Jakarta Commons

&

Google Guava

使用说明

（V1.0）

新一站保险代理有限公司 研发中心

2014年7月

版本控制信息

| 版本 | 日期 | 拟稿和修改 | 说明 |
| --- | --- | --- | --- |
| 1.0 | 2014-07-2 | 吕承纲 |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

[前言 4](#_Toc392594402)

[Jakarta Commons 4](#_Toc392594403)

[基础包(lang) 5](#_Toc392594404)

[math包(lang.math) 8](#_Toc392594405)

[bean包(beanutils) 10](#_Toc392594406)

[函数子包(collections.factor) 12](#_Toc392594407)

[集合包(collections) 16](#_Toc392594408)

[应用包(cli, configuration) 22](#_Toc392594409)

[io包(io) 25](#_Toc392594410)

[xml处理包(digester, betwixt) 28](#_Toc392594411)

[Google Guava 30](#_Toc392594412)

[基础包(base) 30](#_Toc392594413)

[集合包(collect) 35](#_Toc392594414)

[缓存包(cache) 39](#_Toc392594415)

[并发包(concurrent) 42](#_Toc392594416)

[事件总线包(eventbus) 46](#_Toc392594417)

[Io包(io) 47](#_Toc392594418)

## 前言

Jakarta Commons和Google Guava分别是Apache软件基金会和Google公司提供的Java开发的工具类及组件. 这两个工具包在我们日常的程序编写过程中经常被使用到, 编写此文档的目的是为了方便研发人员了解并熟练使用常用工具类及组件.

当然, 由于JDK及这两个工具包都在不断的向前发展, 因此有些内容不会包含在此文档中. 这些内容是:

* 已纳入JDK的工具包. 比如枚举(Enum), 文件过滤器(FileFilter)等等
* 从Jakarta Commons中独立出并成为Apache顶级项目, 比如Velocity,FreeMarker等等
* Jakarta Commons中的一些专属工具包, 比如commons-httpclient, commons-fileupload, commons-net等等.

另外, 此说明文档只介绍常用工具类及常用方法的用法, 针对一些需要注意或较为复杂的组件用法,将会以代码或者单元测试用例的方式给出.

本说明文档配套的单元测试代码已在git.oschina.net上进行托管. 可通过如下git命令进行下载测试:

Git clone https://git.oschina.net/xyzdev/chaos.git

## Jakarta Commons

Jakarta Commons是Apache软件基金会(ASF)提供的用于Java开发的辅助工具类及组件. 经过十几年的发展, Jakarta Commons目前已十分的成熟, 提供的工具类及组件也已涵盖到了J2SE开发的各个方面. 目前, Jakarta Commons中的组件按如下规则进行分类:

* The Commons Proper : 稳定可重用的组件分类
* The Commons Sandbox : 处于实验阶段的组件分类,开发者可提交代码
* The Commons Dormant : 还未启动的组件分类

### 基础包(lang)

基础包对JDK中java.lang包中的类提供了方便开发的工具类. 比如Object类的toString(), equals(), hashCode()等方法, 字符串, 数组,日期类等常用操作方法. 常用工具类及方法介绍如下:

#### StringUtils

|  |  |  |
| --- | --- | --- |
| 方法 | 说明 | 备注 |
| isBlank/isNotBlank | 为null或由空白字符组成/不为Null且不由空白字符组成 |  |
| isEmpty/isNotEmpty | 为Null或空字符组成/不为Null且不由空字符组成 |  |
| trimToNull | 删除空白和控制字符,并判断是否为空字符. 是的话返回null |  |
| trimToEmpty | 删除空白和控制字符,并判断是否为空字符. 是的话返回空字符 |  |
| abbreviate | 用省略号进行字符串缩写.如果字符串本身长度小于缩写长度, 则直接返回字符串本身. 否则以省略号(…)替代 | 需注意缩写长度不能太小,否则会抛IllegalArgumentException.  请参考:  StringUtilsTest. testAbbreviate( ) |
| split | 字符串分割. 功能和String.split( )方法类似 |  |
| substringBetween | 截取字符串的中间部分 |  |
| substringBeforeLast | 截取字符串最后一个匹配字符之前的部分 |  |
| substringAfterLast | 截取字符串最后一个匹配字符之后的部分 |  |
| strip | 删除字符串中指定的字符 | 请参考:  StringUtilsTest.testStrip( ) |
| chomp | 删除换行字符. 包括”\r”,”\n”,”\r\n” |  |
| repeat | 字符复制 |  |
| center | 字符串居中 |  |
| join | 字符串数组合并 |  |
| isNumeric | 判断字符串是否由数字组成 |  |
| isAlpha | 判断字符串是否由unicode字母组成 |  |
| isAlphanumeric | 判断字符串是否由字母数字组成 |  |
| isAlphaSpace | 判断字符串是否由字母空格组成 |  |
| getLevenshteinDistance | 获取两个字符串之间字符差异的个数 | 请参考:  StringUtilsTest.testDifferece( ) |
| difference | 两个字符串比较.并返回第二个字符串从第一个不相同字符开始的部分 | 请参考:  StringUtilsTest.testDifferece( ) |
| indexOfDifference | 两个字符串比较.并返回第二个字符串第一个不相同字符的位置 | 请参考:  StringUtilsTest.testDifferece( ) |

#### ArrayUtils

|  |  |  |
| --- | --- | --- |
| 方法 | 说明 | 备注 |
| clone | 数组复制 |  |
| contains | 判断元素是否在数组中 |  |
| indexOf/lastIndexOf | 判断第一个元素/最后一个元素在数组中的位置 |  |
| reverse | 数组反转 |  |
| toMap | 将二维数组转换为Map对象 |  |

#### ToStringBuilder, EqualsBuilder, HashcodeBuilder

这三个工具类对类Object中的toString( ), equals( ), hashCode( )方法实现起辅助支撑作用. 采用Fluent Style方式,可以让我们用很少的代码快速实现这些方法.

请参考:

PoliticalCandidate

State

ToStringTest

#### CompareToBuilder

这个工具类对Comparable接口的compareTo( )方法实现起辅助支撑作用. 采用Fluent Style方式,可以让我们用很少的代码快速实现这个方法.

请参考:

PoliticalCandidate

CompareToTest

#### DateFormatUtils

DateFormateUtils用于处理Date或Calendar对象到字符串对象之间的转换. 与JDK中的SimpleDateFormat相比, 它是线性安全的. 另外, DataFormatUtils中提供了许多默认的解析格式.

请参考:

DateFormatUtilsTest

#### DateUtils

DateUtils用于Date或Calendar对象处理,或者将日期格式字符串转换为Date对象

|  |  |  |
| --- | --- | --- |
| 方法 | 说明 | 备注 |
| round | 日期四舍五入 |  |
| truncate | 日期截取 |  |
| parseDate | 将日期格式字符串转换为Date对象 |  |

#### StopWatch

StopWatch是一个很方便的计时器. 通常我们统计一段代码的执行耗时,可能会这么写:

long start=System. currentTimeMillis();

doSomething()

long end=System.currentTimeMillis();

System.out.println(“time elapse:”+(end-start));

有了StopWatch后,代码会变得更为简洁和直观,功能也更加强大

|  |  |  |
| --- | --- | --- |
| 方法 | 说明 | 备注 |
| start | 启动定时器 |  |
| suspend | 挂起定时器 |  |
| resume | 恢复计数器 |  |
| stop | 停止计数器 |  |
| reset | 重置计数器 |  |
| getTime | 获取计时时间(毫秒级) |  |
| getNanoTime | 获取计时时间(纳秒级) |  |

### math包(lang.math)

math包提供了科学计算的功能. 虽然,我们在应用中应该避免复杂的数学计算, 但此包中的某些工具类我们还是可能会用到.

#### Fraction

Fraction提供了分数处理的功能.

