链表数据结构实现

# 引言

1. 链表的基本形式；
2. 单向链表的完整实现。

# 具体内容

## 认识链表

链表本质上就等于可变长度的对象数组，属于动态对象数组的范畴。

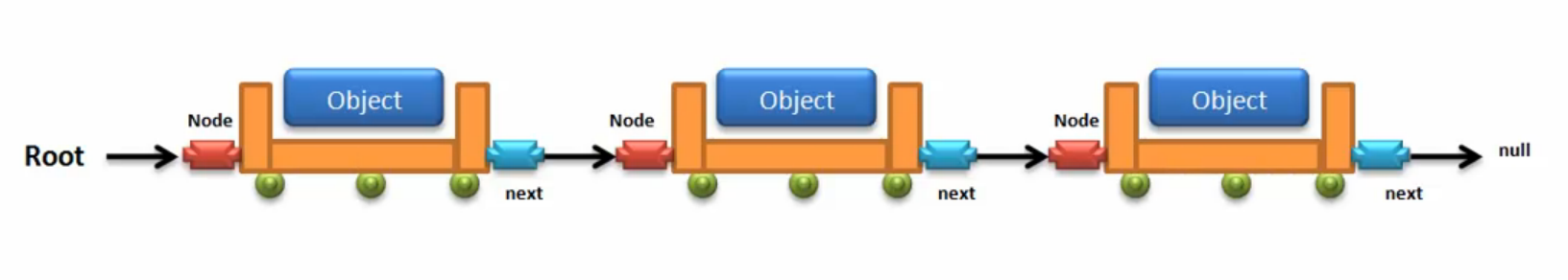
对象数组有哪些问题呢？

* 对象数组可以保存一组对象方便开发；
* 对象数组的长度固定，而且数组的删除、修改、增加处理麻烦。

所有的开发之中都100%不可能避免对象数组的使用。正因为如此如果要想让其可以编写出便于维护的代码，那么就需要实现一个动态对象数组，那么就可以使用链表完成。

但是现在如果要想实现动态的对象数组，要考虑两个问题：

* 为了适应于开发要求，此对象数组要求可以保存所有的数据类型，那么一定首选Object类型；
* 为了可以保存多个数据，需要采用引用的方式来进行保存，但是数据本身是不可能保存顺序的，那么就需要一个可以负责保存顺序的类来包装这个数据。



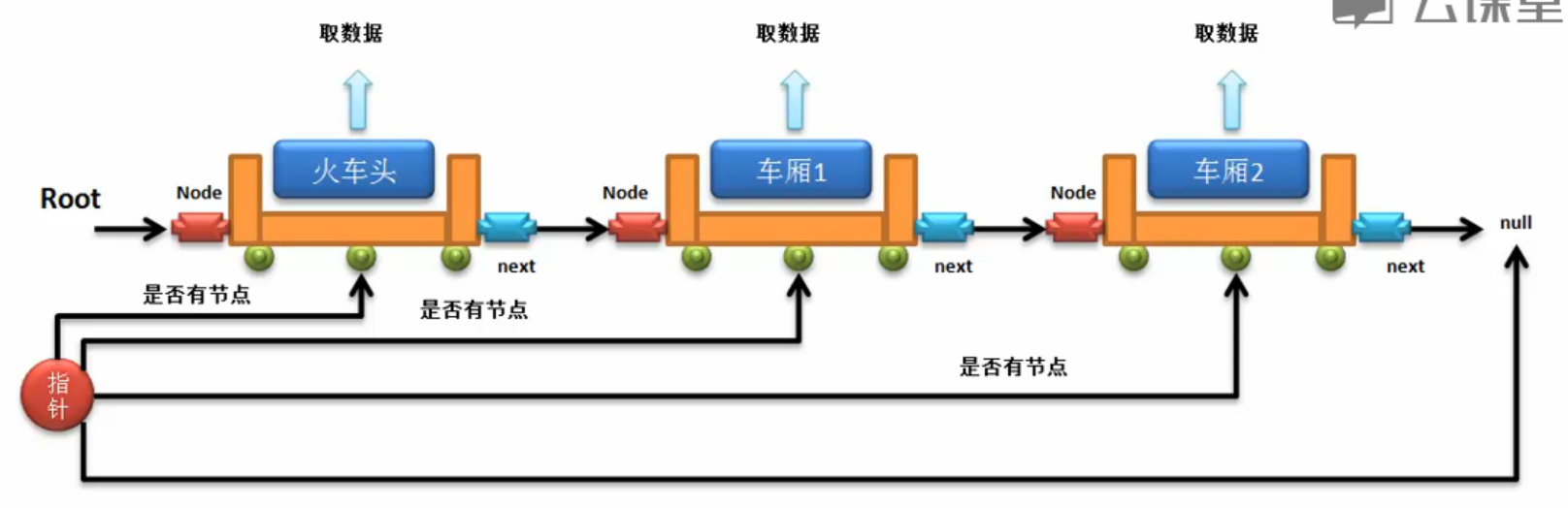
通过以上的分析就可以得出如下的结论：

* 保存数据为了方便使用Object；
* 数据本身不包含先后的逻辑关系，所以讲数据封装在一个Node类中，负责关系的维护。

范例：定义出如下的一个类

|  |
| --- |
| */\*\*  \* 表示定义的节点  \* Created by Dougin on 2017/4/12.  \*/* **public class** Node {  **private** Object **data**;  **private** Node **next**;  **public** Node(Object data) {  **this**.**data** = data;  }  **public void** setNext(Node next) {  **this**.**next** = next;  }  **public** Node getNext() {  **return this**.**next**;  }  **public** Object getData() {  **return this**.**data**;  } } |

完成节点之后，下面就可以进行节点的基本使用了。



范例：采用循环的方式操作节点

|  |
| --- |
| **public class** Main {  **public static void** main(String[] args) {  *// 1、定义各自独立的操作节点* Node root = **new** Node(**"火车头"**);  Node n1 = **new** Node(**"车厢1"**);  Node n2 = **new** Node(**"车厢2"**);  *// 2、设置彼此间的关系* root.setNext(n1);  n1.setNext(n2);  *// 3.输出* Node currentNode = root;  **while** (currentNode != **null**) {  System.***out***.println(currentNode.getData());  currentNode = currentNode.getNext();  }  } } |

以上的操作如果使用循环并不方便。最好的做法是递归调用。

范例：利用递归的方式实现内容的取得

|  |
| --- |
| **public class** Main {  **public static void** main(String[] args) {  *// 1、定义各自独立的操作节点* Node root = **new** Node(**"火车头"**);  Node n1 = **new** Node(**"车厢1"**);  Node n2 = **new** Node(**"车厢2"**);  *// 2、设置彼此间的关系* root.setNext(n1);  n1.setNext(n2);  *// 3.输出  print*(root);   }  **public static void** print(Node node) {  **if** (node == **null**) {  **return** ;  }  System.***out***.println(node.getData());  *print*(node.getNext());  } } |

链表在整个实现的关键就是Node类，Node类要保存数据与下一个节点。

## 链表开发入门

虽然以上的代码已经实现了链的形式，但是以上的代码之中存在有两个问题：

* 用户需要自己手工定义Node类，但是实际上Node类对用户没用；
* Node类的先后关系如果交由用户处理，那么整个代码就乱了。

所以现在发现，需要有一个类，这个类可以负责所有的Node的关系匹配，而用户只需要通过这个类保存数据或取得数据即可，那么就可以编写一个Link类完成。

## 开发可用链表

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

# 总结

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |