Stage是指应用程序的一个窗口

Scene是一个容器，用来装载若干个Node节点，通过这个Node节点的组合构成Stage的内容。

1：JavaFX应用程序的主类扩展了javafx.application。应用程序类。start()方法是所有JavaFX应用程序的主入口点。

2：JavaFX应用程序定义了用户界面的容器的一个舞台,一个场景。JavaFX的Stage类是顶级JavaFX容器。JavaFX的Scene类是所有内容的容器。该例创造舞台和场景,使场景以给定的像素大小中可见。

3：在JavaFX,场景的内容表示为一个层次场景图的节点。在这个例子中,根节点是一个StackPane对象,这是一个可调整大小的布局节点。这意味着,当用户改变场（Scene）景大小或者舞台（Stage）大小时，根节点也会跟着改变。

4：根节点包含一个孩子节点,一个按钮控制文本,再加上一个事件处理程序来打印一个消息当按钮被按下。

CSS支持

所有的JavaFX控件可以通过CSS来定义皮肤。每个控件都有一组属性，诸如前景的颜色或字体大小，与相关的其外观屏幕，而这些性质可通过定义CSS来实现。一个很好的例子显示，你可以在JavaFX中使用AquaFX库提供皮肤应用到到所有默认的JavaFX控件，使它们看起来就像原生的Mac OS控件。你可以在网站http://aquafx-project.com找到相关的开源库和文档。

通过使用CSS，你可以定义一个新的外观为一个类型控件或单一实例。以下代码片段显示了一些CSS代码定义的一个按钮的皮肤：

```
。custom-button{
   -fx-padding:10;
   -fx-background-color:#ffaa99;
   -fx-font:24px "Serif";
```

当使用控件或者自定义他们的工作中，你会经常使用CSS和JavaFX内部的API。

一个JavaFx应用可以通过设置每个UI元素的位置和大小来手动地布局用户界面。但是，一个更简单的做法是使用布局窗格。JavaFx SDK提供了多种布局容器类，叫做窗格，它们可简化对一些经典布局的设置和管理，例如行、列、堆叠、平铺等等。当窗口缩放时，布局窗格会自动地根据节点属性重设其包含的所有节点的位置和大小。

BorderPane

BorderPane布局窗格提供了5个放置节点的区域：top, bottom, left, right, 和 center。这些区域可以是任意大小，如果应用不需要某个区域，你可以不定义它，然后窗格就不会给这个区域分配空间。

border pane可用于这种经典的外观：top:工具栏，bottom:状态栏，left:导航栏，right:附加信息，center:工作区。

HBox

HBox 布局窗格可以让你很容易地将一系列节点排列到一行中。

Padding 属性可以设置节点到 HBox 边缘的距离。 Spacing 可以管理节点之间的距离。Style可用来改变背景色。

VBox

VBox 布局窗格和 HBox 很相似，除了这里所有节点是被排列到一个列中的。

StackPane 布局窗格能将所有的节点放到一个堆栈中，其中每一个新的节点被添加到前一个节点的上方。这个布局模型能让你很容易地在一个形状或图像上面覆盖一个文本，或者用常用形状互相覆盖来创建复杂的形状

GridPane布局窗格能让你创建一个灵活的由行和列组成的网格来放置节点。节点可以被放于任何单元格内，也可以根据需要横跨多个单元格。一个grid pane对于创建表单或者任何以行和列组织的布局很有用。

FlowPane 中的节点会连续地排列，并且会在窗格的边界自动换行（或列）。节点可以垂直地（按列）或水平地（按行）流动。

Grid

Gap属性可用于管理行和列之间的间隔。Padding属性可用于管理节点与网格边缘的距离。Vertical和horizontal alignment属性可用于管理单元格中单独控件的对齐方式。

事件（Events ） 一个事件代表了对应用有意义的事情的出现，如移动鼠标、敲击键盘等。在JavaFX中，一个事件是javafx.event.Event类的实例，或是任意的Event子类。JavaFX提供了好几种事件，包括DragEvent、 KeyEvent、 MouseEvent、 ScrollEvent以及其它。也可扩展Event类来定义自

push.setOnAction(this::processButtonPrerss)

双向链表(Doubly Linked List)

