，单位为毫秒。

### int getConnectTimeout()

//获取连接超时的时间

### void setConnectTimeout(int timeout)

//设置连接超时的时间

这四个方法将0解释为永不超时。如果超时值为负数。两个设置方法都会抛出IllegalArgumentException异常。

## 配置HTTP首部

客户端向服务器发送请求，不仅会发送请求行，还发送首部。Web服务器可以根据此信息向不同客户端提供不同的页面，获取和设置cookie，通过口令设置用户等等。通过在客户端发送和服务器响应的首部中放置不同的字段，就可以完成这些工作。

每个URLConnection的具体子类都在首部中设置一些默认的键值对（实际上只有HttpURLConnection这样做，因为HTTP是唯一以这种方式使用首部的主要协议。）通过以下方法可以设置URLConnection在打开连接增加HTTP首部信息。

### void setRequestProperty(String key, String value)

//此方法只在连接打开之前，否则将会抛出IllegalStateException异常。但该方法仅支持对一个首部设置一个值。HTTP允许一个属性有多个值。各个值之间用分号隔开。例：cookie:username=sjq; session=hdfu23asdjf901j;

### void addRequestProperty(String key, String value)

//该方法用于添加某个指定首部字段的值。

### String getRequestProperty(String key)

//获取首部字段的值

### Map<String,List<String>> getRequestProperties()

//用于获取URLConneciton所拥有HTTP首部。

# HttpURLConnection类

## 基本概念

一个支持HTTP特定功能的URLConnection，使用这个类遵循一下模式：

1、通过调用URL.openConnection()来获得一个新的HttpURLConnection对象，并且将其结果强制转换为HttpURLConnection。

2、准备请求。一个请求主要的参数是它的URI。请求头可能也包含元数据，例如证书，首选数据类型和会话cookies。

3、可以选择性的上传一个请求体。HttpURLConnection实例必须设置setDoOutput(true)，如果它包含一个请求体。通过将数据写入一个由getOutStream()返回的输出流来传输数据。

4、读取响应。响应头通常包含元数据例如响应体的内容类型和长度，修改日期和会话cookies。响应体可以被由getInputStream返回的输入流读取。如果响应没有响应体，则该方法会返回一个空的流。

5、关闭连接。一旦一个响应体已经被阅读后，HttpURLConnection 对象应该通过调用disconnect()关闭。断开连接会释放被一个connection占有的资源，这样它们就能被关闭或再次使用。

# BASE64Encoder

## 基本概念

这个类是用于base64加密的。

## 代码示例

代码如下：

String str = "jinshihui";

BASE64Encoder encoder = new BASE64Encoder();

str = encoder.encode(str.getBytes("UTF-8"));

System.out.println(str);

# BASE64Decoder

## 基本概念

这个类是用于base64解码的。

## 代码示例

代码如下：

BASE64Decoder decoder = new BASE64Decoder();

byte[] bytes = decoder.decodeBuffer(str);

System.out.println(new String(bytes,"UTF-8"));

## 说明

说明：从JDK 1.8开始，就提供了java.util.Base64.Decoder和java.util.Base64.Encoder的JDK公共API，可代替sun.misc.BASE64Decoder和sun.misc.BASE64Encoder的JDK内部API，代码如下：

//这是base64位编码

String str1= "jinshihui";

Encoder encoder2 = java.util.Base64.getEncoder();

String str2 = encoder2.encodeToString(str1.getBytes("UTF-8"));

System.out.println(str2);

//这是base64位解码

Decoder decoder2 = java.util.Base64.getDecoder();

byte[] bytes2 = decoder2.decode(str2);

System.out.println(new String(bytes2,"UTF-8"));

# ScriptEngineManager类

ScriptEngineManager为ScriptEngine类实现发现和实例化机制，并且还维护一组键/值对，用于存储由Manager创建的所有引擎共享的状态。

