**Closure<T>接口及其实现子类和ClosureUtils工具类**

1. **Closure**闭包封装特定的业务功能。
2. **closure** n. 关闭；终止，结束；**关闭，闭锁，封锁** vt. 使终止
3. Closure<T>是一个接口，存在于包org.apache.commons.collections4中，同Predicate、Transformer一样，都是函数式编程。Closure<T>接口有很多实现子类，如ForClosure、SwitchClosure等，

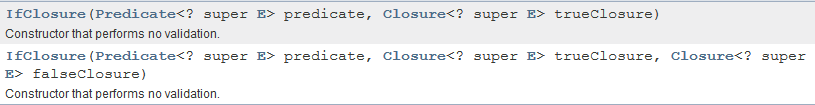
这些实现子类都存在于org.apache.commons.collections4.functors子包中。Closure<T>也具有一个工具类ClosureUtils。

1. ClosureUtils工具类：主要是用于获取Closure的实现类对象和一些循环。
2. Closure<T>接口只定义了一种方法：**execute（T input）方法：**

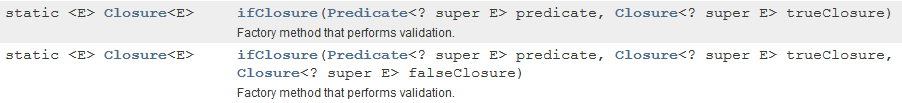


1. IfClosure<T>类：
2. 如何创建IfClosure<T>对象：

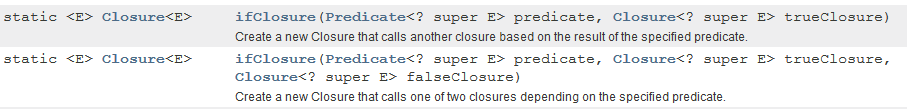
构造方法可以：



静态方法：



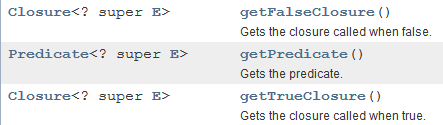
也可以通过ClosureUtils工具类的ifClosure方法：



1. IfClosure可以只有一个true对应的Closure，也可以有两个Closure对象，一个对true一个对false。
2. 成员方法：
   1. execute方法：对Closure接口的实现。



* 1. 对参数的获取：一个Predicate，两个Closure。



1. IfClosure的示例：

**需要一个Predicate，两个Closure。**

Predicate<Goods> predicate = **new** Predicate<Goods>() {

@Override

**public** **boolean** evaluate(Goods object) {

**return** object.getDiscount();

}

};

Closure<Goods> discountClosure = **new** Closure<Goods>() {

@Override

**public** **void** execute(Goods input) {

input.setPrice((**float**) (input.getPrice() \* 0.9));

}

};

Closure<Goods> notDiscountClosure = **new** Closure<Goods>() {

@Override

**public** **void** execute(Goods input) {

**if** (input.getPrice() >= 100) {

input.setPrice(input.getPrice() - 20);

}

}

};

Closure<Goods> closure = ClosureUtils.*ifClosure*(predicate,

discountClosure, notDiscountClosure);

测试：

Goods good1 = **new** Goods("brushesesea",120.0f,**true**);

Goods good2 = **new** Goods("football",100.0f,**false**);

Goods good3 = **new** Goods("basketball",80.0f,**false**);

List<Goods> list = **new** ArrayList<Goods>();

list.add(good1);

list.add(good2);

list.add(good3);

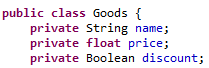
IterableUtils.*forEach*(list, closure);

Iterator<Goods> iterator = list.iterator();

**while**(iterator .hasNext()) {

System.***out***.println(iterator .next());

}

Goods类：

结果：

1. WhileClosure实现类：
2. 特殊功能：**对任何一个对象进行循环多次判断Predicate，直到Predicate返回false。**如工资增长到10000以上，无论多低，总会增到10000以上。
3. 获取WhileClosure对象。
   1. 构造方法：**doLoop为false，为while…do循环，为true时，为do…while循环。**