1. 双向链表的概念

1.1 双向链表的定义

双向链表又称为双链表，是链表的一种。

1.2 双向链表的结点结构

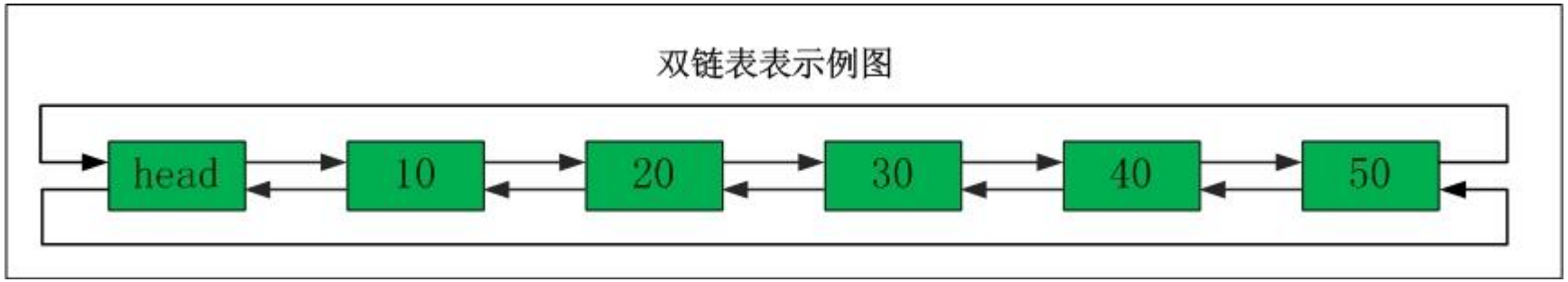
双向链表的结点包括三个部分：前驱指针域、数据域和后继指针域。

(1)前驱指针域(lLink)，又称为左链指针，用于存放一个指针，该指针指向上一个结点的开始存储地址。

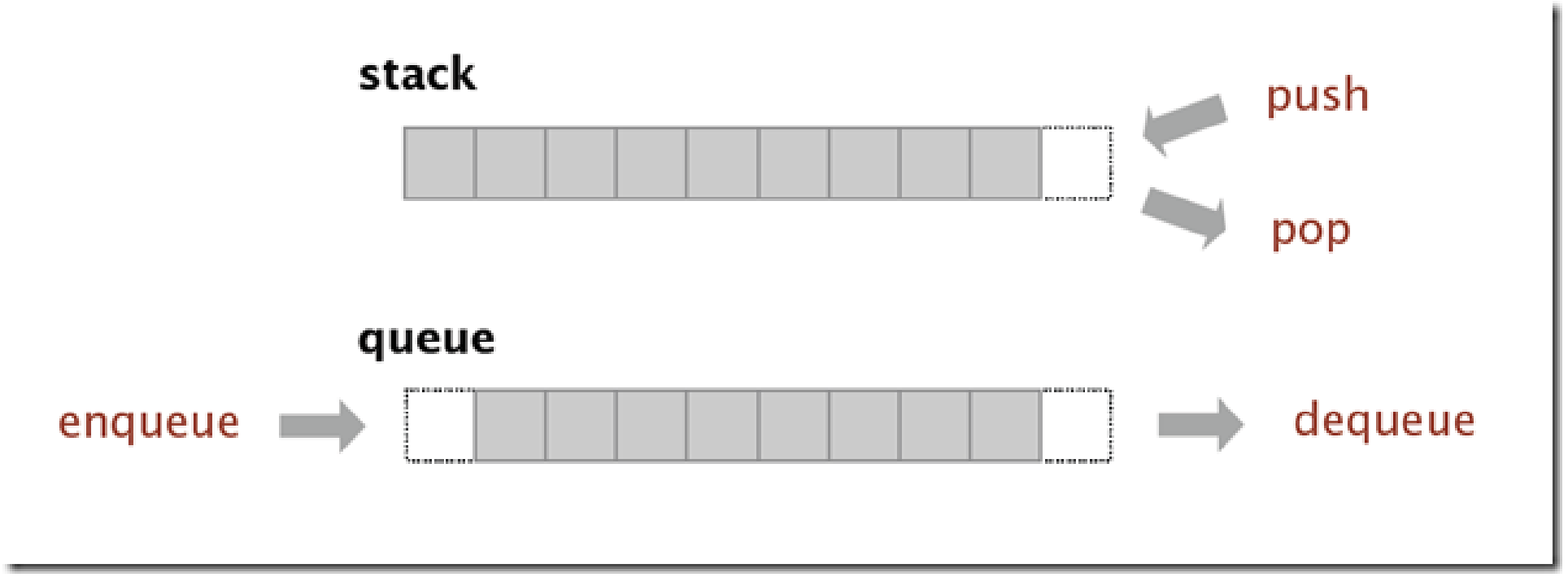
(2)数据域(data)，用于存储该结点的数据元素，数据元素类型由应用问题决定。

(3)后继指针域(rLink)，又称为右链指针，用于存放一个指针，该指针指向下一个结点的开始存储地址。

表头为空，表头的后继节点为"节点10"(数据为10的节点)；"节点10"的后继节点是"节点20"(数据为10的节点)，"节点20"的前继节点是"节点10"；"节点20"的后继节点是"节点30"，"节点30"的前继节点是"节点20"；...；末尾节点的后继节点是表头。



