

**课 程 实 验 报 告**

**课程名称：Java语言程序设计**

**实验名称：泛型栈模拟泛型队列**

**院 系 ： 计算机科学与技术**

**专业班级 ： 物联网1601班**

**学 号 ： U201614907**

**姓 名 ： 杨钧淮**

**指导教师 ： 马光志**

**2019 年 05 月 03 日**

1. 需求分析
2. 题目要求

参见https://docs.oracle.com/javase/7/docs/api/java/util/Queue.html，Java提供的java.util.Queue是一个接口没有构造函数。试用java.util.Stack<E>泛型栈作为父类，用另一个泛型栈对象作为成员变量，模拟实现一个泛型子类Queue<E>，当存储元素的第1个栈的元素超过dump时，再有元素入队列就倒入第2栈。除提供无参构造函数Queue( )外，其它所有队列函数严格参照java.util.Queue的接口定义实现。

import java.util.Stack;

import java.util.NoSuchElementException;

public class Queue<E> extends Stack<E>{

private Stack<E> stk;

public Queue( ){ /\* 在此插入代码\*/ }

public boolean add(E e) throws IllegalStateException, ClassCastException,

NullPointerException, IllegalArgumentException{ /\* 在此插入代码\*/ }

public boolean offer(E e) throws ClassCastException, NullPointerException,

IllegalArgumentException{ /\* 在此插入代码\*/ }

public E remove( ) throws NoSuchElementException { /\* 在此插入代码\*/ }

public E poll( ) { /\* 在此插入代码\*/ }

public E peek ( ) { /\* 在此插入代码\*/ }

public E element( ) throws NoSuchElementException { /\* 在此插入代码\*/ }

}

**考虑到Stack<E>只能存储类型为E的元素，以及Stack是一个存储能力(capacity, 参见有关说明)理论上无限的类型，这可能会影响到相关方法的异常处理，请适当处理上述异常（也许某些异常从来都不会发生）。**

**思考：Queue<E>是否应该提供clone和equals函数，以及其它一些函数如addAll等？**

1. 需求分析

本次实验需要使用栈来模拟队列，与直接实现的队列并不相同。用栈来模拟队列，需要考虑使用栈实现队列时的影响、效率等因素，还需要考虑程序健壮性在内的多种因素。通过分析，用栈来实现队列需要使用两个栈，但是两个栈之间的操作其实给队列的效率有了一定的下降。

1. 系统设计
2. 概要设计

在队列的接口中，主要有6个方法，如下图所示。

图 1 Queue类要实现的方法

所以我们除了要实现6种方法外，还需要实现构造函数。由于我们用栈模拟队列需要用到两个栈来实现，所以我们继承了父类的持有栈之外，还需要一个成员栈。将成员栈作为入队栈，然后将持有栈作为出队栈。当有元素要进入队列时，先进入入队栈，有元素要出队列时，则从出队栈出队。具体操作如下所示

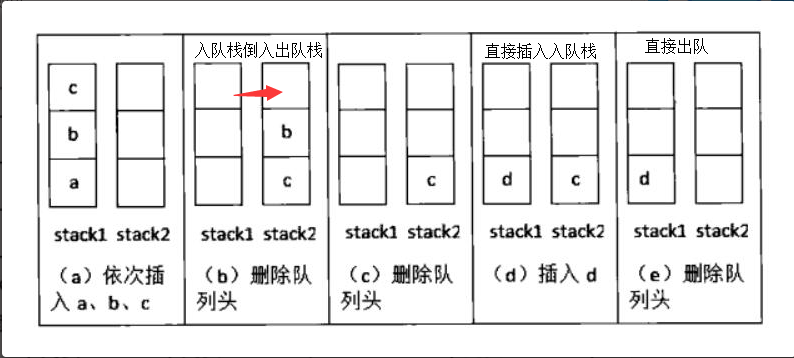


图 2 实现方法

1. 详细设计
2. add方法

add方法把元素加入到队列中，在栈的角度上来说，入队即将元素加入到入队栈，如果入队栈已满，则将入队栈倾倒到出队栈，如果出队栈有元素，则抛出IllegalStateException异常，表示队列已满，不能再插入元素。而NullPointerException在Stack中就可以抛出，所以不需要单独抛出该异常。具体如下图：

