**Java Utils**

封装了一些常用Java操作方法,便于重复开发利用. 另外希望身为Java牛牛的你们一起测试和完善 一起封装和完成常用的Java代码。 节约撸码时间以方便有更多的时间去把妹子～

**开发环境**

Win7x64 && ubuntu14

JDK1.7

IDEA14

# 通用的工具类

## CharsetUtil

编码相关的封装类

## ChinesUtil

中文相关的一些工具类

## ClassUtil

Class与反射相关的一些工具类

## ConvertUtil

常用类型的转换

## FileUtil

文件相关的操作封装

## ProUtil

属性相关的一些封装

## RandomUtil

随机数相关的操作

## RegUtil

正则相关的一些操作

## SecUtil

一些安全相关的方法

## StreamUtil

流相关的操作方法

## StringUtil

字符串相关的一些操作方法

## SysUtil

系统相关的一些操作方法

## ValidUtil

提供些对象有效性的判断

## ZIPUtil

Zip压缩相关的属性

# 时间相关的工具类

DateUtil提供了些时间相关的操作方法。该类中默认的为时间定义了如下通用的格式

*//日期时间类型格式***private static** String *DATETIME\_FORMAT* = **"yyyy-MM-dd HH:mm:ss"**;  
  
*//日期类型格式***private static** String *DATE\_FORMAT* = **"yyyy-MM-dd"**;  
  
*//时间类型的格式***private static** String *TIME\_FORMAT* = **"HH:mm:ss"**;

因为JDK提供的SimpleDateFormat是非线程安全的，因此此处采用线程局部变量技术：

*//注意SimpleDateFormat不是线程安全的***private static** ThreadLocal<SimpleDateFormat> *ThreadDateTime* = **new** ThreadLocal<SimpleDateFormat>();  
**private static** ThreadLocal<SimpleDateFormat> *ThreadDate* = **new** ThreadLocal<SimpleDateFormat>();  
**private static** ThreadLocal<SimpleDateFormat> *ThreadTime* = **new** ThreadLocal<SimpleDateFormat>();  
  
**private static** SimpleDateFormat DateTimeInstance() {  
 SimpleDateFormat df = *ThreadDateTime*.get();  
 **if** (df == **null**) {  
 df = **new** SimpleDateFormat(*DATETIME\_FORMAT*);  
 *ThreadDateTime*.set(df);  
 }  
 **return** df;  
}  
  
**private static** SimpleDateFormat DateInstance() {  
 SimpleDateFormat df = *ThreadDate*.get();  
 **if** (df == **null**) {  
 df = **new** SimpleDateFormat(*DATE\_FORMAT*);  
 *ThreadDate*.set(df);  
 }  
 **return** df;  
}  
  
**private static** SimpleDateFormat TimeInstance() {  
 SimpleDateFormat df = *ThreadTime*.get();  
 **if** (df == **null**) {  
 df = **new** SimpleDateFormat(*TIME\_FORMAT*);  
 *ThreadTime*.set(df);  
 }  
 **return** df;  
}

该工具类提供了一些常用的方法和一些特色方法，具体的方法请查阅相关的API文档。例如

**public static** String DateTime() //获取当前时间的字符串形式

**public static** String DateTime(Date date) //将指定的时间格式化

**public static** Date DateTime(String datestr) //将指定的字符串解析为时间类型

**public static** Date year(**int** year)// 在当前时间的基础上加或减去year年

**public static** Date year(Date date, **int** year)// 在指定的时间上加或减去几年

**public static long** Subtract(Date date1, Date date2)// *时间date1和date2的时间差-单位秒*

**public static long** Subtract(String date1, String date2)// *时间date1和date2的时间差-单位秒*

一些特色方法，例如判断常用的计算工资时间和工作时长的方法

// *返回俩个时间在时间段(例如每天的08:00-18:00)的时长-单位秒*

**public static long** subtimeBurst(String startDate, String endDate, String timeBurst)

//*时间Date在时间段(例如每天的08:00-18:00)上增加或减去second秒*

**public static** Date calculate(String date, **int** second, String timeBurst)

例如(假如你们的工作时间为**08:00-21:00**)loader在早上**2015-01-29 08:32:00**将一件工作交由你做,证件工作的时长为14个小时，那么意味着你必须在**2015-01-30 09:32:00**完成这项工作（）

DateUtil.*DateTime*(DateUtil.*calculate*(**"2015-01-29 08:32:00"**, 3600 \* 14, **"08:00-21:00"**));

//算法验证

**2015-01-29 08:32:00**

**2015-01-29 21:00:00**

**=12:28**

**2015-01-30 08:00:00**

**+1:32**

**2015-01-30 09:32:00**

同样可以直接计算俩个时间之间的工作时长

DateUtil.*subtimeBurst*(**"2015-06-24 08:00:00"**, **"2015-06-23 20:24:00"**, **"08:00-21:00"**)

# 集合相关的工具类

集合相关的工具类CollectionUtil提供了如下一些方法以便于操作，因为Java对集合的操作都是引用型的传递，为了不降低影响，该工具类不会修改原集合的引用关系。

1.使用指定的Filter过滤集合

static <T> java.util.List<T> Filter(java.util.List<T> list, ListFilter filter)

static <K,V> java.util.Map Filter(java.util.Map<K,V> map, MapFilter filter)

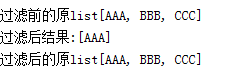
static <T> java.util.Queue Filter(java.util.Queue<T> queue, QueueFilter filter)

static <T> java.util.Set<T> Filter(java.util.Set<T> set, SetFilter filter)

例如可以直接使用下面的方式进行调用

List<String> filter = CollectionUtil.*Filter*(list1, **new** ListFilter() {  
 @Override  
 **public boolean** filter(Object o) {  
 **return "AAA"**.equals((String) o);  
 }  
});

测试结果如下：



2. 求俩个集合的交集

static <T> java.util.List<T> intersection(java.util.List<T> list1, java.util.List<T> list2)

static <K,V> java.util.Map<K,V> intersection(java.util.Map<K,V> map1, java.util.Map<K,V> map2)

static <T> java.util.Queue<T> intersection(java.util.Queue<T> queue1, java.util.Queue<T> queue2)

static <T> java.util.Set<T> intersection(java.util.Set<T> set1, java.util.Set<T> set2)

3. 求俩个集合的差集

static <T> java.util.List<T> subtract(java.util.List<T> list1, java.util.List<T> list2)

static <K,V> java.util.Map<K,V> subtract(java.util.Map<K,V> map1, java.util.Map<K,V> map2)

static <T> java.util.Queue<T> subtract(java.util.Queue<T> queue1, java.util.Queue<T> queue2)

static <T> java.util.Set<T> subtract(java.util.Set<T> set1, java.util.Set<T> set2)

# FTP相关的工具类

FTP常用的操作有文件上传下载等操作。使用该工具类需要在classpath中提供[ftp.properties](ftp://ftp.properties)

配置文件具体的配置信息为：

**FTPTest.host**=**135.224.9.67  
FTPTest.port**=**21  
FTPTest.username**=**ftptest  
FTPTest.password**=**ftptest  
FTPTest.remoteDir**=**./test  
FTPTest.localDir**=**c:/download  
FTPTest.Encoding**=**gbk  
FTPTest.passiveMode**=**true**

之后可以像如下的方式进行使用

|  |
| --- |
| **public static void** main(String[] args) **throws** IOException {  FTPUtil ftp = FTPFactory.*getInstance*(**"FTPTest"**);    String remoteFile = **"test/22.txt"**;  ftp.downLoad(remoteFile);   *//PrintUtil.print(ftp.listFile("./"));* System.***out***.println(**"远程文件是否存在:"** + ftp.isExists(remoteFile));    System.***out***.println(**"获取当前的工作目录:"** + ftp.getWorkDir());  *//ftp.downLoadDir("");* System.***out***.println(**"创建目录:"**+ftp.mkDir(**"test01"**));  System.***out***.println(**"创建目录:"**+ftp.mkDir(**"test02/"**));  System.***out***.println(**"创建目录:"**+ftp.mkDir(**"test03/test1"**));  System.***out***.println(**"创建目录:"**+ftp.mkDir(**"test04/test1/"**));   String uploadFile =**"c:/windows/system.ini"**;  System.***out***.println(**"上传文件:"**+ftp.putFile(uploadFile,**"windows/system.ini"**,**true**));  String deleteFile =**"22.txt"**;  System.***out***.println(**"删除远程文件:"**+ftp.deleteFile(deleteFile));   String deleteDir =**"checkbox"**;  System.***out***.println(**"删除远程目录:"**+ftp.deleteDir(deleteDir));     PrintUtil.*print*(FTPConstant.*REPLYCODE*);  System.***out***.println(**"上传目录:"** + ftp.putDir(**"C:\\Python27"**, **"python27"**));  *//test FTPClient* FTPClient client = ftp.client();   System.***out***.println(client.makeDirectory(**"test"**));  System.***out***.println(client.makeDirectory(**"test1/"**));  System.***out***.println(client.makeDirectory(**"test/test1"**));  System.***out***.println(client.makeDirectory(**"test2/test2"**));    FTPFile[] list = client.listFiles();  System.***out***.println(list);  String[] list1 = client.listNames();  System.***out***.println(list1);   System.***out***.println(**"执行命令/"** + client.sendCommand(**"mkdir testssss"**));  **int** ss = client.sendCommand(**"ls"**);    ftp.destory(); } |