#### NumberUtils

Number作为Java数字类型的基本类型, NumberUtils也提供了大量的工具方法

|  |  |  |
| --- | --- | --- |
| 方法 | 说明 | 备注 |
| min | 取出一组数据中的最小值 |  |
| max | 取出一组数据中的最大值 |  |
| isNumber | 判断是否为数字 |  |
| createInteger  createDouble  createFloat  createBigDecimal  createBigInteger | 将字符串转换为相应数字包装(Wrapper)类型 |  |
| toShort  toInt  toLong  toDouble | 将字符串转换为相应数字原生(Primitive)类型 |  |

#### RandomUtils

RandomUtils是一个随机数生成器. 和JDK中的Random不同, RandomUtils默认采用JVM\_RANDOM,其随机种子(seed)不是固定的, 这样可以保证每次程序运行生成的随机数不同.

请参考:

RandomUtilsTest

### bean包(beanutils)

bean包大量使用Java反射机制来处理对JavaBean对象的操作, 使开发者不需要详细了解反射原理就可以方便,快速的对bean进行操作.

#### PropertyUtils

PropertyUtils用于对Bean属性对象的操作.

|  |  |  |
| --- | --- | --- |
| 方法 | 说明 | 备注 |
| getSimpleProperty | 获取bean的简单属性对象 |  |
| getNestedProperty | 获取bean内嵌属性对象的属性 | 比如,对象的内部结构为:  A{  B{  C c;  };  };  可通过PropertyUtils.getNestedProperty(a,”b.c”);来获取C的值 |
| getIndexedProperty | 如果bean属性对象为一个集合, 可通过这种方式获取集合中的某个元素 | 请参考:  PropertyUtilsTest. testGetIndexProperty( ) |
| getMapProperty | 如果bean属性对象为Map对象, 可通过这种方式获取Map中的某个元素 | 请参考:  PropertyUtilsTest. testGetMapProperty( ) |
| getProperty | 这是个获取bean属性的通用方法,上述几种方式bean属性对象的获取均可以采用这种方式 | 请参考:  PropertyUtilsTest. testGetProperty( ) |
| getPropertyType | 获取bean属性对象的类型 |  |
| copyProperties | Bean属性拷贝 |  |
| setProperty | 这是给bean属性对象设置的通用方法 | 请参考:  PropertyUtilsTest. testSetProperty( ) |
| isReadable/isWritable | Bean属性对象是否可读/可写 |  |
| describe | 将bean转换为Map对象 |  |

#### DynaBean

DynaBean提供了动态Bean的生成方法. 我们甚至不需要有Bean的定义即可动态的创建一个Bean. 比如, 如下的代码:

DynaProperty[] beanProperties = new DynaProperty[] {

new DynaProperty("name", String.class),

new DynaProperty("favoriteColor", String.class) };

BasicDynaClass authorClass = new BasicDynaClass("Author",

BasicDynaBean.class, beanProperties);

构建了Author类,内部包含String类型的name和String类型的favoriteColr两个属性.

请参见:

DynaBeanTest

#### Converter, ConverterUtils

当我们查看BeanUtils.getProperty( )方法源码时, 会发些如下语句:

return (getConvertUtils().convert(value));

这是由于BeanUtils.getProperty返回String类型对象,而Bean的属性对象可能是任意类型的, 因此存在任意类型到String类型的转换.

commons-beanutils 包中提供了Converter接口用于对象转换,以及ConverterUtils工具类用于Converter对象的注册(register), 查找(lookup)和转换(convert)

针对Converter接口,commons-beanutils中还提供了许多默认的实现类, 包括: BigDecimalConverter, BigIntegerConverter, ShortConverter, IntegerConverter, LongConverter, FloatConverter, DoubleConverter, StringConverter等等

### 函数子包(collections.factor)

函数子包是函数式编程的一种方式. JDK中也有函数子接口, 比如我们熟悉的Iterator, Compator接口等等, 均属于函数子.

Jakarta Commons中提供了三个函数子接口,以及JDK函数子的一些有用实现类, 均用于集合类处理中, 它们是:

#### Predicate

Predicate接口定义了一个接口方法:

public boolean evaluate(Object object);

它类似于一个过滤器, 只有在评估通过后, 才可以继续后面的逻辑.

Jakarta commons中提供了许多默认的Predicate实现类, 它们是:

|  |  |  |
| --- | --- | --- |
| 类 | 说明 | 备注 |
| EqualPredicate | 判断评估对象是否和已有对象相同 |  |
| InstanceOfPredicate | 判断评估对象类型是否和已有对象类型相同 |  |
| TruePredicate  FalsePredicate | 总是评估通过  总是评估不通过 |  |
| NullPredicate  NotNullPredicate | 评估对象为空才通过  评估对象不为空才通过 |  |
| UniquePredicate | 评估对象不重复才通过 |  |
| TransformerPredicate | 先调用Transformer.execute( )方法进行对象转换,转换后的对象如果不是Boolean对象,则抛出异常, 否则返回Boolean对象的值 |  |
| TransformedPredicate | 先调用Transformer.execute( )方法进行对象转换, 再评估转换的对象是否通过 |  |
| OrPredicate  AndPredicate | 两个评估条件只要有一个通过即通过  两个评估条件必须全部通过才通过 |  |
| OnePredicate  AllPredicate | 多个评估条件只要有一个通过即通过  多个评估条件必须全部通过才通过 |  |

#### Transformer

Transformer接口定义了一个接口方法:

public Object transform(Object input);

它起到对象转换的作用.Jakarta commons中提供了许多默认的Transformer实现类, 它们是:

|  |  |  |
| --- | --- | --- |
| 类 | 说明 | 备注 |
| CloneTransformer | 复制原有对象并返回新对象 |  |
| ClosureTransformer | 执行Closure接口的execute( )方法, 并返回原有对象 |  |
| ConstantTransformer | 返回原有对象 |  |
| ExceptionTransformer | 返回FunctorException异常对象 |  |
| FactoryTransformer | 返回Factory.create( ) 创建出来的对象 |  |
| InstantiateTransformer | 实例化并返回Class对象的实例对象 |  |
| InvokerTransformer | 调用并返回Method方法返回的对象 |  |
| MapTransformer | 返回Map对象所包含的对象 |  |
| NOPTransformer | 直接返回输入对象 |  |
| PredicateTransformer | 当Predicate评估为true时,返回Boolean.TRUE, 否则返回Boolean.FALSE |  |
| StringValueTransformer | 返回对象的字符串表示 |  |
| ChainedTransformer | 链式一个一个转换 |  |
| SwitchTransformer | 分支转换. 当通过某个位置的Predicate评估后, 进行这个位置的Transformer转换 | 请参见:  Transformer. testConditionalTransformer( ) |

#### Comparator

Comparator是JDK中提供的函数子, 但Jakarta Commons中提供了一些这个接口的实现类. 它们是:

|  |  |  |
| --- | --- | --- |
| 类 | 说明 | 备注 |
| BooleanComparator | Boolean对象比较器 |  |
| NullComparator | Null值对象比较器 | 此类有一个属性: nullsAreHigh, 为true的情况下, Null值对象会排在前面, 为false,Null值对象排在后面 |
| ReverseComparator | 反向比较器 |  |
| TransformingComparator | 先进行对象转换,转换后的对象再进行比较 |  |
| FixedOrderComparator | 保证作为构造函数参数的数据不会进行排序 |  |
| ComparatorChain | 比较器链 |  |

