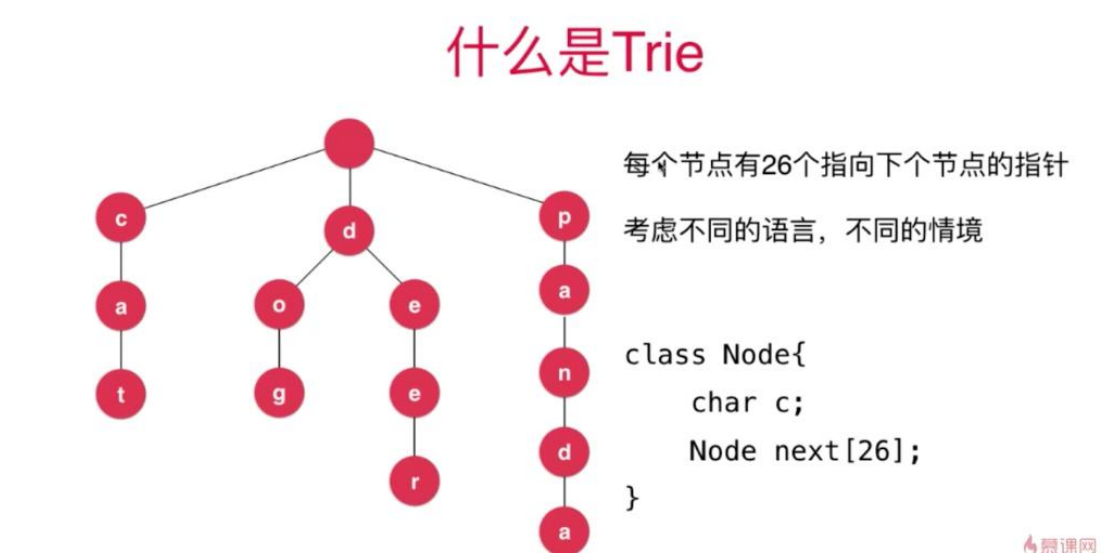
前缀树（字典树）：Trie

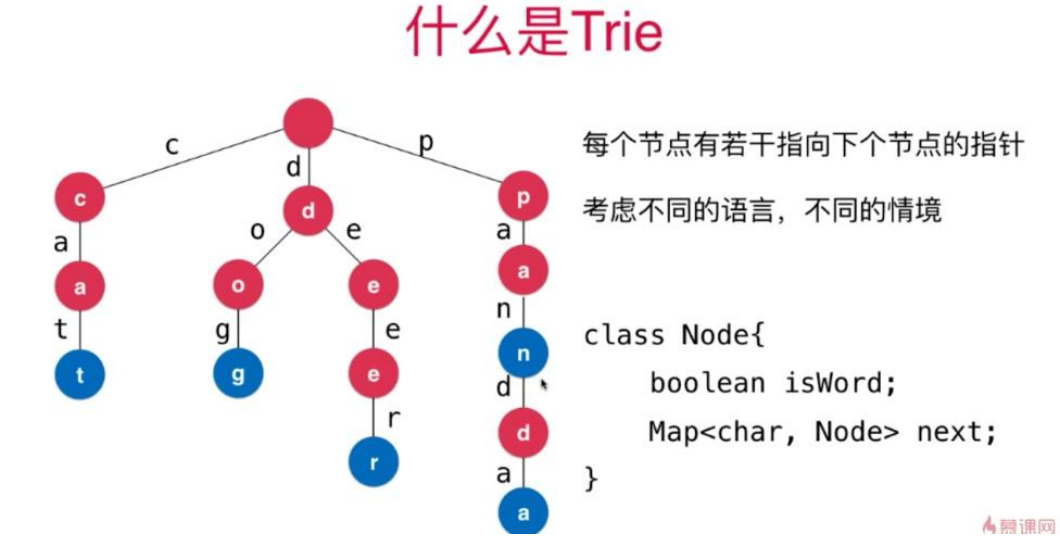
- 对于字符的匹配问题，我们可以采用字典树的形式来辅助我们完成，但是需要注意的是，我们一般将Trie用于字符的比较上