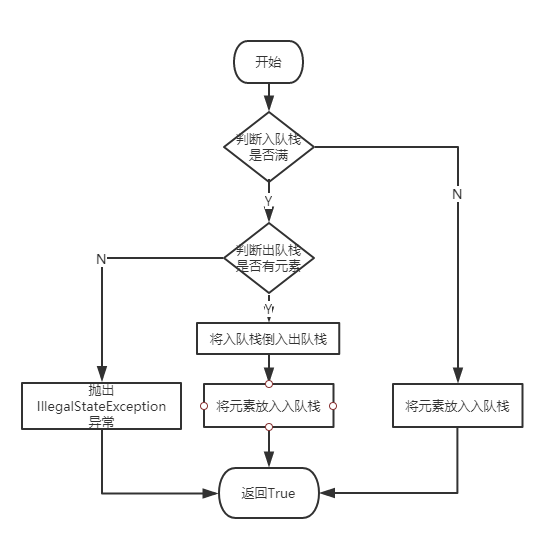


图 3 add方法

1. offer方法

offer方法与add方法类似，只不过其不会抛出IllegalStateException，相反，它在队列已满时返回布尔类型false。其流程图如图 3所示。

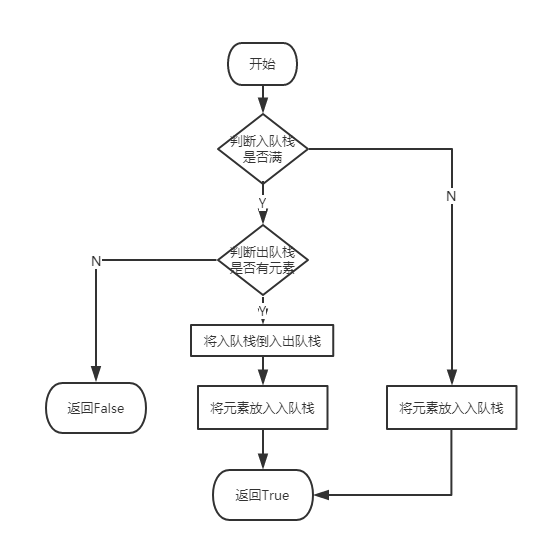


图 4 offer方法

1. remove方法

remove方法用于尝试从队尾移除一个元素并返回这个元素的值。出队即将出队栈的元素出栈即可。若出队栈中没有元素，则将入队栈元素倒入出队栈出队，若入队栈没有元素，则抛出 NoSuchElementException()异常。

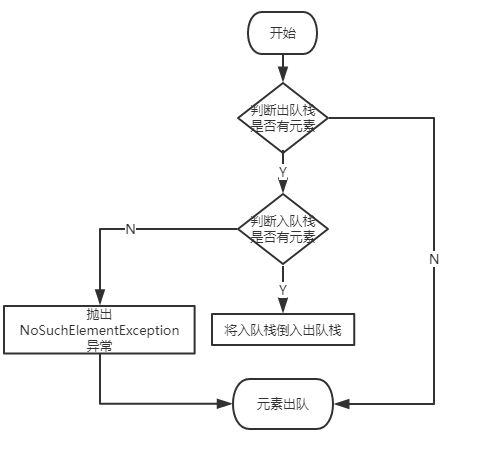


图 5 remove方法

1. poll方法

poll方法与remove方法类似，只不过其不会抛出NoSuchElementException异常，它在队列已满时返回布尔类型false。其流程图如下所示。

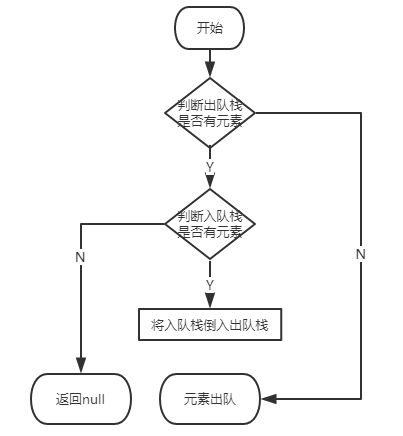


图 6 poll方法

1. element方法

element方法只读取队尾元素的值，而不进行出队操作，所以判断出队栈是否有元素的操作和出队相似。当队列为空时，抛出 NoSuchElementException异常。

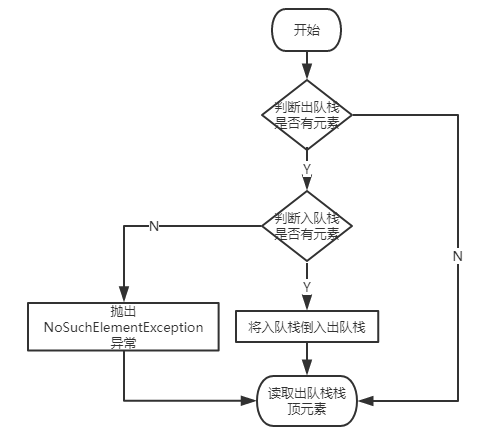


图 7 element方法

1. peek方法

peek方法同element方法，只读取队尾元素的值，而不进行出队操作。不同点在于peek方法不抛出NoSuchElementException异常，而是返回null空指针。

1. 软件开发

开发环境：

Windows 10 家庭版 1803

java version "1.8.0\_201"

Java(TM) SE Runtime Environment (build 1.8.0\_201-b09)

Java HotSpot(TM) 64-Bit Server VM (build 25.201-b09, mixed mode)

使用IntelliJ IDEA 2019.1 x64开发。

1. 软件测试

由于题目没有给出测试样例和测试要求，所以我没有做一个交互式的测试界面，只使用程序从上到下的测试了该队列容器应有的功能和异常处理。

首先测试add方法。想队列中循环插入30个元素，如下图，

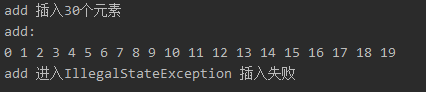


图 8 add测试

可见，在插入第21个元素时抛出异常。

测试remove方法，如下图：

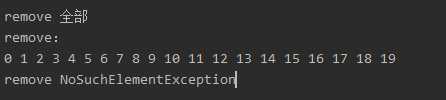


图 9 remove测试

可见，没有元素时抛出异常。

测试offer：

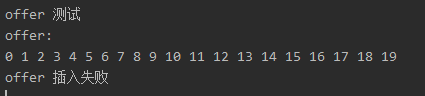


图 10 offer测试

当插入第21个元素时返回false。

测试poll：

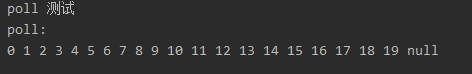


图 11 poll测试

可见队列空时返回null。

测试栈实现队列的正确性：

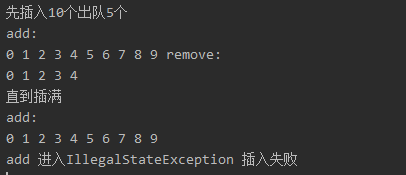


图 12 栈实现队列的正确性

在某些情况下，栈并不能全部装满所期望的元素，当出队栈有元素，入队栈又满了时，就会发生该种情况。可见功能正确。

下面测试peek和element的正确性。使用刚才插入的15个元素。

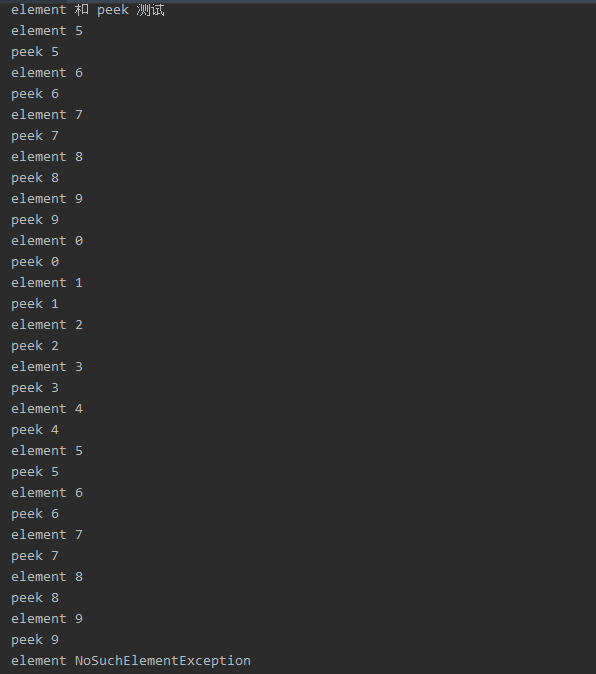


图 13 peek和element

1. 特点与不足
2. 技术特点

该Queue容器代码按照标准接口实现，代码简介易懂，且效率较高，能正确抛出异常，正确实现功能。

1. 不足和改进的建议

代码实现并不是很优雅，两个重复的方法中有很多重复的代码，可以有优化的空间。

1. 过程和体会
2. 遇到的主要问题和解决方法

在一开始的编码中，我将持有栈用作入队栈，将成员栈用作出队栈。这样导致了一个问题，在继承的父类Stack的实现中，pop方法首先调用了自己的peek方法。这样就导致了当入队栈出栈时，由于Java的动态绑特性，导致父类调用自己的peek方法时，调用回了Queue类的peek方法，而peek方法需要在出队栈有元素时才能出队，导致又要调用持有栈的出栈方法，这时又会调用到Queue的peek方法，形成无限循环，导致程序爆栈异常退出。

这个问题是由于Java的动态绑定的特性决定的，且继承的父类Stack的代码无法修改，导致出现了这个问题。后来发现，只需要将持有栈作为出队栈，成员栈作为入队栈就可以解决这个问题。这个时候当出队时，持有栈出栈，回调peek方法，peek方法中的super又调回父类的peek方法中，程序就能正确工作了。

1. 课程设计的体会