#### Iterator

Iterator是JDK中提供的函数子, 但Jakarta Commons中提供了一些这个接口的实现类. 它们是:

|  |  |  |
| --- | --- | --- |
| 类 | 说明 | 备注 |
| ArrayIterator | 数组迭代器 |  |
| ArrayListIterator | ArrayList迭代器 |  |
| CollatingIterator | 校对迭代器.迭代取值时返回一组迭代器中的最小值 |  |
| EmptyIterator | 空迭代器 |  |
| FilterIterator | 过滤迭代器. 先评估,再返回元素 |  |
| IteratorChain | 迭代器链.元素按迭代器添加顺序一个一个返回 | 和CollatingIterator不同, CollatingIterator只返回当前位置最小值的那个迭代器元素. 并不是每个迭代器的每个元素都会返回 |
| LoopingIterator | 循环迭代器 | 需注意死循环问题 |
| ReverseListIterator | 反向迭代器 |  |
| SingletonIterator | 单元素迭代器 |  |
| TransformIterator | 转换迭代器. 元素获取后进行对象转换 |  |
| UniqueFilterIterator | 元素不重复迭代器 |  |
| UnmodifiableIterator | 不可变迭代器. | 不支持remove操作 |

#### Closure

Closure类似于Callback函数,它含有一个接口方法:

public void execute(Object input);

可以用于对结果对象再处理. Jakarta Commons中提供了许多默认的Closure实现类.

|  |  |  |
| --- | --- | --- |
| 类 | 说明 | 备注 |
| ExceptionClosure | 返回FunctorException异常对象 |  |
| NOPClosure | 什么都不做 |  |
| TransformerClosure | Closure执行Transformer转换 |  |
| ChainedClosure | 链式一个一个执行 |  |
| IfClosure | 当Predicate评估通过,执行第一个Closure,否则执行第二个Closure |  |
| SwitchClosure | 分支执行. 当通过某个位置的Predicate评估后, 进行这个位置的Closure执行 |  |
| ForClosure | For循环Closure执行 |  |
| WhileClosure | 循环Predicate评估, 只要评估通过,就执行Closure | 支持do…while和while两种循环方式  请参见:  ClosureTest.testWhileClosure |

### 集合包(collections)

集合包主要针对JDK的集合类提供辅助工具作用.

#### CollectionUtils

|  |  |  |
| --- | --- | --- |
| 方法 | 说明 | 备注 |
| union | 并集. 返回集合A中所有元素和在B中存在而A终不存在的元素 | 请参见:  CollectionUtils.testSetOperation( ) |
| intersection | 交集. 返回集合A,B中均存在的元素 | 请参见:  CollectionUtils.testSetOperation( ) |
| disjunction | 补集.相当于union-intersection | 请参见:  CollectionUtils.testSetOperation( ) |
| subtract | 差集.返回只在集合A中存在而不在B中存在的元素 | 请参见:  CollectionUtils.testSetOperation( ) |
| containsAny | 如果两个集合均含有某个元素,返回true;否则返回false |  |
| getCardinalityMap | 返回Map对象. 可以为集合元素,value为元素个数 |  |
| isSubCollection  isProperSubCollection | 如果集合A的元素在集合B中均存在,返回true;否则返回false  如果集合A的元素在集合B中均存在且A的长度小于B的长度,返回true;否则返回false |  |
| isEqualCollection | 如果集合A的元素及此元素的个数和集合B的元素及此元素的个数相同,返回true;否则返回false |  |
| cardinality | 返回元素在集合中的个数 |  |
| find | 利用Predicate查找第一个评估通过的元素 |  |
| forAllDo | 利用Closure做集合元素处理 |  |
| filter | 利用Predicate做集合元素过滤 |  |
| transform | 利用Transformer做集合元素转换 |  |
| countMatches | 评估通过Predicate的元素个数 |  |
| exists | 只要集合中有一个元素通过评估, 返回true;否则返回false |  |
| select  selectRejected | 返回一个新的集合,元素为Predicate评估通过的元素  返回一个新的集合,元素为Predicate评估不通过的元素 |  |
| collect | 返回新的集合,集合元素经过Transformer转换 |  |
| addAll | 元素添加到集合 |  |
| index | 获取集合中某个位置的元素 |  |
| isEmpty / isNotEmpty | 是否为/不为 Null或空值 |  |
| reverseArray | 数组反转 |  |
| isFull | 针对有边界的集合判断是否还能添加元素 |  |

#### List

JDK中我们常用的List实现类有ArrayList和LinkedList. 他们的区别是:

* ArrayList以动态数组实现集合,非常适合于元素的快速访问及尾部插入删除操作, 但对于中间或随机的元素插入或删除性能则很差,这是由于需要移动后面的数组元素
* LinkedList以链表的形式实现集合,对于元素的顺序插入或删除,物流是在头部或尾部速度都很快,因为只需要修改节点索引就可以了, 但对于元素的访问很慢, 这是由于所有的访问均要从第一个节点开始

Jakarta Commons针对JDK中的List另外提供了许多扩展类. 他们是:

|  |  |  |
| --- | --- | --- |
| 类 | 说明 | 备注 |
| FixedSizeList | 不允许进行add/remove操作 |  |
| GrowthList | 包装一个List并当List元素个数达到size值是仍可无缝增长 | 对于某些个数限定或不支持添加元素的List或抛出异常 |
| LazyList | 懒加载List. 对于获取Null值的元素或在List中不存在的元素,会通过Factory来创建. | 不适用于获取List中的Null值元素 |
| PredicateList | 添加或修改元素前会先评估 |  |
| SetUniqueList | List中元素不允许重复 |  |
| TransformedList | 添加或修改元素前会先转换 |  |
| TreeList | TreeList对于随机的元素插入,删除性能很高 |  |
| UnmodifiableList | 包装并返回一个不可变List |  |

#### Set

JDK中我们常用的Set实现类有HashSet, LinkedHashSet和TreeList. 他们的区别是:

* HashSet利用Hash算法来定位元素添加到集合中的位置,因此不保证元素的添加顺序,可添加Null值
* LinkedHashSet虽然也是利用Hash算法定位元素添加到集合中的位置,但同时还通过链表维护元素的次序.这样元素访问的时候保证是按元素的插入顺序
* TreeSet则根据Compartor或Comparable接口进行元素的排序, 且不允许元素为null

Jakarta Commons针对JDK中的Set另外提供了许多扩展类. 他们是:

|  |  |  |
| --- | --- | --- |
| 类 | 说明 | 备注 |
| CompisiteSet | 包装并维护Set集合的集合 |  |
| ListOrderSet | 包装并维护Set为List,保证元素添加顺序 |  |
| MapBackedSet | 包装并维护Map,使Map具有Set的特性 |  |
| PredicatedSet | 添加元素时先进行评估 |  |
| SynchronizedSet | 线程安全Set |  |
| TransformedSet | 添加或修改元素前会先转换 |  |
| UnmodifiableSet | 包装并返回一个不可变List |  |

#### Map

JDK中我们常用的Map实现类有HashMap, HashTable, ConcurrentHashMap和TreeMap. 他们的区别是:

* HashMap运行key或value为null值, 线程不安全
* HashTable不允许key或value为null值, 线程安全, 但实现为方法级别锁,代价高
* ConcurrentHashMap不允许key或value为null值, 线程安全且由于采用锁分离技术, 代价底
* TreeMap用于Map根据key进行排序

Jakarta Commons针对JDK中的Map另外提供了许多扩展类. 他们是:

|  |  |  |
| --- | --- | --- |
| 类 | 说明 | 备注 |
| CaseInsensitiveMap | Map中的key大小写不区分,一律转换为小写 |  |
| CompositeMap | 包装并维护Map集合的集合 |  |
| DefaultMap | 包装一个Map集合,并当Map中的元素不存在时返回一个默认值 |  |
| FixedSizeMap | 包装一个Map,并只能对Map中的元素修改, 而不能添加和删除 |  |
| IdentityMap | 当两个Map元素进行比较时,基于等号(==)而不是equals( )方法 |  |
| LazyMap | 当访问Map元素不存在时,会通过Factory创建 |  |
| LRUMap | 一个固定数量的Map. 当超过数量最大值并继续进行添加操作时,根据LRU(Least Recent Used)规则删除元素 |  |
| MultikeyMap  MultiValueMap | Map中的key允许由多个元素组成  Map中的value为一个集合 | 不推荐使用 |
| PredicatedMap | Map元素添加或修改前先进行评估 |  |
| ReferenceMap | 内存占满进行gc时,会被回收 |  |
| SingletonMap | Map中只有一个元素 |  |
| TransformedMap | Map元素添加或修改前先进行转换 |  |
| UnmodifiableMap | 包装并返回一个不可变List |  |

#### Bag

Bag是Jakarta Commons中新添的类型. 它继承自Collection接口, 允许一个元素的重复添加以及获取此元素的个数.请注意Bag接口的add, remove, removeAll, retainAll方法含义和Collection接口中相应的方法含义不同

请参见:

BatTest

#### Buffer

Buffer是Jakarta Commons中新添的类型. 它继承自Collection接口,允许集合中元素删除按照指定的排序方式删除.