- Trie的结构
  - 第一种方式：我们构造如下的数据结构，一个Node节点中存放当前节点的值以及当前节点的26个可能的子节点，但是，这会带来一些问题。当我们考虑的字符范围不同时，节点的子孩子节点的数量可能就不一样，如果我们考虑字母的大小的话，长度为26的数组就不足以存放所有可能的值。



  - 第二种方式：我们不采用固定的数组元素存放子节点，我们使用一个Map存放下一节点的值以及下一个节点的引用(Map可以存放多个节点)。当我们采用这种方式时，我们是知道下一个节点的值是什么的，也就知道该如何找到下一个节点（Map中的Key）。此时我们需要添加一个标志位来表示字符是否是最后一位。

    当前节点不知道自身的值，但是上一个节点知道他所指向的下一个节点的值。



- 对于`Trie`我们一般只实现插入和搜索操作

```
//树的构建过程不需要用户判断
//用户不需要有树的相关知识，也不应该让用户判断单词是否已经到结尾

public class Trie{
    private class Node{
        boolean isWord;
        Map<Character,Node> next;

        public Node(){
            Node(false);
        }

        public Node(boolean isWord){
            this.isWord = isWord;
            next = new TreeMap();
        }
    }

    private Node root;
    private int size;

    public Trie(){
        root = new Node();
        size = 0;
    }

    /**
    用户不需要维护树，也就不需要用户传入单词的结束位置
    public Trie(boolean isWord){
        root = new Node(isWord);
    }
    */

    //向Trie中添加元素
    public void add(String word){
        if(word == null)
            return;
        Node cur = root;
```

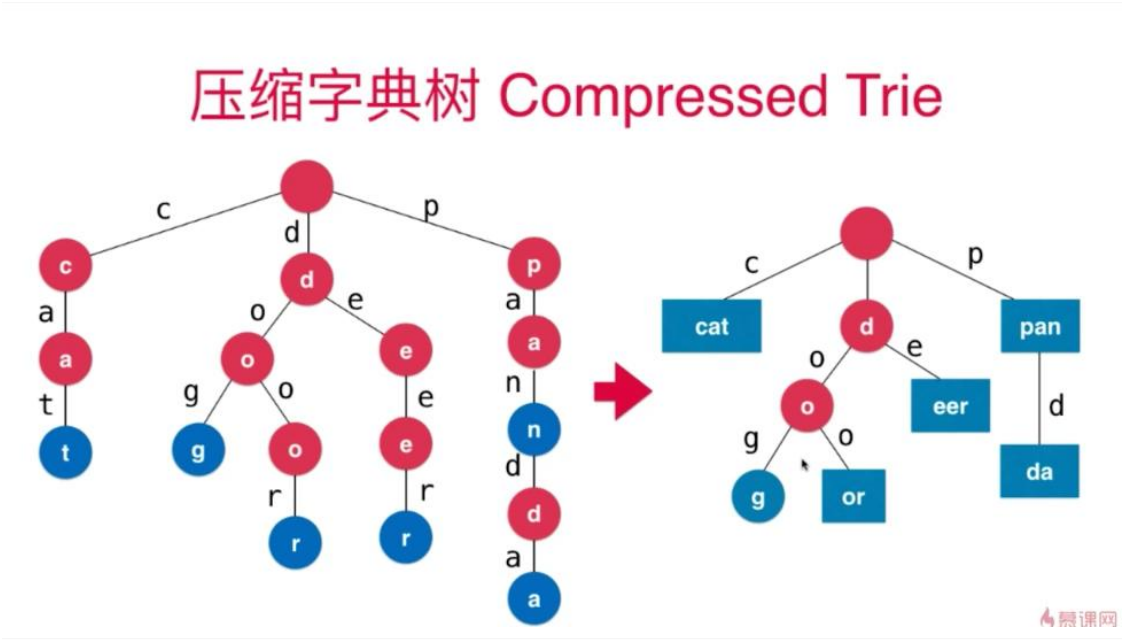
```
        for(int i=0;i<word.length();i++){
            char c = word.charAt(i);
            if(cur.next.get(c)==null)
                cur.next.put(c,new Node());
            //指向下一个节点
            cur = cur.next.get(c);
        }
        //循环结束时会指向单词的结尾节点
        if(!cur.isWord){
            cur.isWord = true;
            size++;
        }
    }


    //查询是否包含某个单词
    public boolean search(String word){
        if(word == null)
            return false;

        Node cur = root;
        for(int i=0;i<word.length();i++){
            char c = word.charAt(i);
            if(cur.next.get(c)==null)
                return false;
            cur = cur.next.get(c);
        }
        //if(!cur.isWord)
            //return false;
        //return true;
        return cur.isWord;
    }

    // 查询是否在Trie中有单词以prefix为前缀
    public boolean isPrefix(String prefix){
        if(word == null)
            return false;
        Node cur = root;
        for(int i=0;i<word.length();i++){
            char c = word.charAt(i);
            if(cur.next.get(c) == null)
                return false;
            cur = cur.next.get(c);
        }
        return true;
    }
     /** Returns if the word is in the data structure. A word could contain
the dot         character '.' to represent any one letter. */
    public boolean search(String word){

    }

    //node 当前节点
    //word 要搜索的前缀
    //index 当前检索到的词的位置
    private boolean match(Node node,String word,int index){
        if(index == word.length())
```