这次实验和C++的第四个实验有一些相似之处，但是又有一些不同之处。比如Java的方法都是C++中的虚函数，都是动态绑定的，无法改变。通过此次课程设计我对于Java的一些特性如继承机制、覆盖与重写机制有了更进一步的了解，让我更加熟悉了Java语言的一些特性。

1. 源码和说明
2. 文件清单及其功能说明

本实验由2个文件构成：Queue.java以及Main.java。

Queue.java为Queue类的实现，而Main.java则是对于Queue.java的测试程序。

1. 用户使用说明书

搭建好开发环境后，在cmd下使用如下命令编译和运行程序：

javac Queue.java

javac Main.java

java Main

也可以直接用IDEA的运行按钮直接运行。

1. 源代码

Queue.java

import java.util.Stack;

import java.util.NoSuchElementException;

public class Queue<E> extends Stack<E>{

private static final long serialVersionUID = 1L;

private Stack<E> stk; // skt只能作为入队栈 父类只能作为出队栈

public int f\_size = 10;

public Queue( )

{

stk = new Stack<E>();

}

public boolean add(E e)

throws IllegalStateException, ClassCastException,

NullPointerException, IllegalArgumentException

{

//判断入堆是否已满

if(stk.size() != f\_size )

{

//直接进入入队栈

stk.push(e);

}

else

{

// 判断出队栈是否有元素

if( this.isEmpty())

{

//没有则将入队栈中的所有元素出栈到出队栈

while (!stk.isEmpty()) {

this.push(stk.pop());

}

//进入入队栈

stk.push(e);

}

else {

throw new IllegalStateException();

}

}

return true;

}

public boolean offer(E e)

throws ClassCastException, NullPointerException, IllegalArgumentException

{

//判断入堆是否已满

if(stk.size() != f\_size )

{

//直接进入入队栈

stk.push(e);

}

else

{

// 判断出队栈是否有元素

if( this.isEmpty())

{

//没有则将入队栈中的所有元素出栈到出队栈

while (!stk.isEmpty()) {

this.push(stk.pop());

}

//进入入队栈

stk.push(e);

}

else {

return false;

}

}

return true;

}

public E remove( ) throws NoSuchElementException

{

//先看出队栈是否有元素

if( !this.isEmpty() )

{

// 直接出队

return this.pop();