请参见:

BufferTest

### 应用包(cli, configuration)

#### CLI

Commons-cli (Command-Line Interface)是Jakarta Commons对命令行参数进行解析的工具包.对于命令行的解析分为三个阶段:

* 定义阶段(Definition Stage): 定义命令行中应该有哪些参数选项; 哪些参数选项是需要有参数值,而哪些是不需要的; 参数是不是互斥的等等
* 解析阶段(Parsing Stage): 通过CommandLineParser对实际参数进行解析, 并返回CommandLine对象
* 询问交互阶段(Interrogation Stage): 通过对CommandLine对象进行分析, 来判断输入的参数是否正确. 如果不正确, 可输出相应的提示信息

常用的类有:

|  |  |  |
| --- | --- | --- |
| 类 | 说明 | 备注 |
| Option | 一个Option对应一个定义参数 |  |
| OptionGroup | Option分组.分组中的Option互斥(既不允许同时出现) |  |
| Options | 是Option的集合. 代表所有定义参数 |  |
| CommandLineParser | 命令行参数解析接口 |  |
| BasicParser | 基本解析器 | 不支持参数字符串token和-- |
| GnuParser | Gnu解析器 | 比如支持ANT参数-buildfile和Java系统参数-Dant.home=c:\ant |
| PosixParser | Posix解析器 | 比如支持分解tar参数-zxf为-z –x –f |
| CommandLine | 解析结果.用于询问交互阶段分析 |  |
| HelpFormatter | 用于格式化用法实例输出 |  |

示例代码如下:

String USAGE = "[-h] [-v] [-f <file> | -m <email>]";

String HEADER = "SomeApp - A fancy and expensive program, Copyright 2010 XYZ.";

String FOOTER = "For more instructions, see our website at: <http://www.xyz.cn>";

CommandLineParser parser = new BasicParser();

Options options = new Options();

options.addOption("h", "help", false, "Print this usage information");

options.addOption("v", "verbose", false,

"Print out VERBOSE information");

OptionGroup optionGroup = new OptionGroup();

optionGroup.addOption(OptionBuilder.hasArg(true).withArgName("file")

.withLongOpt("file").create('f'));

optionGroup.addOption(OptionBuilder.hasArg(true).withArgName("email")

.withLongOpt("email").create('m'));

options.addOptionGroup(optionGroup);

try {

CommandLine commandLine = parser.parse(options, new String[] { "-f" });

} catch (Exception e) {

HelpFormatter helpFormatter = new HelpFormatter();

helpFormatter.setWidth(80);

helpFormatter.printHelp(USAGE, HEADER, options, FOOTER);

}

请参见:

CliTest

#### Configuration

commons-configuration是Jakarta Commons对多种来源的数据进行解析的工具包. 其核心接口为Configuration.

|  |  |  |
| --- | --- | --- |
| 方法 | 说明 | 备注 |
| subset | 返回去除前缀的Configuration对象 | 如果属性Configuration包含如下内容:  prefix.name=lcg  prefix.gender=0  则subset(“prefix”)后得到的configuration内容为:  name=lcg  gender=0 |
| isEmpty | 是否为空 |  |
| containsKey | 是否含有key |  |
| addProperty  setProperty  clearProperty | 添加属性  修改属性  清除属性 |  |
| clear | 清除全部属性 |  |
| getProperty | 获取属性 |  |
| getKeys | 获取key |  |
| getBoolean  getByte  getDouble  getFloat  getInt  getInteger  getLong  getShort  getBigDecimal  getBigInteger  getString | 获取并转换为Boolean值  获取并转换为Byte值  获取并转换为Double值  获取并转换为Float值  获取并转换为primitive int值  获取并转换为Integer值  获取并转换为Long值  获取并转换为Short值  获取并转换为BigDecimal值  获取并转换为BigInteger值  获取并转换为String值 |  |
| getList | 获取并转换为List集合 |  |

Jakarta Commons中还提供了许多Configuration的实现类, 用于处理不同来源和格式的数据, 它们是:

|  |  |  |
| --- | --- | --- |
| 类 | 说明 | 备注 |
| PropertiesConfiguration | 处理属性文件 |  |
| XMLConfiguration | 处理XML文件 | 支持XPath, 使用起来很方便 |
| INIConfiguration | 处理Windows INI文件 |  |
| JNDIConfiguration | 处理JNDI配置数据 | 只读,不能修改 |
| DatabaseConfiguration | 处理数据库数据 | 不适用.估计没人会通过这个类访问数据库 |
| SystemConfiguration | 处理系统参数 |  |
| EnvironmentConfiguration | 处理环境变量参数 | 只读 |

### io包(io)

commons-io是Jakarta Commons对IO处理提供的工具包. 包括六部分的内容, 分别为:

* 工具类 – 主要的类有FileUtils和IOUtils
* 输入流 – 提供了多种InputStream或Reader的实现类
* 输出流 – 提供了多种OutputStream或Writer的实现类
* 过滤器 – 提供了各种文件过滤器
* 比较器 – 提供了各种文件比较器,用于排序
* 文件时间监听器 – 各种对文件操作的事件监听

主要类介绍如下:

#### FileUtils

|  |  |  |
| --- | --- | --- |
| 方法 | 说明 | 备注 |
| getFile | 在目录下创建一堆文件并返回目录 |  |
| getTempDirectoryPath  getTempDirectory | 返回系统临时目录路径  返回系统临时目录 | Windows 临时目录: ~\Loal Settings\Temp  Unix 临时目录: /tmp 或/var/tmp |
| getUserDirectoryPath  getUserDirectory | 返回用户目录路径  返回用户目录 |  |
| openInputStream  openOutputStream | 打开输入流  打开输出流 |  |
| touch | 若文件不存在,则创建;否则修改文件的LastModified属性 |  |
| listFiles  listFilesAndDirs | 根据过滤器递归搜索文件  根据过滤器递归搜索文件和子目录 |  |
| copyFileToDirectory | 将文件拷贝至目录下 |  |
| copyFile | 拷贝文件 |  |
| copyDirectoryToDirectory | 拷贝目录至某一目录下 |  |
| copyDirectory | 拷贝目录 |  |
| deleteDirectory | 递归目录删除 | 可能抛出异常 |
| delereQuietly | 文件递归删除 | 不会抛出异常 |
| directoryContains | 判断文件是否在目录下 |  |
| cleanDirectory | 递归删除目录内的文件和子目录 |  |
| readFileToString  readFileToByteArray | 将文件内容读取到String中  将文件内容读取到ByteArray中 |  |
| readLines  writeLines | 将文件内容读取到List<String>集合中  将集合内容写入到文件中 |  |
| writeStringToFile  writeByteArrayToFile | 将String写入到文件中  将ByteArray写入到文件中 |  |
| sizeof | 返回文件长度 |  |
| isFileOlder  isFileNewer | 判断文件新旧 |  |
| moveFile  moveDirectory  moveFileToDirectory  moveDirectoryToDirectory | 文件移动  目录移动  将文件移动到某个目录下  将目录移动到某个目录下 |  |
| isSymlink | 判断文件是否为符号链接 | Windows系统总返回false |

