# 1.函数式接口概念

JDK1.8

方法引用:就是为一个方法设置别名

**注意:**

1. 函数的引用必须定义一个接口,这个接口只能有一个方法当然这个接口可以还有static方法和default方法，为了说明只能有一个接口,对这个类使用@FunctionalInterface(函数式接口)

2）函数式接口可以还可以重写继承于Object的方法,如

@FunctionInterface

Public interface Comparator<T>{

Public int compare(T o1,T o2);

Public boolean equals(Object obj);//继承于Object

}

**方法引用java8一共定义了4种形式:**

1. **引用静态方法**: 类名称::static 方法名称;
2. **引用某个对象的普通方法:** 实例化对象 :: 普通方法;
3. **引用特定类型的方法:** 特定类 :: 普通方法;
4. **引用构造方法:** 类名称 :: new;

范例:1)引用静态方法

在String 类里有一个valueOf()方法:public static String valueOf(数据类型);

|  |
| --- |
| ublic class test{  public static void main(String args[]){  //将Stirng.valueOf()方法变为了IMessage接口里的zhuanhuan方法  IMessage<Integer,String> msg = String::valueOf;  String str = msg.zhuanhuan(1000);  System.out.print(str.replaceAll("0","9"));  }  }  @FunctionalInterface  interface IMessage<P,R>{  /\*\*  \*  \* @param <P>引用方法的参数类型  \* @return <R>引用方法的返回值类型  \*/  public R zhuanhuan(P p);  } |

**范例:2) 引用某个对象的普通方法**

|  |
| --- |
| public class test{  public static void main(String args[]){  IMessage<String> msg = "hello"::toUpperCase;  System.out.print(msg.upper());  }  }  @FunctionalInterface  interface IMessage<R>{  public R upper();  } |

**范例:3)引用特定类型的方法**

如:String类: public int compareTo(Stirng anotherString)

|  |
| --- |
| public class test{  public static void main(String args[]){  IMessage<String> msg = String::compareTo;  System.out.print(msg.compare("A","B"));  }  }  interface IMessage<P>{  public int compare(P p1,P p2);  } |

**范例:4)引用构造方法**

|  |
| --- |
| public class test{  public static void main(String args[]){  IMessage<Book> msg = Book::new;  Book b = msg.create("java开发",34.6);  System.out.print(b);  }  }  interface IMessage<C>{  public C create(String t,double p);  }  class Book{  private String title;  private double price;  public Book(String title,double price){  this.price = price;  this.title = title;  }  @Override  public String toString() {  return "书名:" + this.title + ",价格:" + this.price;  }  } |

# 2.内建的函数式接口java.util.function

1)功能型接口(Funciton): public interface Function<T,R> {public R apply(T t);}

此接口需要接受一个参数T,返回一个处理结果R

2)消费型接口(Consumer): public interface Consumer<T>{public void accept(T t);}

接受数据,不返回处理结果

3)供给型接口(Supplier): public interface Supplier<T>{public T get();}

不接受参数,但是能返回处理结果

4) 断言型接口(Predicate): public interface Predicate<T>{public boolean test(T t);}

进行判断使用

由于JDK1.8有四个功能型接口,所以一般很少用户去自定义新的函数式接口

范例: 1)功能型接口

Strng类的一个方法:public boolean startsWith(String str)

|  |
| --- |
| public class test{  public static void main(String args[]){  Function<String,Boolean> f = "##hello"::startsWith;  System.out.print(f.apply("##"));  }  } |

范例:2) 消费型接口

|  |
| --- |
| public class test{  public static void main(String args[]){  Consumer<String> c = new MyDemo()::print;  c.accept("hello,world");  }  }  class MyDemo{  public void print(String str){ //此方法没有返回值有参数  System.out.print(str);  }  } |

范例:3) 供给型接口

String里的toUpperCaes()方法: public String toUpperCase();

|  |
| --- |
| public class test{  public static void main(String args[]){  Supplier<String> s = "hello"::toUpperCase;  System.out.print(s.get());  }  } |

范例:4) 断言型接口

String里有一个equalsIgnoreCase():public boolean equalsIgnoreCase(String anotherString)

|  |
| --- |
| public class test{  public static void main(String args[]){  Predicate<String> p = "hello"::equalsIgnoreCase;  System.out.print(p.test("HELLO"));  }  } |

# 3.lambda和函数式接口

|  |
| --- |
| //p为断言型接口 boolean test(T t)  Predicate<String> p = (x)->x.contains("a");  System.out.println(p.test("hello")); //fale  System.out.println(p.test("abc")); //true |