}

else

{

// 如果入队栈不为0，则将入队栈出到出队栈

if( !stk.isEmpty() )

{

while (!stk.isEmpty()) {

this.push(stk.pop());

}

//再出栈

return this.pop();

}

else {

throw new NoSuchElementException();

}

}

}

public E poll()

{

//先看出队栈是否有元素

if( !this.isEmpty() )

{

// 直接出队

return this.pop();

}

else

{

// 如果入队栈不为0，则将入队栈出到出队栈

if( !stk.isEmpty() )

{

while (!stk.isEmpty()) {

this.push(stk.pop());

}

//再出栈

return this.pop();

}

else {

return null;

}

}

}

public E peek()

{

//先看出队栈是否有元素

if( !this.isEmpty() )

{

// 直接出队

return super.peek();

}

else

{

// 如果入队栈不为0，则将入队栈出到出队栈

if( !stk.isEmpty() )

{

//   System.out.println(stk.empty());

while (!stk.empty()) {

this.push(super.pop());

}

//再出栈

return this.peek();

}

else {

return null;

}

}

}

public E element( ) throws NoSuchElementException

{

//先看出队栈是否有元素

if( !this.isEmpty() )

{

// 直接出队

return super.peek();

}

else

{

// 如果入队栈不为0，则将入队栈出到出队栈

if( !stk.isEmpty() )

{

while (!stk.isEmpty()) {

this.push(stk.pop());

}

//再出栈

return this.peek();

}

else {

throw new NoSuchElementException();

}

}

}

}

Main.java

import java.io.IOException;

import java.util.NoSuchElementException;

public class Main {

public static void main(String[] args) throws IOException {

Queue<Integer> que = new Queue<>();

que.f\_size = 10; // 设置大小

int i;

// add test

System.out.println("add 插入30个元素");

try{

System.out.println("add: ");

for(i = 0; i < que.f\_size \* 3; ++i) {

que.add(i);

System.out.printf("%d ", i);

}

System.out.println();

} catch (IllegalStateException e) {

System.out.println("\nadd 进入IllegalStateException 插入失败");

}

// remove test

System.out.println();

System.out.println("remove 全部");

try {

System.out.println("remove： ");

for(i = 0; i < que.f\_size\*3; ++i)

System.out.printf("%d ", que.remove());

System.out.println();

} catch (NoSuchElementException e) {

System.out.println("\nremove NoSuchElementException");

}

// offer test

System.out.println();

System.out.println("offer 测试");

System.out.println("offer: ");

for(i = 0; i < que.f\_size \* 3; ++i) {

if (que.offer(i))

System.out.printf("%d ", i);

else {

System.out.println("\noffer 插入失败");

break;

}

}

// poll test

System.out.println();

System.out.println("poll 测试");

try {

System.out.println("poll: ");

for(i = 0; i < que.f\_size\*2 + 1; ++i)

System.out.printf("%d ", que.poll());

System.out.println();

} catch (NoSuchElementException e) {

System.out.println("\npoll NoSuchElementException");

}

System.out.println();

System.out.println("先插入10个出队5个");

try{

System.out.println("add: ");

for(i = 0; i < 10; ++i) {

que.add(i);

System.out.printf("%d ", i);

}

} catch (IllegalStateException e) {

System.out.println("\nadd 进入IllegalStateException 插入失败");

}

try {

System.out.println("remove: ");

for(i = 0; i < 5; ++i)

System.out.printf("%d ", que.remove());

} catch (NoSuchElementException e) {

System.out.println("\nremove NoSuchElementException");

}

System.out.println();

System.out.println("直到插满");

try{

System.out.println("add: ");

for(i = 0; i < 20; ++i) {

que.add(i);

System.out.printf("%d ", i);

}

} catch (IllegalStateException e) {

System.out.println("\nadd 进入IllegalStateException 插入失败");

}

System.out.println();

System.out.println("element 和 peek 测试");

try{

for(i = 0; i < que.f\_size \* 2 + 1; ++i) {

System.out.println("element " + que.element());

System.out.println("peek " + que.peek());

que.remove();

}

} catch (NoSuchElementException e) {

System.out.println("element NoSuchElementException");

}

}

}