#### IOUtils

|  |  |  |
| --- | --- | --- |
| 方法 | 说明 | 备注 |
| close | 关闭HttpURLConnection |  |
| closeQuietly | 关闭各种流 | 不会抛出异常 |
| toBufferedInputStream  toBufferedReader | 包装为BufferedInputStream  包装为BufferedReader |  |
| toByteArray  toCharArray  toString | 将各种输入流数据输入到ByteArray中  将各种输入流数据输入到CharArray中  将各种输入流数据输入到字符串中 |  |
| readLines  writeLines | 从各种输入流读取一行数据  从各种输出流输出一行数据 |  |
| toInputStream | 将字符或字符串输入到输入流 |  |
| write | 将数据输出到各种输出流中 |  |
| copy | 将输入流数据拷贝到输出流 |  |
| skip  skipFully | 从输入流中跳过一些字符  从输入流中跳过一些字符,如果输入流可用字符数小于跳过字符数,抛出异常 |  |
| read  readFully | 从输入流读取数据  从输入流读取数据,如果读取数据长度小于要求长度, 则抛出异常 |  |

### xml处理包(digester, betwixt)

#### digester

digester最早是Struts的一部分, 后来被分离出来并加入到Jakarta Commons中,形成今天的commons-digester子包. 它是Jakarta Commons中一款非常优秀的处理XML到JavaBean的工具,机理仍然使用的基于事件的SAX方式来处理XML文件.

使用Digester进行XML到JavaBean对象转换的基本步骤为:

* 创建Digester对象实例
* 设置该Digester对象属性(比如namespaceAware, namespaceURI等等),这一步可选
* 将初始对象push到Digester对象的对象栈上(初始对象为我们想要得到的结果对象.如果rule中定义了ObjectCreateRule,则此步骤可以省略)
* 注册所有的XML元素匹配模式与处理规则之间的映射关系(可通过XML或Java API方式建立)
* 调用digester.parse( )方法进行XML解析并返回javaBean对象

常用的类有:

|  |  |  |
| --- | --- | --- |
| 类 | 说明 | 备注 |
| Rule | 规则类. 用于定义XML元素如何转换 |  |
| ObjectCreateRule | 当元素结束标签被触发时, 就创建一个新的对象 |  |
| FactoryCreateRule | 当元素结束标签被触发时,由Factory创建对象 |  |
| SetPropertiesRule | 当元素结束标签被触发时,将标签值设置到JavaBean属性中 |  |
| SetNextRule | 当元素结束标签被触发时,digester将栈顶元素设置进次栈顶元素中 | 建立父子关系 |
| SetTopRule | 当元素结束标签被触发时,digester将次栈顶元素设置进栈顶元素中 | 建立子父关系 |
| SetRootRule | 当元素结束标签被触发时,digester将元素设置为root元素 | 设置顶层元素 |
| CallMethodRule  CallParamRule | 这两个对象往往配合使用. 当元素结束标签被触发时,执行指定的方法, 方法参数由CallParamRule给出 |  |

请参见:

DigesterTest

#### betwixt

commons-betwixt 是另一款Jakarta Commons提供的JavaBean和XML转换器. 它既可以处理XML到JavaBean的转换, 但更多的是用来处理JavaBean到XML的转换.其核心类为BeanReader和BeanWriter.

由于betwixt在日常情况下使用不多, 这里就不详细介绍了.有兴趣的话可以去阅读它的API Doc

请参见:

BetwixtTest

## Google Guava

Google Guava是Google为Java开发人员提供的基础工具包. 和Jakarta Commons不同, 它的发展历史比较短(源于2007年的Google Collections Library), 因此里面的子包并没有被拆分开来. 但同样, Guava涵盖了Java开发的方方面面. 尤其Guava强调Fluent Style的方式,使得代码编写十分的简洁.

Google Guava和Jakarta Commons相辅相成, 但也有工具类功能相同, 存在冲突关系. 我们建议在这种情况下优先考虑使用Jakarta Commons.

Google Guava中有些类并不稳定, 可能会在将来的版本中做出重大调整, 这些类以@Beta注解标识, 应用中应尽量避免使用这些类.

### 基础包(base)

基础包对JDK中java.lang包中的类提供了方便开发的工具类. 另外, 用于集合包中的函数子也包括在基础包中. 常用的类有:

#### Strings

|  |  |  |
| --- | --- | --- |
| 方法 | 说明 | 备注 |
| commonPrefix  commonSuffix | 返回两个字符串的共有最长前缀  返回两个字符串的共有最长后缀 |  |
| emptyToNull  nullToEmpty | 字符串为空返回Null值  字符串为Null返回空值 |  |
| padEnd  padStart | 尾部用给定的字符补足  头部用给定的字符补足 |  |
| repeat | 返回重复n次的字符串 |  |

#### Joiner

典型用法示例:

String res=Joiner.on(“|”).skipNulls().join(new String[]{“aaa”,”bbb”,”ccc”});

Assert.assertThat(res, is(“aaa|bbb|ccc”));

|  |  |  |
| --- | --- | --- |
| 方法 | 说明 | 备注 |
| on | 设定分割符 |  |
| appendTo | 内容输出 |  |
| join | 字符串合并 |  |
| skipNulls | 输出时忽略null值 |  |
| useForNull | 输出时用给定值替换null值 |  |

请参见:

JoinerTest

#### Splitter

典型用法示例:

Iterator<String> strIter =Splitter.on(“|”).omitEmptyStrings().split(“aaa|bbb|ccc”);

|  |  |  |
| --- | --- | --- |
| 方法 | 说明 | 备注 |
| on | 设定分割符 |  |
| omitEmptyStrings | 忽略空白字符 |  |
| split | 字符串分割,返回Iterator迭代器 |  |
| splitToList | 字符串分割,返回List集合 |  |
| trimResults | 清除分割后字符串开头和结尾的空白字符 |  |
| skipNulls | 输出时忽略null值 |  |
| useForNull | 输出时用给定值替换null值 |  |

#### Predications

Predications类似于Assert断言. 主要方法有:

|  |  |  |
| --- | --- | --- |
| 方法 | 说明 | 备注 |
| checkArgument | 参数bool值判断 | 失败抛出IllegalArgumentException |
| checkElementIndex  checkPositionIndex | 数组,集合等越界判断 |  |
| checkNotNull | 参数非空判断 |  |
| checkState | 参数状态bool值判断 | 失败抛出IllegalStateException |

#### Function

Function是一个对象转换的函数子, 和Jakarta Commons中的Transformer功能一致.其内部重要的接口方法为:

@Nullable T apply(@Nullable F input);

Function一般和集合工具类或Functions加集合工具类配合使用, 来达到集合元素转换的目的.典型用法实例如下:

//定义Date🡪String转换器

DateFormatFunction dateFormatFunction = new DateFormatFunction(format);

//定义Long🡪Date转换器

LongToDateFunction longToDateFunction = new LongToDateFunction();

SimpleDateFormat dateFormat = new SimpleDateFormat(format);

Date date = dateFormat.parse("18/11/2003");

long time = date.getTime();

Function<Long,String> composedFunction =

Functions.compose(dateFormatFunction,longToDateFunction);

assertThat(composedFunction.apply(time),is("18/11/2003"));

#### Functions

Functions是Function函数子的工具类, 其内部定义了一些默认Function的实现.其主要方法有:

|  |  |  |
| --- | --- | --- |
| 方法 | 说明 | 备注 |
| compose | 组合两个Function,形成Function链 | 第二个Function参数执行在前,第一个Function参数执行在后 |
| constant | 返回ConstantFunction. | ConstantFunction不进行转换 |
| forMap | 将Map的key和Value作为转换的输入输出对象 |  |
| forPredicate | 将Predicate对象转换为Function对象 | 即转换结果为Predicate评估结果 |
| forSupplier | 将Supplier对象转换为Function对象 | 即转换结果为Supplier生成结果 |

#### Predicate

Predicate是一个对象过滤器的函数子, 和Jakarta Commons中的Predicate功能一致.其内部重要的接口方法为:

boolean apply(@Nullable T input);