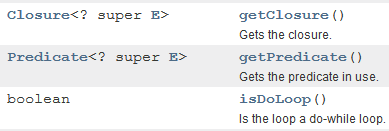
* 1. 静态方法：



* 1. ClosureUtils中的whileClosure方法：**（默认为while—do循环）**



1. 获取参数：



1. 实现Closure接口中的execute方法：



1. 示例：

**public** **static** **void** main(String[] args) {

Predicate<Employee> predicate = **new** Predicate<Employee>() {

@Override

**public** **boolean** evaluate(Employee object) {

**return** object.getWage()<10000;

}

};

Closure<Employee> addWage = **new** Closure<Employee>() {

@Override

**public** **void** execute(Employee input) {

input.setWage((**int**)(input.getWage()\*1.2));

}};

// Closure<Employee> closure = ClosureUtils.whileClosure(predicate, addWage );

Closure<Employee> closure = WhileClosure.*whileClosure*(predicate, addWage, **true**);

Employee employee1 = **new** Employee("马大帅",30000);

Employee employee2 = **new** Employee("马大帅2",3000);

Employee employee3 = **new** Employee("马大帅3",2000);

List<Employee> list = **new** ArrayList<Employee>();

list.add(employee1);

list.add(employee2);

list.add(employee3);

IterableUtils.*forEach*(list, closure);

Iterator<Employee> iterator = list.iterator();

**while**(iterator.hasNext()){

System.***out***.println(iterator .next());

}

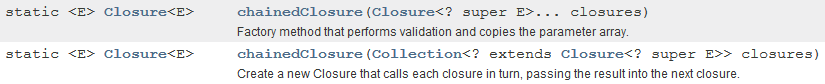
结果：false true



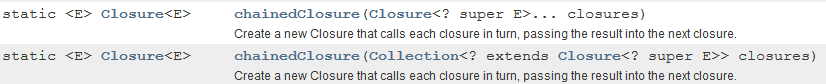
1. ChainClosure<E>实现类：
2. 创建新的对象：
   1. **构造方法**：



* 1. **静态方法**：引入了JDK5的新特性，参数任意多个。



* 1. ClosureUtils的**chainClosure方法**：



1. 实现接口Closure中的execute方法



1. 获取连接的Closure对象，返回一个Closure数组。



1. 示例：

Closure<Goods> discountClosure = **new** Closure<Goods>() {

@Override

**public** **void** execute(Goods input) {

**if** (input.getPrice() > 100) {

input.setPrice((**float**) (input.getPrice() \* 0.9));

}

}

};

Closure<Goods> notDiscountClosure = **new** Closure<Goods>() {

@Override

**public** **void** execute(Goods input) {

**if** (input.getPrice() >= 100) {

input.setPrice(input.getPrice() - 20);

}

}

};

**Closure<Goods> closure = ClosureUtils.*chainedClosure*(discountClosure,**

**notDiscountClosure);**

Goods good1 = **new** Goods("brushesesea", 120.0f, **true**);

Goods good2 = **new** Goods("football", 101.0f, **true**);

Goods good3 = **new** Goods("basketball", 80.0f, **true**);

List<Goods> list = **new** ArrayList<Goods>();

list.add(good1);

list.add(good2);

list.add(good3);

IterableUtils.*forEach*(list, closure);

Iterator<Goods> iterator = list.iterator();

**while** (iterator.hasNext()) {

System.***out***.println(iterator.next());

}}

结果：，第一个先打折，然后减20.

1. 自定义Closue<T>实现类：只需要实现execute方法即可。

如：Closure<Employee> closure = **new** Closure<Employee>() {

@Override

**public** **void** execute(Employee input) {

input.setSalary(input.getSalary()\*2);

} };

List<Employee> list = **new** ArrayList<Employee>();

list.add(**new** Employee("马大帅",2000));

list.add(**new** Employee("刘老根",10000));

list.add(**new** Employee("猪无能",3000));

// CollectionUtils.forAllDo(list, closure);

IterableUtils.*forEach*(list, closure);

Iterator<Employee> iterator = list.iterator();

**while**(iterator.hasNext()) {

System.***out***.println(iterator .next());

}

结果： 

用于操作**集合**时：

4.0版本CollectionUtils中有方法forAllDo（Collection,Closure）,但是从4.1开始，此方法过时了，改用IterableUtils的forEach方法。