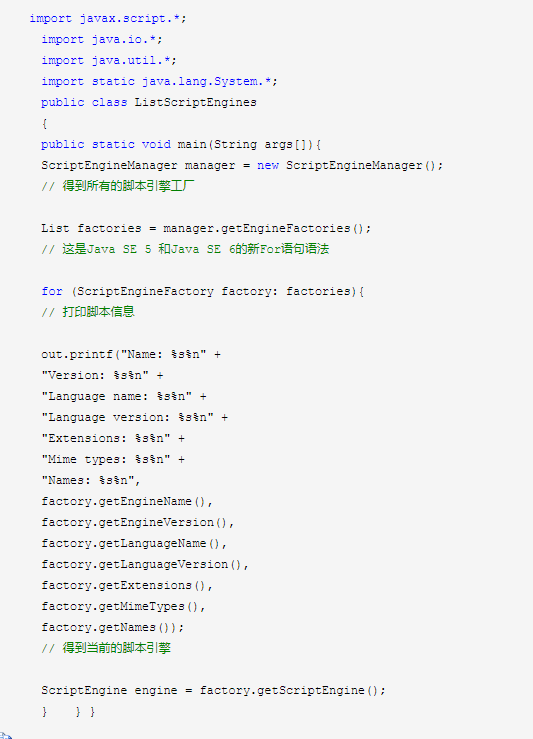
# ScriptEngine接口

## 基本概念

Java SE 6最引人注目的新功能之一就是内嵌了脚本支持。在默认情况下，Java SE 6只支持JavaScript，但这并不以为着Java SE 6只能支持JavaScript。在Java SE 6中提供了一些接口来定义一个脚本规范，也就是JSR223。通过实现这些接口，Java SE 6可以支持任意的脚本语言(如PHP或Ruby)。

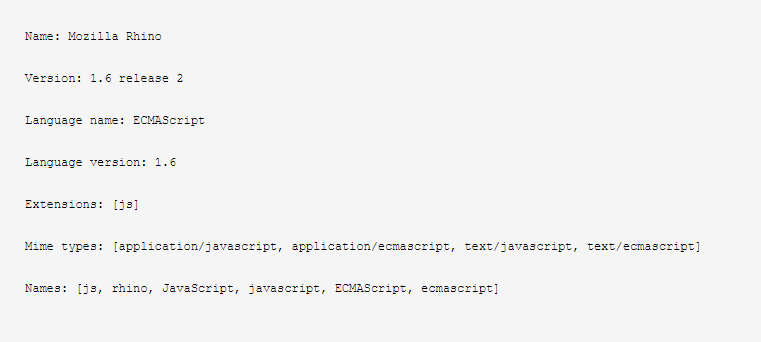
在使用Java SE 6运行脚本之前，必须要知道你的Java SE 6支持什么脚本语言。在javax.script包中有很多的类，但这些类中最主要的是ScriptEngineManager，可以通过这个类得到当前 Java SE 6所支持的所有脚本。

示例代码：



上面的例子必须要在Java SE 6中编译。其中import static java.lang.System.\*是新的语法，将System中的所有静态成员进行引用，以后就可以直接使用out、in或err了。

通过运行java ListScriptEngines，将显示如下信息：



在最下面一行是脚本的别名，也就是使用它们中的任意一个都可以。得到一个具体的脚本引擎有3种方法。

## 根据扩展名得到脚本引擎



getEngineByExtension的参数就是Extensions:[js]中[…]里的部分。

## 根据Mime类型得到脚本引擎



getEngineByMimeType的参数可以是Mime types: [application/javascript, application/ecmascript, text/javascript,text/ecmascript]中的任何一个，可以将text/javascript改成text/ecmascript。

## 根据名称得到脚本引擎



getEngineByName后的参数可以是Names: [js, rhino, JavaScript, javascript, ECMAScript, ecmascript]中的任何一个，如可以将javascript改成ecmascript。

上面已经讨论了执行脚本的第一步，就是得到一个可用的脚本引擎。在完成这项工作之后就可以利用这个脚本引擎执行相应的脚本了。我们可以使用ScriptEngine的eval方法来执行脚本。eval方法被重载的多次，但最常用的是 public Object eval(String script)。

## 代码示例

下面的例子演示了如何使用eval方法来执行javascript脚本



上面的例子通过得到当前的小时，并将其转化为问候语。上面的程序的输出信息为：



这个例子最值得注意的是执行的2句脚本，最后一句是date.getHours()。并未将这个值赋给一个javascript变量。这时，eval方法就将这样的值返回。

## 和脚本语言进行交互

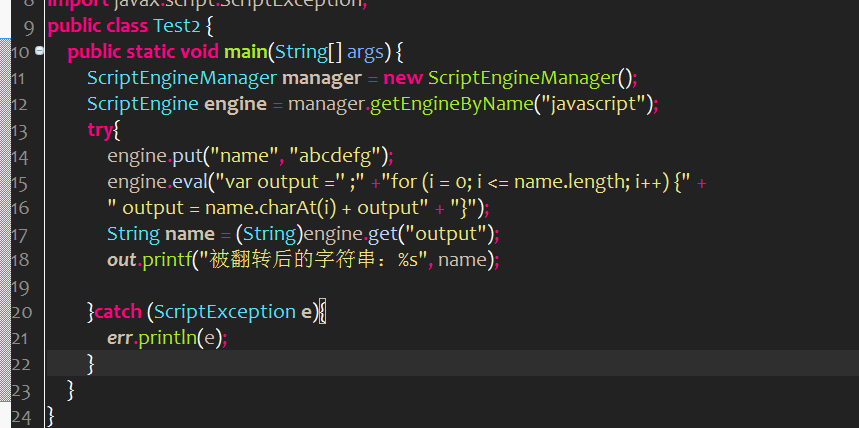
上面例子只是运行了一个非常简单的脚本。这个脚本是孤立的，并未通过Java向这脚本传递任何的值。虽然从这个脚本返回了一个值，但这种返回方式是隐式的。

脚本引擎除了这些简单的功能，还为我们提供了更强大的功能。甚至可以通过Java向脚本语言中传递参数，还可以将脚本语言中的变量的值取出来。这些功能要依靠ScriptEngine中的两个方法put和get。

put方法有两个参数，一个是脚本变量名，另一个是变量的值，这个值是Object类型，因此，可以传递任何值。get方法有一个参数，就是脚本变量的名。

### 代码示例

下面的代码通过javascript脚本将一个字符串翻转(这个字符串是通过java传给javascript的)，然后通过java得到这个被翻转后的字符后，然后输出。



以上代码的输出结果为：



https://www.cnblogs.com/zhangtan/p/8110210.html

## 方法

### public Object eval(String script, Bindings n)

在脚本执行期间，使用Bindings参数作为ScriptEngine的ENGINE\_SCOPE Bindings执行脚本。 使用默认ScriptContext的Reader，Writer和非ENGINE\_SCOPE绑定。 ScriptEngine的ENGINE\_SCOPE绑定不会更改，并且脚本执行不会改变其映射。

参数：script 脚本的源码 n：用于脚本执行的属性的绑定。

返回：脚本返回的值

# LogRecord类

## 基本概念

LogRecord 对象用于在日志框架和单个日志 Handler 之间传递日志请求。将 LogRecord 传递到日志框架中后，它在逻辑上已经属于该框架，客户端应用程序不应再使用或更新它。

注意：如果客户端应用程序尚未显式指定源方法名和源类名，则 LogRecord 类将在第一次访问它们时通过解析调用堆栈来自动推导（根据对 getSourceMethodName 或 getSourceClassName 的调用）。因此，如果日志 Handler 要将 LogRecord 传递给另一个线程或者通过 RMI 传输它，并且如果它希望后续获取方法名或类名信息，则其应该调用 getSourceClassName 和 getSourceMethodName 的其中之一来强制将值填入。

序列化注意事项：

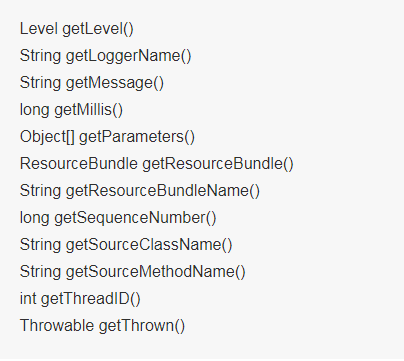
（1）LogRecord 类是可序列化的。

（2）因为参数数组中的对象可能不可序列化，所以在序列化过程中，应该写入参数数组中所有对象的相应 String（使用 Object.toString）。

（3）ResourceBundle 不是作为序列化形式的一部分传输的，但是资源包的名称是，而接收对象的 readObject 方法将尝试查找合适的资源包。

## 对这个类的理解

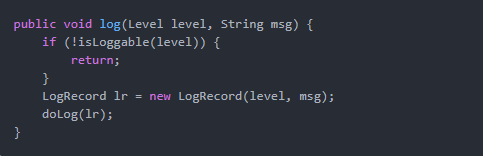
LogRecord可以理解为是一个DTO，那么LogRecord里面到底存储了哪些数据呢？我们可以看一下LogRecord类中含有哪些get/set方法，get方法如下所示：



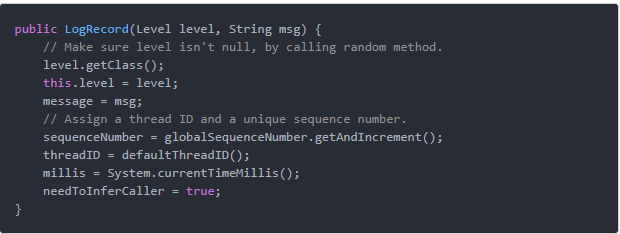
每个get方法都有相应的set方法。在记录日志的过程中可以直接使用LogRecord对象，也可以不直接使用LogRecord对象，而是由日志组件内部为我们自动组装成一个LogRecord对象。我们之前通过logger.info这一类方法记录日志时，内部会根据我们传入的参数以及调用的方法自动组装成一个LogRecord对象。例如：

logger.info("Hello")

