1.Locale对象方法

//获得默认设置的Locale对象  
Locale locale = Locale.*CHINA*;  
//获得对应语言标签相应的Locale对象  
Locale locale1 = Locale.*forLanguageTag*("en-abc-def-us");  
//获得默认（本地）Locale  
Locale locale2 = Locale.*getDefault*();  
//设置默认Locale  
Locale.*setDefault*(locale2);  
//返回一个当前的Locale中所表示的用来描述Locale的名字  
String name = locale.getDisplayName();  
//获得语言代码  
String code = locale.getLanguage();  
//返回在当前Locale中所表示的语言名称  
String languageName = locale.getDisplayLanguage();  
//返回国家代码  
String countryCode = locale.getCountry();  
//返回国家名  
String countryCode0 = locale.getDisplayCountry();  
//返回语言标签  
String languageTag = locale.toLanguageTag();  
//返回Locale描述，包括语言和国家，用\_分隔  
String toString = locale.toString();

2.

//获得该对象支持的Locale类型  
Locale[] locales = DateFormat.*getAvailableLocales*();

3.数字格式化

//数字格式  
//1.获得Locale对象  
//2.使用工程方法，得到格式器对象   
//3.使用格式器完成格式化和解析工作

//获得Locale对象  
Locale cn = Locale.*CHINA*;

//获得格式器  
NumberFormat nf = NumberFormat.*getCurrencyInstance*(cn);

//格式化  
String result = nf.format(Double.*valueOf*("13333.33"));

//解析

Number num = nf.parse(result.trim());  
Double d = num.doubleValue();

常用方法：

NumberFormat.*getNumberInstance*(cn); //处理数字  
NumberFormat.*getPercentInstance*(cn); //处理百分比  
NumberFormat.*getCurrencyInstance*(cn); //处理货币  
nf.format(Double.*valueOf*("13333.33")); //格式化  
nf.parse("￥13,333.33"); //解析  
nf.setParseIntegerOnly(true); //设置是否应该只解析整型  
nf.isParseIntegerOnly(); //是否只解析整型  
  
nf.setMaximumFractionDigits(1);  
nf.getMaximumFractionDigits();  
nf.setMinimumFractionDigits(1);  
nf.getMinimumFractionDigits();  
nf.setMinimumIntegerDigits(1);  
nf.getMinimumIntegerDigits();  
nf.setMaximumIntegerDigits(1);  
nf.getMaximumIntegerDigits();  
//设置获取整数或小数部分所允许的最大或最小位数

货币Currency的各种方法

//返回域给定ISO4217货币代码或戈丁的Locale中的国家相对应的Currency对象  
Currency currency = Currency.*getInstance*("CNY");  
//设置currency对象  
nf.setCurrency(currency);  
//获得currency对象的ISO4217代码  
currency.toString();  
currency.getCurrencyCode();  
//获得货币的格式化符号,如￥,$  
currency.getSymbol();  
currency.getSymbol(locale);  
//获取该货币小数点后的默认位数  
currency.getDefaultFractionDigits();  
//获取所有可用货币  
Currency.*getAvailableCurrencies*();

4. 时间和日期的格式器

//日期与时间  
//1.月份和星期应该用本地语言来表示  
//2.年月日的顺序要符合本地习惯  
//3.公历可能不是本地首选的日期表示方法  
//4.必须要考虑本地的时区  
//选择一种风格  
FormatStyle style = FormatStyle.*MEDIUM*;  
//获得时间日期格式器，使用当前locale  
DateTimeFormatter dateFormatter = DateTimeFormatter.*ofLocalizedDate*(style);  
DateTimeFormatter timeFormatter = DateTimeFormatter.*ofLocalizedDateTime*(style);  
DateTimeFormatter dateTimeFormatter = DateTimeFormatter.*ofLocalizedDateTime*(style);  
//使用不同locale  
DateTimeFormatter dateTimeFormatter1 = DateTimeFormatter.*ofLocalizedDateTime*(style, style).withLocale(locale);  
//格式化日期/时间(这个解析后的格式为2019.03.10 22:12:33)  
String dateTime = dateTimeFormatter.format(LocalDateTime.*now*());  
//解析日期/时间  
TemporalAccessor localDateTime = dateTimeFormatter.parse(dateTime);

