# 作业六 栈

## 1、什么是栈？它有哪两种存储方式？栈的运算规则是什么？

栈（Stack）是一种特殊的线性表，其特点是只允许在表的一端进行插入和删除操作。这一端被称为栈顶，相对地，把另一端称为栈底。向一个栈插入新元素又称作进栈、入栈或压栈，它是把新元素放到栈顶元素的上面，使之成为新的栈顶元素。从一个栈删除元素又称作出栈或退栈，它是把栈顶元素删除掉，使其相邻的元素成为新的栈顶元素。

栈有两种基本的存储方式：

1. **顺序存储：**利用一段连续的内存空间进行存储。这种方式能快速随机访问存储的元素，通过下标 index 访问，支持随机访问，查询速度快，但存在元素在数组空间中大量移动的操作，增删效率低。
2. **链式存储：**利用非连续的内存空间存储，元素之间使用指针进行链接。这种方式只支持顺序访问，在某些遍历操作中查询速度慢，但增删元素快。

栈的运算规则主要包括：

* **进栈：**如果栈未满，则将元素加入使之成为新栈顶。
* **出栈：**如果栈非空，则弹出栈顶元素。
* **判断栈是否为空：**如果栈为空，则返回true，否则返回false。
* **获取栈顶元素：**如果栈非空，则返回栈顶元素。

这些操作都遵循“后进先出”（LIFO）的原则，即最后进入栈的元素最先被取出。这也是栈这种数据结构的核心特性。在实际应用中，栈的这种特性被广泛利用，例如在程序执行查看调用堆栈、加减运算、甚至在搜索算法中的深度优先搜索等。

## 2、栈有哪些基本操作？

栈的基本操作主要包括：

* **进栈**：如果栈未满，则将元素加入使之成为新栈顶。
* **出栈**：如果栈非空，则弹出栈顶元素。
* **判断栈是否为空**：如果栈为空，则返回true，否则返回false。
* **获取栈顶元素**：如果栈非空，则返回栈顶元素。

这些操作都遵循“后进先出”（LIFO）的原则，即最后进入栈的元素最先被取出。这也是栈这种数据结构的核心特性。在实际应用中，栈的这种特性被广泛利用，例如在程序执行查看调用堆栈、加减运算、甚至在搜索算法中的深度优先搜索等。

## 3、请简述括号匹配检验的算法思路。

括号匹配检验的算法思路通常使用\*\*栈\*\*数据结构来实现。以下是算法的基本步骤：

1. 创建一个空栈。

2. 从左到右遍历输入字符串的每个字符：

 - 如果当前字符是一个\*\*开括号\*\*（例如'('、'{'或'['），则将其\*\*压入栈\*\*中。

 - 如果当前字符是一个\*\*闭括号\*\*（例如')'、'}'或']'）：

 - 检查栈是否为空。如果栈为空，那么括号不匹配，因为没有相应的开括号与当前的闭括号匹配。

 - 如果栈不为空，则\*\*弹出栈顶\*\*的开括号，并检查它是否与当前的闭括号匹配。如果不匹配，那么括号不匹配。

3. 遍历完输入字符串后，检查栈是否为空。如果栈为空，那么括号匹配。如果栈不为空，那么括号不匹配，因为有一些开括号没有相应的闭括号。

这种算法可以有效地检查括号是否匹配，时间复杂度为O(n)，其中n为输入字符串的长度。这是因为每个字符只被访问一次，并进行O(1)的操作。空间复杂度也为O(n)，在最坏的情况下，即所有的括号都是开括号，此时所有的开括号都会被压入栈中。所以，这种算法在处理大规模数据时也具有很好的性能。

## 4、请解释以下代码的含义：

public interface Stack {

public int getSize();

 public boolean isEmpty();

 public void push(Object e);

 public Object pop() throws StackEmptyException;

public Object peek() throws StackEmptyException;

}

这段代码定义了一个名为 Stack 的接口。在计算机科学中，栈（Stack）是一种特殊的线性表，只能在表的一端（称为栈顶）进行插入或删除操作。这个接口定义了栈数据结构应有的基本操作：

* getSize()：返回栈的大小，即栈中元素的数量。
* isEmpty()：检查栈是否为空，如果为空返回 true，否则返回 false。
* push(Object e)：将元素 e 压入栈顶。
* pop()：从栈顶弹出一个元素并返回。如果栈为空，将抛出 StackEmptyException 异常。
* peek()：返回栈顶元素但不弹出。如果栈为空，将抛出 StackEmptyException 异常。

这个接口可以被任何类实现，以提供栈的具体实现，例如，可以使用数组或链表来实现这个接口。这是一种典型的数据结构设计，使得具体实现和抽象接口分离，提高了代码的可复用性和可维护性。