# 加密解密相关的工具类

Base32与Base64

# Bean相关工具类

Bean相关的工具类BeanUtil提供了一些常用Bean相关的工具类。

在国内想如下的编码方式在国内是比较常用见的

业务实体1定义属性：

**private** String **operationName**;  
  
**private** String **operation\_type**;

业务实体2定义属性：

**private** Integer **logId**;  
  
**private** String **logyype**;  
  
**private** String **operationname**;  
  
**private** String **operation\_type**;

因为Java本身区分大小写所以一般的Bean工具类无法通过hasProperty/ getProperty

等方法在业务实体2中获取到**operationname**属性的值。更不用说进行属性对拷了。

例如判断是否有属性。

判断自己定义的而非继承的属性pro是否存在

static boolean hasDeclaredProperty(java.lang.Object bean, java.lang.String pro)

static boolean hasProperty(java.lang.Object bean, java.lang.String pro)

static boolean hasPropertyFilter(java.lang.Object bean, java.lang.String pro, PropertyFilter filter)

static boolean hasPropertyIgnoreCase(java.lang.Object bean, java.lang.String pro)

@Test  
**public void** testHasProperties() {  
 BusinessLog bean = **new** BusinessLog();  
  
 *//Bean类是否有operationName属性  
 assertEquals*(**true**, BeanUtil.*hasProperty*(bean, **"operationName"**));  
  
  
 *//Bean类中是否存在operationName属性(判断将忽略大小写)  
 assertEquals*(**true**, BeanUtil.*hasPropertyIgnoreCase*(bean, **"OperationName"**));  
  
 *//Bean类中是否定义属性logId  
 assertEquals*(**false**, BeanUtil.*hasDeclaredProperty*(bean, **"logId"**));  
  
 *//Bean类中是否存在属性(对属性命执行自定义的过滤函数后比较)  
 assertEquals*(**true**, BeanUtil.*hasPropertyFilter*(bean, **"operationType"**,

**new** PropertyFilter(){  
 @Override  
 **public** String Properties(String pro) {  
 *//忽略属性字段中"\_" 并安装小写比较* **return** StringUtil.*remove*(pro, **"\_"**).toLowerCase();  
 }  
 }));  
  
}

获取属性值

static java.lang.Object getDeclaredProperty(java.lang.Object bean, java.lang.String pro)

获取对象自定义的属性

static java.lang.Object getDeclaredPropertyPeaceful(java.lang.Object bean, java.lang.String pro)

获取对象自定义的属性

static java.lang.Object getProperty(java.lang.Object bean, java.lang.String pro)

获取对象的属性

static java.lang.Object getPropertyFilter(java.lang.Object bean, java.lang.String pro, PropertyFilter filter)

使用自定义的过滤器获取对象的属性获取对象的属性

static java.lang.Object getPropertyFilterPeaceful(java.lang.Object bean, java.lang.String pro, PropertyFilter filter)

使用自定义的过滤器获取对象的属性

static java.lang.Object getPropertyIgnoreCase(java.lang.Object bean, java.lang.String pro)

获取对象的属性(忽略属性名字大小写)

static java.lang.Object getPropertyIgnoreCasePeaceful(java.lang.Object bean, java.lang.String pro)

获取对象的属性(忽略属性名字大小写)

static java.lang.Object getPropertyPeaceful(java.lang.Object bean, java.lang.String pro)

获取对象的属性

@Test  
**public void** testGetProperties() **throws** InvocationTargetException, IllegalAccessException {  
 String value = **"Test BeanUtil getProperties Method"**;  
 BusinessLog bean = **new** BusinessLog();  
 bean.setOperationName(value);  
  
 *assertEquals*(value, BeanUtil.*getProperty*(bean, **"operationName"**));  
  
 *//获取属性(忽略大小写):  
 assertEquals*(value, BeanUtil.*getPropertyIgnoreCase*(bean, **"operationname"**));  
  