5.排序器

//获得该类支持的Locale对象  
Locale[] locales1 = Collator.*getAvailableLocales*();  
//获得一个排序器  
Collator.*getInstance*();  
Collator collator = Collator.*getInstance*(locale);  
//设置和获取排序器的强度，可以是Collator.PRIMARY,Collator.SECONDARY,Collator.TERTIARY  
collator.setStrength(Collator.*PRIMARY*);  
collator.getStrength();  
//设置和获取排序器的分解模式，分解得越细，判断两个字符串相等就越严格。  
// Collator.CANONICAL\_DECOMPOSITION，Collator.NO\_DECOMPOSITION,Collator.FULL\_DECOMPOSITION  
collator.setDecomposition(Collator.*CANONICAL\_DECOMPOSITION*);  
//返回一个排序器键，这个键包含一个对一组字符串按特定格式分解的结果，可以快速地和其他排序器键进行比较  
CollationKey collationKey = collator.getCollationKey("zJx");  
CollationKey collationKey1 = collator.getCollationKey("Wjx");  
//比较两个排序器键  
collationKey.compareTo(collationKey1);  
//返回str的泛化形式，form的值可以是ND、NKD、NC、NKC之一。  
String s = Normalizer.*normalize*("吴嘉贤", Normalizer.Form.*NFC*);

6．消息格式化

//占位符后面可以跟一个类型和一个风格  
//类型可以是number time date choice  
//number的风格有 integer currency percent  
//date或time的风格有 short medium long full 或者一个日期格式模式，如yyyy-MM-dd HH:mm:ss  
String msg = MessageFormat.*format*("on {0,date,yyyy-MM-dd HH:mm:ss}, a {1} ddd of {2}", new Date(),"wjx","birth");  
//新建一个格式器  
MessageFormat mf = new MessageFormat("on {0,date,yyyy-MM-dd HH:mm:ss}, a {1} ddd of {2}", locale);  
//给消息格式器设置模式  
mf.applyPattern("on {0,date,yyyy-MM-dd HH:mm:ss}, a {1} ddd of {2}");  
//设置或获取Locale对象，这个locale仅被用applyPattern设置的模式使用。  
mf.setLocale(locale);  
//把格式化的字符串追加到sb末尾并返回,后面的FieldPostion不懂用途，可以设置为null  
StringBuffer sb = mf.format(  
 new Object[]{new Date(), "wjx", "birth"}, new StringBuffer(), new FieldPosition(1));  
//格式化选择器  
mf.applyPattern("on {0,date,yyyy-MM-dd HH:mm:ss}, {1,choice,0#a wjx|1#the wjx|3#hello wjx} ddd of {2}");  
//使用格式化选择器，用下表表示选择第几个  
mf.format(new Object[]{new Date(), 2, "birth"});

7.编码格式

//多个平台上通用的行结束符%n  
out.printf("wjx %n %n wwww %n xgsf" );  
//设置java.util.logging库的日志编码格式  
//在日志配置文件中设置java.util.logging.FileHandler.encoding=UTF-8  
//文件的编码格式：  
//1.java 源文件：本地编码  
//2.class 类文件： modified UTF-8  
//3.JVM 虚拟机： UTF-16

8.资源包的各种方法

//获取源文件  
ResourceBundle currentResource = ResourceBundle.*getBundle*("test", locale);  
ResourceBundle currentResource0 = ResourceBundle.*getBundle*("com.test.TestResourceBundle");  
//获取里的属性的值  
String name0 = currentResource.getString("name");  
//获取配置文件的值并转换成数组  
String[] names = currentResource0.getStringArray("array");  
//返回一个枚举对象，枚举出资源包的所有键  
Enumeration<String> enumeration = currentResource0.getKeys();