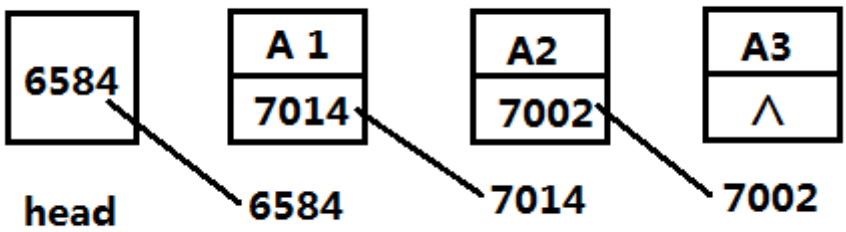
线性表的链式存储结构

```
function A{}
let x = new A()
//x储存的是A实例的地址
let h = x
x = 1
//此时h为new A(),x为1。
```

链式存储结构

链式存储结构：使用若干地址分散的存储单元存储数据元素。

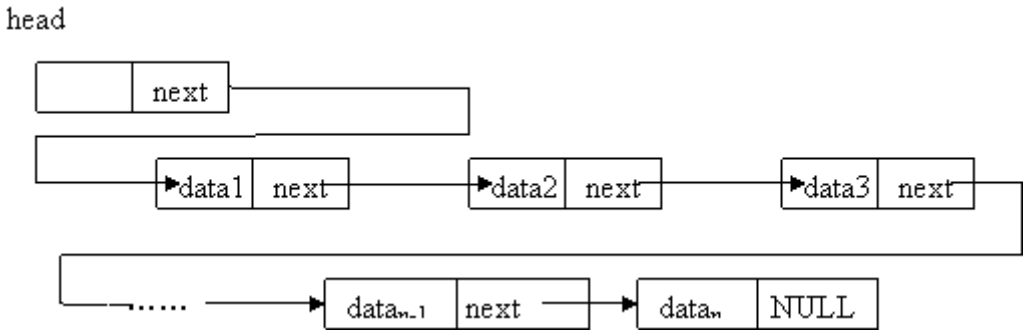


> 1. 当然我们需要一个地址域记住头元素的地址。
> 2. 不开辟连续的存储空间（可连可不连）
> 3. 逻辑的相连，地址不一定相连。
> 4. 数据域和地址域组成一个节点表示一个数据元素。

结点（Node）：这种储存方式的储存单元和链式存储结构不同，一个存储单元包含地址域和数据域合成

线性链表（Linked LIst）：采用链式存储结构的线性表。

单链表（Single Linked List）



> 每个节点只有一个地址域的线性链表

ADT表示

> 单链表有一个一个节点连接而成，以下定义结点类和单链表类来描述单链表。

单链表节点类

```java
public class Node<T>{
    public T data;
    public Node<T> next;        //就实例而言，next储存的确实是下一个结点的物理地址。
    public Node(T data,Node<T> next){
        this.data = data;
        this.next = next;
    }
    public Node(){
        this(null);
        this(null);
    }
    public String toString()
        return this.data.toString();
    }
}
```

> Node<T> 是自引用类，这里相当于声明了一个储存对象时当前类实例的成员变量。

结点的接本操作

```
n1.data
n1.next.data
```

头结点

```
Node<T>head = null //（空表）
head.next = Node<T>("1",null);
```

> 链表的头结点也是Node结点的实例。

单链表的基本操作

1.遍历

遍历操作不能改变链表的头指针，所以需要声明一个结点实例p来遍历。

```
Node<T> p = head
while(p!=null){
    System.out.print(p.data.toSting())
    p = p.next;
}
```

> 注意不要对p使用改变操作元素的语句，因为此时p和实际结点引用的是同一内存地址。

2.空表插入

```
Node<T> head = null;    //创建空表。
int x = 1;
head = new Node<T>(x.null); //储存第一个元素。
```

3.头插入

```
Node<T>p = new Node<T>(x,null);      //创建要插入的结点
p.next = head;                       //赋予结点next正确的指向
head = p;                            //改变head的指向为p
```

[链表非空（head != null）]