



EE基础与应用

翁秀木

内容梗概

- Java SE的温故与知新
- Java EE的基本学习
- Java 开源项目的初涉与进阶
- Java EE应用服务器和IDE的操练

考核方案

- 学习表现20%
- 练习和作业（任务） 25%
- 期中测试25%
- 期末考试30%

第二回 常用基本工具

- 字符工具
- 日期工具

学习目标

- 掌握String、StringBuffer、StringBuilder、Character类和开源项目commons中的基本方法，以play with 字符串和单字符。
- 掌握字符类的最佳实践（如String，StringBuilder vs. StringBuffer）
- 掌握Date、Calendar类的常用方法，以play with 日期。

String类的声明

String是项目实践中经常用的类，如教材例子，在学生类中我们声明3个属性（field）分别为学生学号、姓名以及院系。

```
public class Student {  
    private String sno, sname, sdept;  
}
```

String类的赋值

```
String str1="Java";  
str1="Java EE";
```

```
str1= new String("Java EE");
```

String的比较

(1) 用 `==` 比较。

(2) 用 `equals()`和 `equalsIgnoreCase()`方法比较

(3) 用 `compareTo()`方法比较

`str1.compareTo(str2)` , 返回值为小于0、0或大于0, 取决于`str1`和`str2`的字典顺序。注意`compareTo()`区分大小写。`compareTo()`是`Comparable`接口声明的方法。

String类中的常用方法

substring(int beginIndex, int endIndex)

substring(int beginIndex)

charAt(int index)

split(String regex) // 正则表达式, 参见Pattern类javadoc

indexOf()

lastIndexOf()

startsWith(String str)

endsWith(String str)

contains(CharSequence s) // since java 1.5

String类中的常用方法

isEmpty() // 判断是否为空串

trim()

replace(char oldChar, char newChar)

replaceAll(String regex, String replacement)

replace(CharSequence target,
CharSequence replacement) // since 1.5

replace**First**(String regex, String replacement)

valueOf(int)

valueOf(boolean)

valueOf(char)

Character类

Useful Methods in the Character Class

Method	Description
<pre>boolean isLetter(char ch) boolean isDigit(char ch)</pre>	Determines whether the specified char value is a letter or a digit, respectively.
<pre>boolean isWhitespace(char ch)</pre>	Determines whether the specified char value is white space.
<pre>boolean isUpperCase(char ch) boolean isLowerCase(char ch)</pre>	Determines whether the specified char value is uppercase or lowercase, respectively.
<pre>char toUpperCase(char ch) char toLowerCase(char ch)</pre>	Returns the uppercase or lowercase form of the specified char value.
<pre>toString(char ch)</pre>	Returns a <code>String</code> object representing the specified character value — that is, a one-character string.

StringBuffer类

StringBuffer是内存中的字符串变量。可修改的字符串序列，该类的对象实体内存空间可以自动改变大小，可以存放一个可变的字符序列。与String最大的区别在于StringBuffer本身串的内容是可变的，所以如果需要频繁对字符串本身内容修改，又不想改变串的内存引用地址时适合使用它。

StringBuffer类

1、构造器

`StringBuffer()`: 当使用无参数的构造器时, 分配给该对象的实体初始容量可以容纳16个字符, 当该扩展字符序列长度 >16 时, 实体容量自动增加以适应新字符串。

`StringBuffer(int size)`: 可以指定分配给该对象的实体的初始容量为参数size指定的字符个数。当对象实体长度 $>size$ 时自动增加。

StringBuffer类

StringBuffer(String s): 分配给该对象的实体的初始容量为参数字符串s的长度+16，当对象实体长度大于初始容量时，实体容量自动增加。

```
StringBuffer buf = new StringBuffer("java");
```

StringBuffer对象可以通过length()函数获取实体存放的字符序列长度。通过capacity()函数获取当前实体的实际容量。。

StringBuffer类

2、容量、实际长度

容量指StringBuffer目前可以存放字符个数，如果不够则可以自动再开辟存储空间。而长度则是StringBuffer实际存储的字符个数。

`buf1.capacity();`//获取buf1的目前容量。

`buf2.length();`//获取buf2的目前实际字符的个数。

StringBuffer类

3、向StringBuffer追加对象append(Object)

```
buf1.append("Java").append(" C#").append(" VB");
```

当使用append(Object)函数是buf1本身内容在改变，这和String有本质区别，如String使用substring()函数时String本身并没有变，只是读取其部分内容赋给另外的串而已。

StringBuffer类

4、插入子串insert(int index,Object)

buf1.insert(4," c++");//在第5个位置处插入" c++“

5、删除字符串delete(int start,int end)

buf1.delete(4,8); //删除从第5^{处理}个位置到第8个位置的串。

如果你对字符串中的内容经常进行操作，特别是修改串内容时，则可以使用StringBuffer。如果最后需要String，那么使用StringBuffer的toString()方法好了。

StringBuffer与StringBuilder类

StringBuilder类与StringBuffer类完全相同，除了StringBuffer类的方法标上了synchronized。

StringBuilder类创建于java5.0，若StringBuffer类的实例只供单线程用或多线程访问时是安全的（即线程安全），建议用StringBuilder替代StringBuffer，以实现更高效率。

字符类的最佳实践 (best practices)

- 若其他类型更合适，就不要使用String

不要用String代替数值类型，例如int, float, double, BigInteger, BigDecimal等（注意：若需要精确小数值，用BigDecimal，不要用float和double，尤其是表示金额时）。

不要用String作复合类型（aggregate type）

如String compoundKey = 区号 + "#" + 电话号码
应该写一个class来表示这个复合类型，即这个class的属性包括区号和电话号码。

字符类的最佳实践 (best practices)

- **注意String连接的效率**

```
String s = "GEM";  
for (int i = 0; i < 100; i++) {  
    s = s + " GEM";  
}
```

连接String应如下:

```
StringBuilder sb = new StringBuilder("GEM");  
for (int i = 0; i < 100; i++) {  
    sb.append(" GEM");  
}
```

什么是自由软件（开源软件）

The **freedom** to **run the program**, for any purpose (freedom 0).

The **freedom** to **study how the program works**, and **change it** so it does your computing as you wish (freedom 1). **Access to the source code** is a precondition for this.

The **freedom** to **redistribute copies** so you can **help your neighbor** (freedom 2).

The **freedom** to **distribute copies of your modified versions** to **others** (freedom 3). By doing this you can give the whole community a chance to benefit from your changes. **Access to the source code** is a precondition for this

开源项目-

Apache Commons的lang包（网站见备注）

- 判断String是空（""）、或空白（1或多个空格）、或null

用： `if (str == null || "".equals(str.trim()))`

还是用： lang的StringUtils类：
`StringUtils.isBlank(str)`

Apache Commons的lang包

- StringUtils里的方法是**null safe**的，即**不会**抛出 **NullPointerException**，除非是标注了 **Deprecated** 的方法。
- 判断两字符串是否相等：
if ((str1 == null && str2 == null) || (str1 != null && str1.equals(str2)))
 - 若用**StringUtils** : **StringUtils.equals(str1, str2);**

StringUtils.equals(null, null) = true

StringUtils.equals(null, "abc") = false

StringUtils.equals("abc", null) = false

StringUtils.equals("abc", "abc") = true

StringUtils.equals("abc", "ABC") = false

最佳实践 (Best Practices)

```
String str1 = getString();  
String str2 = "GEM";  
if (str1.equals(str2)) {  
    // business logic  
}
```

应该先判断是否为null

```
String str1 = getString();  
String str2 = "GEM";  
if (str1 != null && str1.equals(str2)) {  
    // business logic  
}
```


org.apache.commons.lang3.StringUtils



- **IsEmpty/IsBlank** - checks if a String contains text
- **Trim/Strip** - removes leading and trailing whitespace
- **Equals** - compares two strings null-safe
- **startsWith** - check if a String starts with a prefix null-safe
- **endsWith** - check if a String ends with a suffix null-safe
- **IndexOf/LastIndexOf/Contains** - null-safe index-of checks
- **IndexOfAny/LastIndexOfAny/IndexOfAnyBut/LastIndexOfAnyBut** - index-of any of a set of Strings
- **ContainsOnly/ContainsNone/ContainsAny** - does String contains only/none/any of these characters
- **Substring/Left/Right/Mid** - null-safe substring extractions
- **SubstringBefore/SubstringAfter/SubstringBetween** - substring extraction relative to other strings
- **Split/Join** - splits a String into an array of substrings and vice versa
- **Remove/Delete** - removes part of a String
- **Replace/Overlay** - Searches a String and replaces one String with another
- **Chomp/Chop** - removes the last part of a String
- **AppendIfMissing** - appends a suffix to the end of the String if not present
- **PrependIfMissing** - prepends a prefix to the start of the String if not present
- **LeftPad/RightPad/Center/Repeat** - pads a String
- **UpperCase/LowerCase/SwapCase/Capitalize/Uncapitalize** - changes the case of a String
- **CountMatches** - counts the number of occurrences of one String in another
- **IsAlpha/IsNumeric/IsWhitespace/IsAsciiPrintable** - checks the characters in a String
- **DefaultString** - protects against a null input String
- **Reverse/ReverseDelimited** - reverses a String
- **Abbreviate** - abbreviates a string using ellipsis
- **Difference** - compares Strings and reports on their differences
- **LevenshteinDistance** - the number of changes needed to change one String into another

java.util.Date

- 创建java.util.Date： 获取系统日期

```
Date date=new Date(); //以系统当前日期构造对象
```

```
int year=date.getYear()+1900;// date.getYear()得到当前年份与  
1900年得差值。
```

```
int month=date.getMonth()+1;//月份下标从 0开始
```

注意： 不要将java.util.Date与java.sql.Date混淆

SimpleDateFormat

SimpleDateFormat是一个能将**Date型日期**按照指定格式输出**String型日期**，也可以将**String型日期**解析成**Date型日期**。

日期和时间模式： 参见SimpleDateFormat的**javadoc**

- **Date型日期变String型日期：**

```
SimpleDateFormat format=  
    new SimpleDateFormat("yyyy-MM-dd");
```

```
String s=format.format(date);
```

SimpleDateFormat

- String型日期变Date型日期：

假设有一个日期型的文本字符串，而希望从文本日期数据创建一个日期对象，则需要通过SimpleDateFormat类的parse()方法。要注意字符串与格式要一一对应。否则会出现解析异常

ParseException，如：

```
SimpleDateFormat format =  
    new SimpleDateFormat("yyyy-MM-dd");  
String s="2012-10-15";  
try {  
    Date d2=format.parse(s);  
} catch(ParseException e) {  
    throw new RuntimeException(e)  
}
```

Date - getTime()

- 计算日期差：

Date类提供了一个getTime()函数计算当前日期对象与1970年01月01日00:00:00之间相差的毫秒数。通过该函数可以计算两个日期之间差的天数等。

Java Calendar的使用

➤ Calendar 日历类使用：

Calendar 是日历类，完成日历的一些计算功能。Calendar 是一个抽象类，也就是说你无法直接通过new方法获得它的一个实例，GregorianCalendar 是Calendar的一个具体实现。下面对该类的使用予以介绍。

构造一个日历实例

```
Calendar cal=Calendar.getInstance();
```

以系统当期日期构造Calendar实例。

注意Calendar.getInstance()背后机制**不是**单
例模式。

DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss aaa"); // **HH为24小时制**, hh为12小时制

```
Calendar cal1 = Calendar.getInstance();  
System.out.println("cal1=" +  
dateFormat.format(cal1.getTime()));
```

```
Thread.sleep(3000);
```

```
Calendar cal2 = Calendar.getInstance();  
System.out.println("cal2=" +  
dateFormat.format(cal2.getTime()));
```