Predicate一般和集合工具类或Predicates加集合工具类配合使用, 来达到集合元素过滤的目的.典型用法实例如下:

//定义第一个过滤器

LowRainfallPredicate dryPredicate = new LowRainfallPredicate();

//定义第二个过滤器

SmallPopulationPredicate smallPopulationPredicate = new SmallPopulationPredicate();

//定义”与”过滤器链

Predicate<City> smallAndDry = Predicates.and(dryPredicate, smallPopulationPredicate);

City smallWetCity = new City.Builder().averageRainfall(56.87).population(400000).build();

assertThat(smallAndDry.apply(smallWetCity),is(false));

City smallDryCity = new City.Builder().averageRainfall(25.63).population(450000).build();

assertThat(smallAndDry.apply(smallDryCity),is(true));

#### Predicates

Predicates是Predicate函数子的工具类, 其内部定义了一些默认Predicate的实现.其主要方法有:

|  |  |  |
| --- | --- | --- |
| 方法 | 说明 | 备注 |
| alwaysFalse  alwaysTrue | 返回评估始终不通过Predicate  返回平台始终通过Predicate |  |
| and | 返回AND关系的组合Predicate |  |
| compose | 返回CompositionPredicate | 先执行Function转换,再执行Predicate评估 |
| contains | 返回一正则匹配的Predicate |  |
| in | 返回一判断元素是否在集合中的Predicate |  |
| instanceOf | 返回一判断某个对象是某一类型的Predicate |  |
| isNull  notNull | 返回一判断某个对象为null/不为null的Predicate |  |
| not | 返回反向判断的Predicate |  |
| or | 返回OR关系的组合Predicate |  |

#### Supplier

Supplier是一个对象创建的函数子, 类似于工厂类.其内部重要的接口方法为:

T get( );

Supplier一般和集合工具类或Suppliers加集合工具类配合使用, 来达到集合元素构建的目的.典型用法实例如下:

//构建MemoizingSupplier, 缓存构建出来的对象

Supplier<List<Book>> cachedSupplier = Suppliers.memoize(bookListSupplier);

//将构建出来的对象通过Function转换

bookMapSupplier = Suppliers.compose(function, cachedSupplier);

//获取构建对象, 此对象可能是即时构建出来的, 也可能是缓存的, 并经过了转换

Map<String, String> bookMap = bookMapSupplier.get();

#### Suppliers

Suppliers是Supplier函数子的工具类, 其内部定义了一些默认Supplier的实现.其主要方法有:

|  |  |  |
| --- | --- | --- |
| 方法 | 说明 | 备注 |
| compose | 返回SupplierComposition | 先进行对象构建, 再通过Function进行对象转换 |
| memoize | 返回MemoizingSupplier | 会缓存构建出来的对象 |
| memoizeWithExpiration | 返回ExpiringMemoizingSupplier | 只会缓存有限的时间. 过期后会重新构建并缓存 |
| ofInstance | 返回SupplierOfInstance | 构建对象即为输入参数本身 |
| supplierFunction | 将Supplier转换为Function |  |

### 集合包(collect)

集合包主要针对JDK的集合类提供辅助工具作用.主要的工具类有:

#### Collections2

|  |  |  |
| --- | --- | --- |
| 方法 | 说明 | 备注 |
| constainsAllImpl | 判断一个集合是否为另一个集合的子集 |  |
| filter | 利用Predicate进行集合元素过滤 |  |
| isPermutation  permutations | 判断一个集合是否为另一个集合的排列组合/返回一个集合的排列组合 | 集合A: [“1”,”2”,”3”]  集合B:[“1”,”2”]  isPermutation(A,B)返回true,而集合B的排列组合方式有3种  不常用 |
| isPositiveInt | 判断是否为正整数 |  |
| safeContains  saveRemove | 安全的判断是否含有/删除某个元素 | 不会抛出异常 |
| transformer | 利用Function进行集合元素转换 |  |

#### Lists

|  |  |  |
| --- | --- | --- |
| 方法 | 说明 | 备注 |
| addAllImpl | 将第二个list从index位置开始加入到第一个list中 |  |
| cartesianProduct | 返回笛卡尔积组合List | 不常用 |
| indexOf  LastIndexOf | 等效于List.indexOf( )和List.LastIndexOf( ) |  |
| newArrayList  newCopyOnWriteArrayList  newLinkedList | 简单工厂模式创建ArrayList/CopyOnWriteArrayList/LinkedList |  |
| partition | 列表分割.每个分割后的子列表元素个数不大于入参size值 |  |
| reverse | List反转 |  |
| transform | 列表元素转换 |  |

#### Sets

|  |  |  |
| --- | --- | --- |
| 方法 | 说明 | 备注 |
| cartesianProduct | 返回笛卡尔积组合List | 不常用 |
| difference  symmetricDifference  intersection  union | 取差集(集合A存在且不在集合B存在)/取差集(集合A存在且不在集合B存在或集合B存在且不在集合A存在)/交集/并集 | 注意这些方法和Jakarta Commons中Collections的union,intersection,subtract方法返回List不同, 这里返回的是Set. |
| filter | Set集合元素过滤 |  |
| newConcurrentHashSet  newHashSet  newLinkedHashSet  newTreeSet | 简单工厂模式创建ConcurrentHashSet/HashSet/LinkedHashSet/TreeSet |  |

#### Maps

|  |  |  |
| --- | --- | --- |
| 方法 | 说明 | 备注 |
| asMap | 将Set集合通过Function转换为Map |  |
| transformEntries | 转换Map的Entry对象为一个新的Map value对象 | 请参见:  MapsTest.transformEntriesTest( ) |
| transformValues | 转换Map的Value对象为一个新的Map value对象 | 请参见:  MapsTest.transformValuesTest( ) |
| safeContainsKey  saveGet  saveRemove | 安全的判断元素是否在Map中/获取Map元素/删除Map元素 | 不会抛出异常 |
| difference | 取两个Map的差集 |  |
| filterEntries  filterKeys  filterValues | 评估Map的Entry对象进行过滤  评估Map的key进行过滤  评估Map的value进行过滤 |  |
| newHashMap  newLinkedHashMap  newTreeMap | 简单工厂模式创建HashMap/LinkedHashMap/TreeMap |  |
| uniqueIndex | 利用Function生成Map中的key |  |

#### MultiMap

MultiMap类似于Jakarta Commons中的MultiValueMap, 运行一个key有多个value. Guava中还提供了一些MultiMap的实现类. 它们是:

|  |  |  |
| --- | --- | --- |
| 方法 | 说明 | 备注 |
| ListMultiMap | Map中允许存在重复的key-value pair | 这是个接口 |
| ArrayListMultiMap | 由ArrayList维护Map中的key |  |
| SetMultiMap | Map中不允许存在重复的key-value pair | 这是个接口 |
| FilteredMultiMap | MultiMap元素添加修改操作前须进行评估 | 这是个接口 |

#### Table

Table是Guava给集合包新增的一种类型.Table有行(row), 列(columne)和单元格(value)属性. Table的主要方法有:

|  |  |  |
| --- | --- | --- |
| 方法 | 说明 | 备注 |
| clear | 清除Table数据 |  |
| column  columnKeySet  columnMap | 返回某一列值/获取column集合/返回以col作为key, row和cell作为value的Map |  |
| contains  containsColumn  containsRow  containsValue | 判断某个单元格/列/行/Map是否在table中 |  |
| isEmpty | 判断table是否为空 |  |
| put  putAll | 数据/table数据添加 |  |
| row  rowKeySet  rowMap | 返回某一行值/获取row集合/返回以row作为key,col和cell作为value的Map |  |
| size | 返回单元格数 |  |
| values | 返回所有单元格值的集合 |  |