## 5、请解释以下代码的含义：

public class StackArray implements Stack {

 private final int LEN=8;

 private Object[] elements;

 private int top;

 public StackArray(){

 top=-1;

 elements=new Object[LEN];}

 public int getSize() {

 return top+l;}

 public boolean isEmpty() {

 return top<0;

 }

 public void push(Object e) {

 if (getSize()>=elements.length) expandSpace();

 elements[++top] = e;}

 private void expandSpace(){

 Object[] a = new Object[elements.length\*2];

 for (int i=0; i<elements.length; i++)

 a[i]=elements[i];

 elements=a;

}

public Object pop() throws StackEmptyException {

 if (getSize()<1)

 throw new StackEmptyException("错误，堆栈为空。");

 Object obj = elements[top];

 elements[top--] = null;

 return obj;

}

public Object peek() throws StackEmptyException {

 if (getSize()<1)

 throw new StackEmptyException("错误，堆栈为空。");

 return elements[top];

 }

}

这段代码定义了一个名为 StackArray 的类，它实现了 Stack 接口。这个类表示一个栈数据结构，栈是一种后进先出（LIFO）的数据结构。

以下是每个方法的详细解释：

* StackArray()：这是构造函数，用于初始化栈。它将 top 设置为 -1，表示栈为空，然后创建一个长度为 LEN 的 Object 数组 elements 作为栈的存储空间。
* getSize()：此方法返回栈的当前大小，即栈中元素的数量。
* isEmpty()：此方法检查栈是否为空。如果 top 小于 0，则栈为空。
* push(Object e)：此方法将一个对象 e 压入栈顶。如果栈的大小已经达到了 elements 的长度，那么就会调用 expandSpace() 方法来扩大 elements 的空间。然后，top 加一并将 e 放在 top 所指向的位置。
* expandSpace()：此方法用于扩大 elements 的空间。它创建一个新的 Object 数组 a，其长度是 elements 的两倍，然后将 elements 中的所有元素复制到 a 中，最后将 elements 指向 a。
* pop()：此方法用于弹出栈顶的元素。如果栈为空（即 getSize()<1），则抛出 StackEmptyException 异常。否则，取出栈顶元素，将栈顶位置设为 null，然后 top 减一。
* peek()：此方法用于查看栈顶的元素但不弹出。如果栈为空（即 getSize()<1），则抛出 StackEmptyException 异常。否则，返回栈顶元素。

这个类提供了一个基本的栈数据结构的实现，可以进行压栈、弹栈和查看栈顶元素等操作，并且可以动态地扩大栈的空间。当栈空间不足时，它会自动扩大两倍，从而避免了栈溢出的问题。这是一种常见的数据结构实现技巧。

## 6、请解释以下代码的含义：

public class StackSLinked implements Stack {

 private SLNode top;

 private int size;

 public StackSLinked() {

 top = null;

 size = 0;}

 public int getSize() {

 return size;

 }

public boolean isEmpty() {

 return size==0;

 }

 public void push(Object e) {

 SLNode q= new SLNode(e,top);

 top =q;

 size++;

 }

public Object pop() throws StackEmptyException {

 if(size<1)

 throw new StackEmptyException("错误，堆栈为空。");

 Object obj = top.getData();

 top=top.getNext();

 size--;

 return obj;

 }

public Object peek() throws StackEmptyException {

 if(size<1)

 throw new StackEmptyException("错误，堆栈为空。");

 return top.getData();

 }

}

这段代码是一个使用单链表实现的栈（Stack）的Java类。下面是每个部分的详细解释：

* private SLNode top; 和 private int size;：这两行定义了栈的顶部节点和栈的大小。
* public StackSLinked() {...}：这是构造函数，用于创建一个新的空栈。
* public int getSize() {...}：这个方法返回栈的大小。
* public boolean isEmpty() {...}：这个方法检查栈是否为空。
* public void push(Object e) {...}：这个方法将一个元素压入栈顶。
* public Object pop() throws StackEmptyException {...}：这个方法删除并返回栈顶的元素。如果栈为空，就会抛出一个异常。
* public Object peek() throws StackEmptyException {...}：这个方法返回栈顶的元素，但不删除它。如果栈为空，就会抛出一个异常。

这个类实现了栈的基本操作，包括push（压入）、pop（弹出）和peek（查看栈顶元素）。这些操作都遵循栈的后进先出（LIFO）原则。在这个实现中，栈顶是链表的头部，这样push和pop操作的时间复杂度都是O(1)。这是因为在链表的头部插入和删除元素都只需要常数时间。这种实现方式比使用数组或者链表的尾部作为栈顶要高效。因为在数组或者链表的尾部插入和删除元素可能需要线性时间（当需要扩展数组或者遍历链表时）。