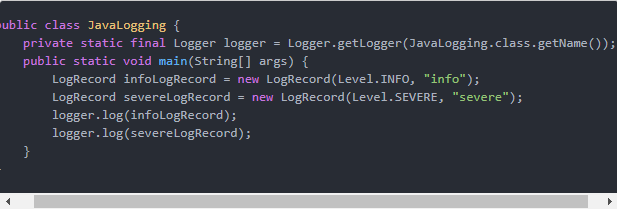
该方法首先转调方法logger.log(Level.INFO, “Hello”),该方法内部自动帮我们创建了LogRecord：



LogRecord给我们提供了一个含有两个参数的构造方法，第一个参数指定需要记录的信息的级别，第二个参数指定需要记录的信息，即:



当然我们也可以直接构造LogRecord对象，然后通过Logger.log(LogRecord record)方法来记录日志，如下所示：



## 构造方法

### [LogRecord](https://www.oschina.net/uploads/doc/javase-6-doc-api-zh_CN/java/util/logging/LogRecord.html#LogRecord(java.util.logging.Level,%20java.lang.String))([Level](https://www.oschina.net/uploads/doc/javase-6-doc-api-zh_CN/java/util/logging/Level.html) level, [String](https://www.oschina.net/uploads/doc/javase-6-doc-api-zh_CN/java/lang/String.html) msg)

用给定级别和消息值构造 LogRecord

参数：

level：日志级别值

msg：原始的非本地化日志消息（可以为 null）

## 方法

### Level getLevel()

获取日志消息级别，例如 Level.SEVERE。

### void [setLoggerName](https://www.oschina.net/uploads/doc/javase-6-doc-api-zh_CN/java/util/logging/LogRecord.html#setLoggerName(java.lang.String))([String](https://www.oschina.net/uploads/doc/javase-6-doc-api-zh_CN/java/lang/String.html) name)

设置logger的名称

### void [setResourceBundleName](https://www.oschina.net/uploads/doc/javase-6-doc-api-zh_CN/java/util/logging/LogRecord.html#setResourceBundleName(java.lang.String))([String](https://www.oschina.net/uploads/doc/javase-6-doc-api-zh_CN/java/lang/String.html) name)

设置本地化资源包名称。

### void [setResourceBundle](https://www.oschina.net/uploads/doc/javase-6-doc-api-zh_CN/java/util/logging/LogRecord.html#setResourceBundle(java.util.ResourceBundle))([ResourceBundle](https://www.oschina.net/uploads/doc/javase-6-doc-api-zh_CN/java/util/ResourceBundle.html) bundle)

设置本地化资源包

### void [setSequenceNumber](https://www.oschina.net/uploads/doc/javase-6-doc-api-zh_CN/java/util/logging/LogRecord.html#setSequenceNumber(long))(long seq)

设置序列号

### void [setSourceClassName](https://www.oschina.net/uploads/doc/javase-6-doc-api-zh_CN/java/util/logging/LogRecord.html#setSourceClassName(java.lang.String))([String](https://www.oschina.net/uploads/doc/javase-6-doc-api-zh_CN/java/lang/String.html) sourceClassName)

设置（可能）发出日志请求的类的名称。

### void [setSourceMethodName](https://www.oschina.net/uploads/doc/javase-6-doc-api-zh_CN/java/util/logging/LogRecord.html#setSourceMethodName(java.lang.String))([String](https://www.oschina.net/uploads/doc/javase-6-doc-api-zh_CN/java/lang/String.html) sourceMethodName)

设置（可能）发出日志请求的方法的名称。

### void [setThreadID](https://www.oschina.net/uploads/doc/javase-6-doc-api-zh_CN/java/util/logging/LogRecord.html#setThreadID(int))(int threadID)

设置产生消息的线程的标识符。

### void [setThrown](https://www.oschina.net/uploads/doc/javase-6-doc-api-zh_CN/java/util/logging/LogRecord.html#setThrown(java.lang.Throwable))([Throwable](https://www.oschina.net/uploads/doc/javase-6-doc-api-zh_CN/java/lang/Throwable.html) thrown)

设置与日志事件相关联的 throwable。

# Logger类

## 构造函数

### static Logger getLogger(String name)

为指定子系统查找或创建一个 logger。

### static Logger getLogger(String name, String resourceBundleName)

为指定子系统查找或创建一个 logger。

说明：name是Logger的名称，当名称相同时候，同一个名称的Logger只创建一个。

# PropertyResourceBundle类

## 基本概念

资源包的具体子类使用一组从一个属性文件的静态字符串，用于管理一个本地化的资源。以下是有关的PropertyResourceBundle要点：

这个类可以从InputStream或Reader，该代表一个属性文件来构建

## 类的源码