Design Pattern 设计模式

The elements of this language are entities called patterns. Each **pattern** describes a **problem** that occurs over and over again in our **environment**, and then describes the core of the **solution** to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice. — Christopher Alexander

A **pattern** is a **solution** to a **problem** in a **context**.


```
public class LazySingleton {  
    private static LazySingleton myInstance;  
    private LazySingleton() { }
```

```
public static LazySingleton getInstance() {  
    if (myInstance == null) {  
        myInstance = new LazySingleton();  
    }  
    return myInstance ;  
}  
}
```

```
LazySingleton ls = LazySingleton.getInstance();  
LazySingleton ls2 = LazySingleton.getInstance();
```

```
public class EagerSingleton {  
    private static final EagerSingleton myInstance =  
        new EagerSingleton();  
  
    private EagerSingleton() { }  
  
    public static EagerSingleton getInstance() {  
        return myInstance;  
    }  
}
```

```
EagerSingleton es =  
EagerSingleton.getInstance();  
EagerSingleton es2 =
```

Java Calendar的使用

➤ 读取日期某个部分值：(参见Calendar的**javadoc**)

通过get函数，该函数需要一个日期部分描述符，表示取哪个部分。例如，若想知道今天是一年中的哪一天，可以用：

```
Calendar calendar = Calendar.getInstance();  
int day = calendar.get(Calendar.DAY_OF_YEAR);
```

```
int year= calendar.get(Calendar.YEAR);  
int weekday= calendar.get(Calendar.DAY_OF_WEEK);  
int day=
```

```
calendar.getActualMaximum(Calendar.DAY_OF_MONTH);
```

Java Calendar的使用

➤设置时间 set :

通过set函数可以重新设置日期某个部分值。该函数有两个参数，一个是日期部分描述符，一个该部分值。如：

```
cal.set(Calendar.YEAR, 2010);  
cal.set(Calendar.MONTH, 4);// 1月为0，此为5月  
cal.set(Calendar.DAY_OF_MONTH, 5);//下标从1开始  
cal.set(Calendar.HOUR, 2);。
```

Java Calendar的使用

➤ 日期加法：

通过add函数可以对日期某个部分值进行加减（负值即减法）。该函数有两个参数，一个是日期部分描述符，一个该部分值。如：

```
cal.add(Calendar.MINUTE, 15);//将当前日期实例分钟加上15
```

Java Calendar的使用

➤一些常见日期描述符：

AM 指示上午（下午为PM）。

DATE 指示一个月中的某天。

DAY_OF_MONTH 指示一个月中的某天，与DATE同

DAY_OF_WEEK 指一个星期中的某天，周六返回6，周日返回0

DAY_OF_WEEK_IN_MONTH 指示当前月第几个星期

DAY_OF_YEAR 指示当前年中的天数。

HOUR 指示上午或下午的小时，12小时制（0~11），中午和午夜用0而不用12表示。

HOUR_OF_DAY 指示一天中的小时，24小时制。

WEEK_OF_YEAR指示当前年中的星期数。

Calendar与Date对象的转换

```
Date date = new Date();
```

```
Calendar calendar = Calendar.getInstance();  
calendar.setTime(date);
```

```
Date date2 = calendar.getTime();
```

isSameDay(Calendar cal1, Calendar cal2):

Checks if two calendar objects are on the same day ignoring time.

isSameDay(Date date1, Date date2):

Checks if two date objects are on the same day ignoring time.

addDays(Date date, int amount):

Adds a number of days to a date returning a new object.

addHours(Date date, int amount):

Adds a number of hours to a date returning a new object.

toCalendar(Date date):

Converts a Date into a Calendar.

setDays(Date date, int amount):

Sets the day of month field to a date returning a new object.

setHours(Date date, int amount):

Sets the hours field to a date returning a new object.

未完待续，谢谢！