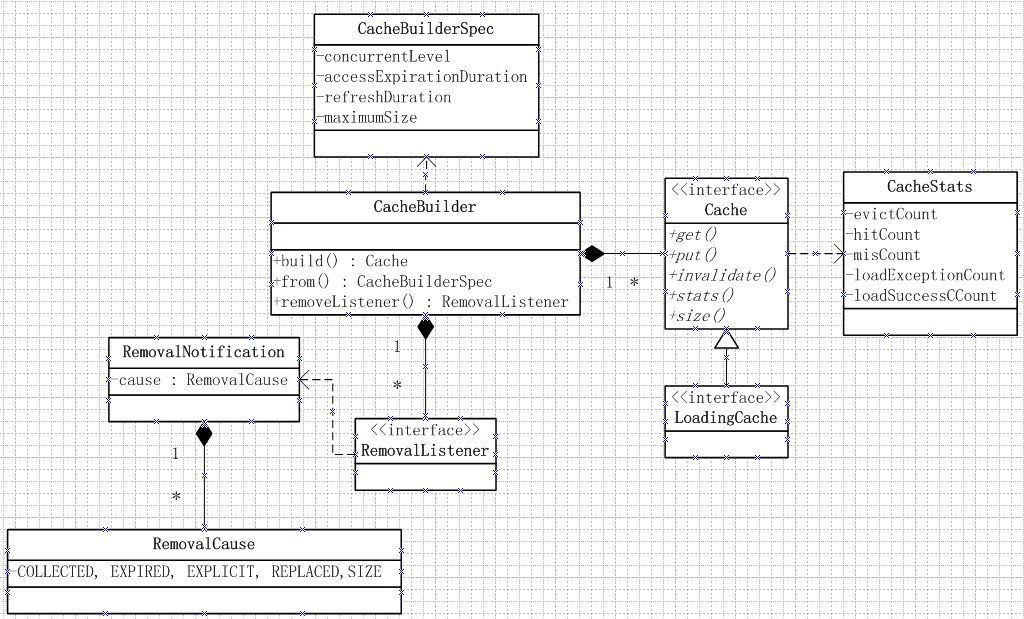
### 缓存包(cache)

关于缓存这个古老的话题, 我们可以有多种实现方法. 既可以通过JDK中的Map简单实现, 也可以使用诸如EhCache,OSCache等进程内缓存, 或者JbossCache, Memcached,Redis等分布式缓存.

Guava Cache是一种进程内缓存. 它具有如下的特点:

1. 小巧方便, 轻量级
2. 采用Fluent Style编程方式, 代码看起来十分优雅
3. 支持缓存过期, 缓存参数配置, 缓存删除事件触发, 缓存统计信息采集等功能.

Guava Cache主要类及其之间的关系如下:



说明如下:

#### Cache

Cache接口定义了缓存操作的主要方法.

|  |  |  |
| --- | --- | --- |
| 方法 | 说明 | 备注 |
| asMap | 以Map形式返回缓存 |  |
| cleanUp | 缓存清除 |  |
| get  getAllPresent  getIfPresent | 缓存获取 | get的获取策略为:缓存中没有,则创建并放入缓存中,然后返回; 缓存中有,从缓存返回  getAllPresent,getIfPresent只从缓存中获取 |
| invalidate  invalidateAll | 缓存失效 |  |
| put  putAll | 缓存添加/修改 |  |
| size | 缓存大小 |  |
| stats | 获取缓存统计信息 |  |

请参见:

LoadingCacheTest

#### CacheBuilder

CacheBuilder是缓存对象创建的工厂类

|  |  |  |
| --- | --- | --- |
| 方法 | 说明 | 备注 |
| from | 从字符串或CacheBuilderSpec对象中获取缓存参数配置 |  |
| build | 创建缓存对象 |  |
| concurrencyLevel | 并发线程数设置 |  |
| expireAfterAccess  expireAfterWrite | 缓存访问/写入后多长时间失效 |  |
| maxiumSize | 缓存最多容纳个数 |  |
| removalListener | 添加RemovalListener监听器 |  |
| softValues  weakValues | 使缓存对象变为SoftReference或WeakReference对象 |  |

#### CacheBuilderSpec

CacheBuilderSpec是个配置类, 用于定义缓存对象参数定义.

|  |  |  |
| --- | --- | --- |
| 属性或方法 | 说明 | 备注 |
| accessExpirationDuration  accessExpirationTimeUnit | 访问过期时长/单位 |  |
| refreshDuration  refreshTimeUnit | 刷新过期时长/单位 |  |
| writeDuration  writeTimeUnit | 写入过期时长/单位 |  |
| initialCapacity | 初始容量 |  |
| maxiumSize | 最大容量 |  |
| concurrencyLevel | 并发线程数 |  |
| keyStrength  valueStrength | 键/值强度 | 指的是对象引用是强引用(Strong), 软引用(Soft)还是虚引用(Weak) |
| parse( ) | 解析字符串生成CacheBuilderSpec对象 |  |

请参见:

CacheBuilderSpecTest

#### CacheStats

CacheStats对象用于统计缓存使用情况, 需要注意的是:不建议在生产环境开启此对象统计, 毕竟对性能有影响. 其主要属性有:

|  |  |  |
| --- | --- | --- |
| 属性 | 说明 | 备注 |
| hitCount | 从缓存中命中次数 |  |
| missCount | 未从缓存中命中而是直接生成缓存对象次数 |  |
| evictionCount | 对象缓存失效次数 | 不包括手动invalidate( )方法调用 |
| hitRate | 命中率 | hitCount/requestCount |
| loadExceptionCount | 缓存返回抛出异常次数 |  |
| loadSuccessCount | 缓存返回成功数 |  |
| loadExceptionRate | 加载异常率 | loadExceptionCount/(loadSuccessCount+loadExceptionCount) |

请参见:

CacheStatsTest

#### RemovalCause

RemovalCause对象反映缓存被删除的原因. 可能有以下四种情况:

* COLLECTED : 缓存被垃圾回收掉了. 这是由于可能使用了SoftReference或WeakReference对象
* EXPIRED : 缓存对象过期被释放了
* EXPLICIT : 缓存对象被明确删除了, 比如调用了invalidate或remove方法
* REPLACED : 缓存对象被替换了. 比如调用replace, refresh等方法
* SIZE : 缓存对象由于大小限制被释放了.

请参见:

RemovalListenerTest

### 并发包(concurrent)

自从JDK1.5引入并发编程包(java.util.concurrent)后, Java对于多线程的处理得到了极大的拓展. Guava在JDK1.5的基础上, 又给并发编程加入了一些新的特性. 他们是:

#### Monitor

Monitor和JDK中的ReentrantLock很像, 提供了一种锁的机制, 并且这种锁是可重用的.典型的Monitor用法如下:

if (monitor.enterIf(guardCondition)) {

try {

doWork();

} finally {

monitor.leave();

}

}

//或

monitor.enterWhen(guardCondition);

try {

doWork();

} finally {

monitor.leave()

}

Monitor主要方法有:

|  |  |  |
| --- | --- | --- |
| 方法 | 说明 | 备注 |
| enter | 锁进入 | 有两个重构方法. 要不无限等待直到获取到Monitor对象, 要不等到指定的timeout时间 |
| enterIf  tryEnterIf | 条件锁不等待进入 | 有两个重构方法. 要不无限等待直到获取到Monitor对象, 要不等到指定的timeout时间  获取Monitor对象后,不会再等待Guard对象的条件(Condition)是否满足, 但会返回一个boolean值表示Guard对象的条件(Condition)是否满足 |
| enterWhen | 条件锁等待进入 | 等待直到获取Monitor对象,并且获取Monitor对象后,等待直到Guard对象的条件(Condition)满足为止 |

请参见:

MonitorExampleTest

#### ListenableFuture, FutureCallback, FutureFallback

ListenableFuture是对类Future的扩展, 允许可以添加一个回调函数(FutureCallback或FutureFallback). 这样可以避免Future的get( ) 方法调用阻塞. 其典型用法为:

ListenableFuture future = executorService.submit(new Task());

FutureCallback futureCallback=new FutureCallbackImpl();

Futures.addCallback(future,futureCallback);

FutureFallback futureFallback=new FutureFallbackImp();

Futures.withFallback(future, futureFallback);

FutureCallback接口有两个接口方法, 它们是:

//成功从Future获取结果后调用

void onSuccess(@Nullable V result);

//未能从Future获取结果或取消时调用

void onFailure(@Nullable V result);

FutureFallback不同于FutureCallback, 它给Future获取结果失败的默认处理方式提供了可能.

ListenableFuture<V> create(Throwable t) throws Exception;

请参见:

ListenableFuturesWithCallbacksTest

#### Futures

Futures是Future的工具类, 其主要的方法有:

|  |  |  |
| --- | --- | --- |
| 方法 | 说明 | 备注 |
| addCallback | 给ListenableFuture添加callback |  |
| allAsList | 将Future集合转换为CombinedFuture |  |
| get | 相当于Future.get方法 |  |
| immediateCancelledFuture  immediateCheckedFuture  immediateFailedCheckedFuture  immediateFailedFuture  immediateFuture | 返回ImmediateCancelledFuture / ImmediateSuccessfulCheckedFuture / ImmediateFailedCheckedFuture / ImmediateFailedFuture / ImmediateSuccessfulFuture |  |
| transform  lazyTransform | 返回一个新的ListenableFuture对象,其结果是Function函数子转换后的结果  返回一个新的Future对象,其结果是Function函数子转换后的结果, 且转换过程直到Future.get方法调用才进行 |  |
| witchFallback | 给ListenableFuture对象添加默认失败或异常处理的回调函数 |  |

请参见:

FuturesTest

#### RateLimiter

RateLimiter可以看做线程并发数量控制器, 它对单位时间内并发线程的数量起到限制作用.典型的用法为:

//每秒最多只能两个线程并发

private RateLimiter rateLimiter = RateLimiter.create(2.0);

//in Thread

rateLimiter.acquire( );

doThreadThing( );

请参见:

RateLimiterTest

### 事件总线包(eventbus)

当我们要实现事件触发机制时, 一般我们会用观察者模式. Guva中提供了更为强大的事件触发机制: 事件总线.它具有如下特点:

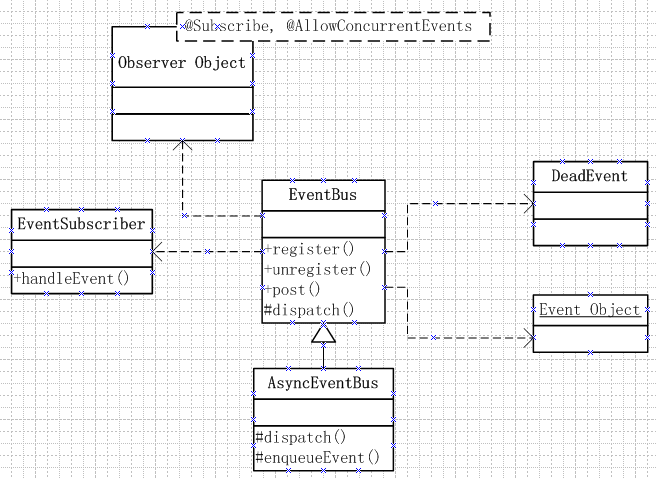
* 支持串行或并行事件触发
* 非常方便的支持事件注册, 注销
* 支持孤儿或死亡事件(DeadEvent)的处理

当我们使用Guava的事件总线包时, 一般的步骤为:

* 获取EventBus对象引用, 并进行注册
* 对事件监听方法添加@Subscribe注解
* 调用EventBus对象的post方法, 进行事件发布

Guava中除了有同步事件总线外, 还有异步事件总线, 异步事件总线允许多个事件触发并行执行. 使用异步事件总线需将上述的EventBus引用替换为AsyncEventBus对象引用, 并且在监听方法上还要加上@AllowConcurrentEvents注解.

Guava事件总线包主要类及其关系见下:



请参见:

SimpleTradeAuditorTest

UnregisteredTradesAuditorTest

DeadEventSubscriberTest

AsyncEventBusTest

### Io包(io)

Guava中也提供了对文件及输入输出进行处理的工具类, 它们是:

#### Files

|  |  |  |
| --- | --- | --- |
| 方法 | 说明 | 备注 |
| append | 文件内容添加 |  |
| asByteSink  asByteSource  asCharSink  asCharSource | 将文件转换为byteSink(字节输出)/byteSource(字节输入)/charSink(字符输入)/charSource(字符输入) | Source可看做是InputStream的工厂类, Sink可看做是OutputStream的工厂类 |
| copy | 文件或流复制 |  |
| createParentDirs | 递归目录生成 |  |
| createTempDir | 在系统临时目录下生成临时目录 |  |
| getFileExtension | 获取文件扩展名 |  |
| getNameWithoutExtension | 获取文件名(不包括扩展名) |  |
| newInputStreamSupplier  newOutputStreamSupplier  newReader  newReaderSupplier  newWriter  newWriterSupplier | 简单工厂模式创建InputStreamSupplier/OutputstreamSupplier/Reader/Writer/ReaderSupplier/WriterSupplier |  |
| readBytes | 将文件内容读入ByteProcessor |  |
| readFirstLine  readLines | 读取第一行内容/读取内容到字符串集合 |  |
| write | 文件输出 |  |

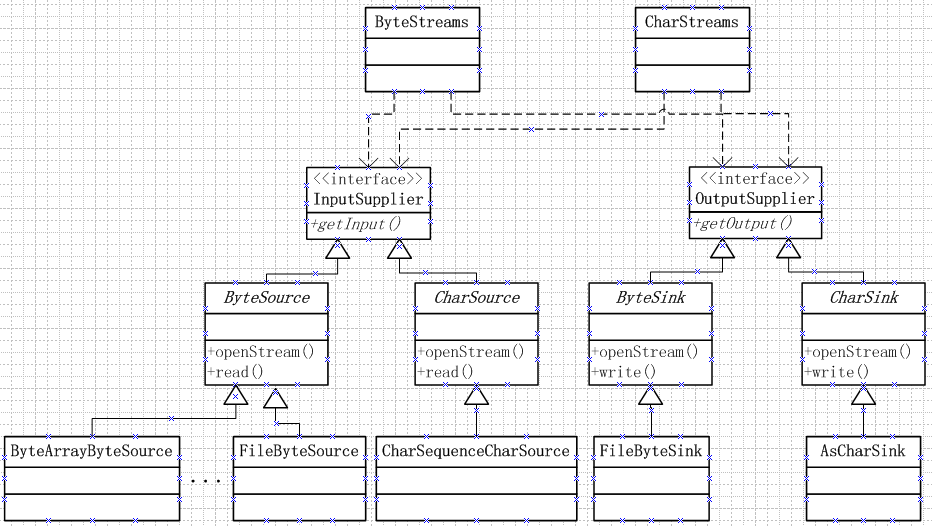
请参见:

FilesTest

#### Source, Sink

我们在使用JDK的输入输出流时, 经常会遇到一个问题, 我们需要手动的打开流, 进行输入输出操作后,再关闭流. 如果我们忘记了流的释放, 轻微的后果是对象不释放, 不会被gc, 严重的后果可能造成安全隐患,或者在多线程并发的情况下造成输入输出与预想的不一致. 针对这个问题, Guava中引入了Source和Sink的概念: Source可以看做输入流的工厂类, 对Source的每个输入操作方法(read调用等)都会创建一个新的InputStream; Sink可以看做输出流的工厂类, 对Sink的每个输入操作方法(write调用等)都会创建一个新的OutputStream.

Guava中关于Source和Sink的相关类及其之间的关系为:



请参见:

ByteSourceTest

ByteSinkTest

CharSourceTest

CharSinkTest

ByteStreamsTest

CharStreamsTest