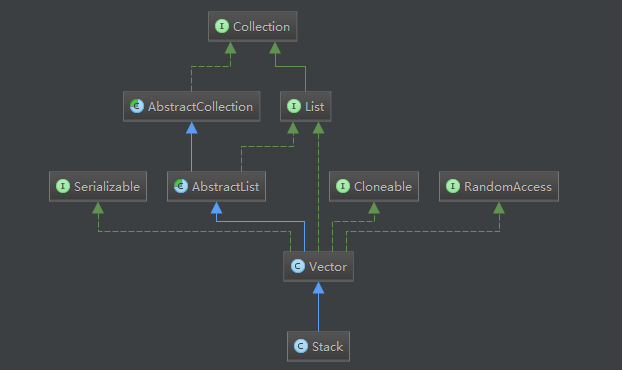
**一、Stack源码分析**

**1.继承结构**

栈是数据结构中一种很重要的数据结构类型，因为栈的**后进先出**功能是实际的开发中有很多的应用场景。Java API中提供了栈（Stack）的实现。

Stack类继承了Vector类，而Vector类继承了AbstractList抽象类，实现了List接口，Cloneable接口，RandomAcces接口以及Serializable接口，需要指出的Vector内部还有两个内部类ListItr和Itr，Itr在继承Vector的同时实现了Iterator接口，而ListItr在继承了Itr类的同时实现了ListIterator接口。



**2.构造方法**

**public** Stack() { //一个无参构造方法，能直接创建一个Stack

}

**3.**

其他值的方法是从Vector类继承而来，通过源码可见Vector有几个属性值：

**protected** Object[] elementData;

//用于保存Stack中的每个元素

**protected** **int** elementCount;

//用于动态的保存元素的个数，即实际元素个数

**protected** **int** capacityIncrement;

//用来保存Stack的容量（一般情况下大于elementCount）

**private** **static** **final** **int** ***MAX\_ARRAY\_SIZE*** = Integer.***MAX\_VALUE*** - 8;

//用于限制Stack能够保存的最大值数量

通过这几属性我们可以发现，**Stack底层是采用数组来实现的**。

**4.常用方法**

（1）public E push(E item) //向栈顶压入一个项

**public** E push(E item) {

// 调用Vector类里的添加元素的方法

addElement(item);

**return** item;

}

**public** **synchronized** **void** addElement(E obj) {

// 通过记录modCount参数来实现Fail-Fast机制

modCount++;

// 确保栈的容量大小不会使新增的数据溢出

ensureCapacityHelper(elementCount + 1);

elementData[elementCount++] = obj;

}

**private** **void** ensureCapacityHelper(**int** minCapacity) {

// 防止溢出。超出了数组可容纳的长度，需要进行动态扩展！！！

**if** (minCapacity - elementData.length > 0)

grow(minCapacity);

}

// 数组动态增加的关键所在

**private** **void** grow(**int** minCapacity) {

**int** oldCapacity = elementData.length;

// 如果是Stack的话，数组扩展为原来的两倍

**int** newCapacity = oldCapacity + ((capacityIncrement > 0) ?

capacityIncrement : oldCapacity);

// 扩展数组后需要判断两次

// 第1次是新数组的容量是否比elementCount+1的小(minCapacity)

**if** (newCapacity - minCapacity < 0)

newCapacity = minCapacity;

// 第1次是新数组的容量是否比指定最大限制Integer.MAX\_VALUE-8大

// 如果大，则minCapacity过大，需要判断下

**if** (newCapacity - ***MAX\_ARRAY\_SIZE*** > 0)

newCapacity = *hugeCapacity*(minCapacity);

elementData = Arrays.*copyOf*(elementData, newCapacity);

}

// 检查容量的int值是不是已经溢出

**private** **static** **int** hugeCapacity(**int** minCapacity) {

**if** (minCapacity < 0) // overflow

**throw** **new** OutOfMemoryError();

**return** (minCapacity > ***MAX\_ARRAY\_SIZE***) ?

Integer.***MAX\_VALUE*** :

***MAX\_ARRAY\_SIZE***;

}

（2）public synchronized E peek() //查找栈顶对象，而不从栈中移走

**public** **synchronized** E peek() {

**int** len = size();

**if** (len == 0)

**throw** **new** EmptyStackException();

**return** elementAt(len - 1);

}

// Vector里的方法，获取实际栈里的元素个数

**public** **synchronized** **int** size() {

**return** elementCount;

}

**public** **synchronized** E elementAt(**int** index) {

**if** (index >= elementCount) {

// 数组下标越界异常

**throw** **new** ArrayIndexOutOfBoundsException(index +

" >= " + elementCount);

}

// 返回数据下标为index的值

**return** elementData(index);

}

@SuppressWarnings("unchecked")

E elementData(**int** index) {

**return** (E) elementData[index];

}

（3）public synchronized E pop() //移走栈顶对象，将该对象作为

**public** **synchronized** E pop() {

E obj;

**int** len = size();

obj = peek();

// len-1的得到值就是数组最后一个数的下标

removeElementAt(len - 1);

**return** obj;

}

// Vector里的方法

**public** **synchronized** **void** removeElementAt(**int** index) {

modCount++;

// 数组下标越界异常出现的情况

**if** (index >= elementCount) {

**throw** **new** ArrayIndexOutOfBoundsException(index +

" >= " + elementCount);

}**else** **if** (index < 0) {

**throw** **new** ArrayIndexOutOfBoundsException(index);

}

// 数组中index以后的元素个数，由于Stack调用的该方法，j始终为0

**int** j = elementCount - index - 1;

**if** (j > 0) {

// 数组中index以后的元素，整体前移，（这个方法挺有用的！！）

System.*arraycopy*(elementData, index + 1,

elementData, index, j);

}

elementCount--;

elementData[elementCount] = **null**; /\* to let gc do its work \*/

}

（4）public boolean empty() //测试栈是否为空

**public** **boolean** empty() {

**return** size() == 0;

}

（5）public synchronized int search(Object o)//返回栈中对象的位置

**public** **synchronized** **int** search(Object o) {

**int** i = lastIndexOf(o);

**if** (i >= 0) {

**return** size() - i;

}

**return** -1;

}

// Vector里的方法

**public** **synchronized** **int** lastIndexOf(Object o) {

**return** lastIndexOf(o, elementCount-1);

}

**public** **synchronized** **int** lastIndexOf(Object o, **int** index) {

**if** (index >= elementCount)

**throw** **new** IndexOutOfBoundsException(index +

" >= "+ elementCount);

// Vector、Stack里可以放null数据

**if** (o == **null**) {

**for** (**int** i = index; i >= 0; i--)

**if** (elementData[i]==**null**)

**return** i;

} **else** {

**for** (**int** i = index; i >= 0; i--)

**if** (o.equals(elementData[i]))

**return** i;

}

**return** -1;

}