栈 (Stack)是一种后进先出(last in first off，LIFO)的数据结构，而队列(Queue)则是一种先进先出(fisrt in first out，FIFO)的结构



将中缀表达式转换为后缀表达式：

(1) 初始化两个栈：运算符栈S1和储存中间结果的栈S2；
(2) 从左至右扫描中缀表达式；
(3) 遇到操作数时，将其压入S2；
(4) 遇到运算符时，比较其与S1栈顶运算符的优先级：
(4-1) 如果S1为空，或栈顶运算符为左括号“(”，则直接将此运算符入栈；
(4-2) 否则，若优先级比栈顶运算符的高，也将运算符压入S1（注意转换为前缀表达式时是优先级较高或相同，而这里则不包括相同的情况）；
(4-3) 否则，将S1栈顶的运算符弹出并压入到S2中，再次转到(4-1)与S1中新的栈顶运算符相比较；
(5) 遇到括号时：
(5-1) 如果是左括号“(”，则直接压入S1；
(5-2) 如果是右括号“)”，则依次弹出S1栈顶的运算符，并压入S2，直到遇到左括号为止，此时将这一对括号丢弃；
(6) 重复步骤(2)至(5)，直到表达式的最右边；
(7) 将S1中剩余的运算符依次弹出并压入S2；
(8) 依次弹出S2中的元素并输出，结果的逆序即为中缀表达式对应的后缀表达式（转换为前缀表达式时不用逆序）。

例如，将中缀表达式“1+((2+3)×4)-5”转换为后缀表达式的过程如下：

扫描到的元素	S2(栈底->栈顶)	S1 (栈底->栈顶)	说明
1	1	空	数字，直接入栈
+	1	+	S1为空，运算符直接入栈
(	1	+ (	左括号，直接入栈
(	1	+ ((	同上
2	1 2	+ ((	数字
+	1 2	+ ((+	S1栈顶为左括号，运算符直接入栈
3	1 2 3	+ ((+	数字
)	1 2 3 +	+ (	右括号，弹出运算符直至遇到左括号
×	1 2 3 +	+ (×	S1栈顶为左括号，运算符直接入栈
4	1 2 3 + 4	+ (×	数字
)	1 2 3 + 4 ×	+	右括号，弹出运算符直至遇到左括号
-	1 2 3 + 4 × +	-	-与+优先级相同，因此弹出+，再压入-
5	1 2 3 + 4 × + 5	-	数字
到达最右端	1 2 3 + 4 × + 5 -	空	S1中剩余的运算符

运用后缀表达式进行计算：

(1)建立一个栈S；

(2)从左到右读后缀表达式，读到数字就将它转换为数值压入栈S中，读到运算符则从栈中依次弹出两个数分别到Y和X，然后以“X 运算符 Y”的形式计算机出结果，再压加栈S中；

(3)如果后缀表达式未读完，就重复上面过程，最后输出栈顶的数值则为结束。

举例：

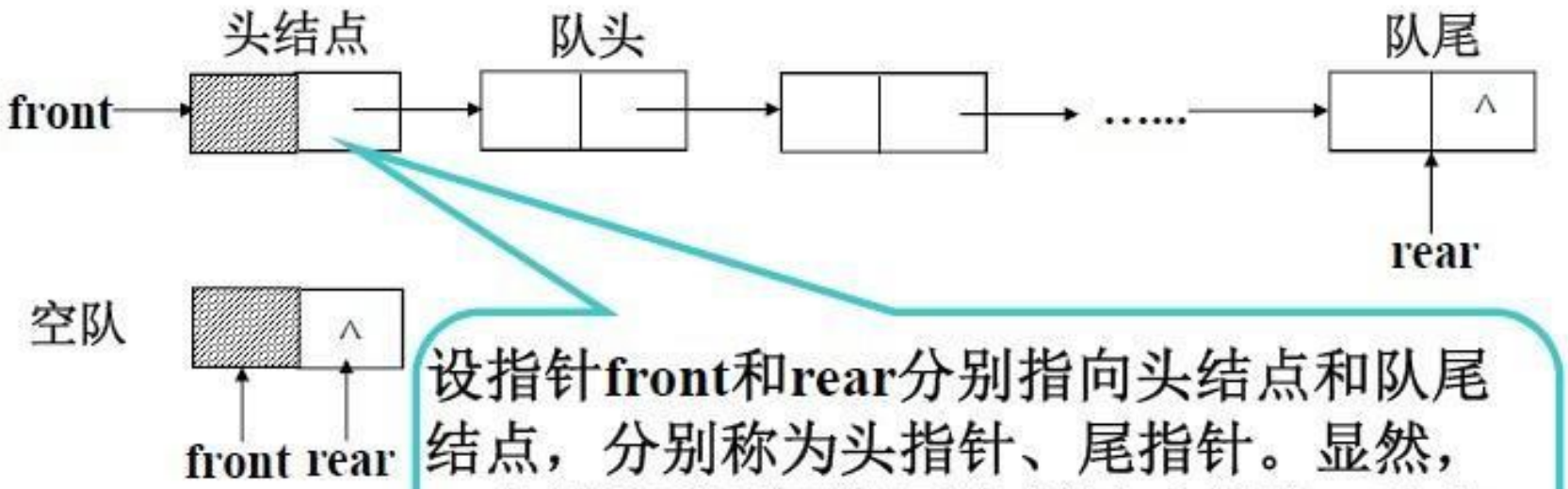
3+(2-5)*6/3=-3 ,其后缀表达式为：325-6*3/+。其运算结果如下：

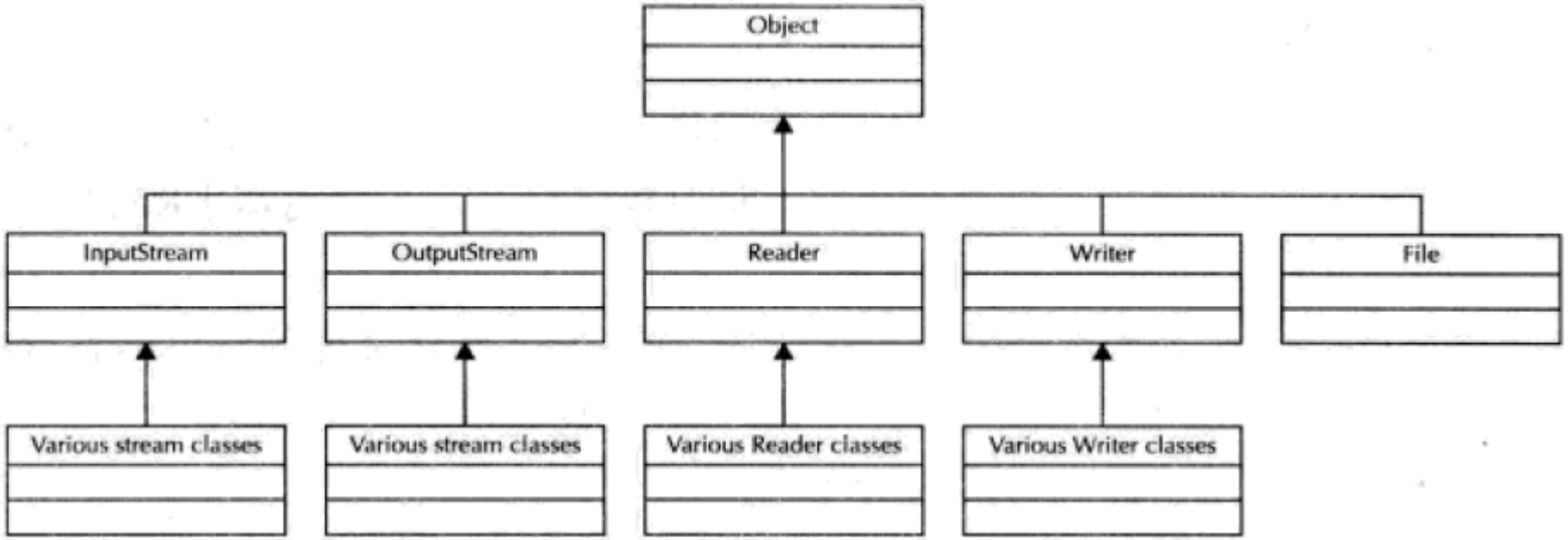
- 3+(2-5)*6/3=-3 ，其后缀表达式为：325-6*3/+
- 运算过程如下：

栈	运算	
3 2 5		325入栈
3	2-5=-3	
3 -3		运算结果进栈
3 -3 6		
3	-3*6=-18	
3 -18 3	-18/3=-6	
3 -6	3+(-6)=-3	
-3		

（1）入队（Enqueue）：将一个数据元素插入队尾；

（2）出队（Dequeue）：读取队头节点数据并删除该节点；





Java中所有的类都继承自Object类，所以各种IO类也不例外。InputStream和OutputStream类操作字节数据。Reader和Writer类工作在字符是上。File类为文件提供接口。

我们很有必要了解IO类的演变历史，如果缺乏历史的眼光，那么我们对什么时候该使用那些类，以及什么时候不该使用那些类而感到迷惑。
JDK1.0的时候，所有与输入相关的类都继承于InputStream，所有与输出相关的类都继承于OutputStream。
JDK1.1的时候，增加了面向字符的IO类，包括Reader和Writer。