 *//平静的获取属性  
 assertEquals*(value, BeanUtil.*getPropertyPeaceful*(bean, **"operationName"**));  
  
 *//获取属性(使用自定的过滤函数):  
 assertEquals*(value, BeanUtil.*getPropertyFilter*(bean, **"operation\_Name"**, **new** PropertyFilter() {  
 @Override  
 **public** String Properties(String pro) {  
 **return** StringUtil.*remove*(pro, **"\_"**).toLowerCase();  
 }  
 }));  
  
 *//获取bean定义的属性  
 //PrintUtil.print(BeanUtil.getDeclaredPropertyPeaceful(bean, "operationName"));  
 //PrintUtil.print(BeanUtil.getPropertyIgnoreCasePeaceful(bean, "operationName"));*}

设置属性

static void setDeclaredProperty(java.lang.Object bean, java.lang.String pro, java.lang.Object value)

设置对象的自定义属性

static void setDeclaredPropertyPeaceful(java.lang.Object bean, java.lang.String pro, java.lang.Object value)

设置对象的自定义属性

static void setProperty(java.lang.Object bean, java.lang.String pro, java.lang.Object value)

设置对象的属性

static void setPropertyFilter(java.lang.Object bean, java.lang.String pro, java.lang.Object value, PropertyFilter filter)

使用自定义的filter进行属性设值

static void setPropertyFilterPeaceful(java.lang.Object bean, java.lang.String pro, java.lang.Object value, PropertyFilter filter)

使用自定义的filter进行属性设值

static void setPropertyIgnoreCase(java.lang.Object bean, java.lang.String pro, java.lang.Object value)

设置对象的属性忽略大小写

static void setPropertyIgnoreCasePeaceful(java.lang.Object bean, java.lang.String pro, java.lang.Object value)

设置对象的属性忽略大小写

static void setPropertyPeaceful(java.lang.Object bean, java.lang.String pro, java.lang.Object value)

设置对象的属性

@Test  
**public void** testSetProperties() **throws** InvocationTargetException, IllegalAccessException {  
 BusinessLog bean = **new** BusinessLog();  
 BeanUtil.*setProperty*(bean, **"operationName"**, **"Properties's value1"**);  
 PrintUtil.*println*(bean);  
 BeanUtil.*setPropertyIgnoreCase*(bean, **"operationname"**, **"Properties's value2"**);  
 PrintUtil.*println*(bean);  
 BeanUtil.*setPropertyFilter*(bean, **"operation\_Name"**, **"Properties's value3"**, **new** PropertyFilter() {  
 @Override  
 **public** String Properties(String pro) {  
 **return** StringUtil.*remove*(pro, **"\_"**).toLowerCase();  
 }  
 });  
 PrintUtil.*println*(bean);  
}

属性对拷

static void copyProperties(java.lang.Object srcBean, java.lang.Object destBean)

复制同名属性

static void copyProperties(java.lang.Object srcBean, java.lang.Object destBean, PropertyFilter filter)

使用自定义的属性过滤函数

static void copyPropertiesIgnoreCase(java.lang.Object srcBean, java.lang.Object destBean)

复制同名属性(忽略大小写)

static void copyProperty(java.lang.Object srcBean, java.lang.Object destBean, java.lang.String[] pros)

拷贝对象指定的属性

static void copyPropertyPeaceful(java.lang.Object srcBean, java.lang.Object destBean, java.lang.String[] pros)

拷贝对象指定的属性

例如：

@Test  
**public void** testCopyProperties() **throws** InvocationTargetException, IllegalAccessException {  
 BusinessLog bean1 = **new** BusinessLog();  
 bean1.setOperationName(**"operationName test"**);  
 bean1.setOperation\_type(**"operationName type"**);  
 bean1.setLogType(**"logTypevalue"**);  
  
 Log2 bean2 = **new** Log2();  
  
 PrintUtil.*print*(**"复制前:"** + bean1);  
 PrintUtil.*print*(**"复制前:"** + bean2);  
 BeanUtil.*copyProperties*(bean1, bean2, **new** PropertyFilter() {  
 @Override  
 **public** String Properties(String pro) {  
 **return** StringUtil.*remove*(pro, **"\_"**).toLowerCase().replaceAll(**"yy"**, **"ty"**);  
 }  
 });  
 PrintUtil.*print*(**"复制后:"** + bean1);  
 PrintUtil.*print*(**"复制后:"** + bean2);  
}

# Web相关的工具类

Web相关的工具类有UrlUtil和WebUtil工具类