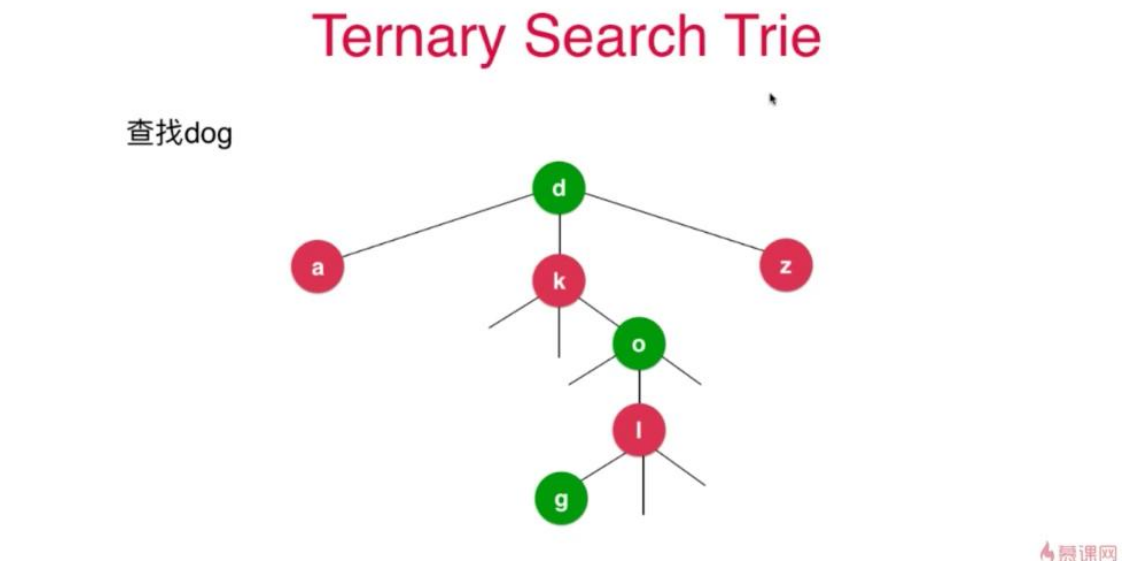
```
            return node.isWord;
        char c = word.charAt(index);
        if('.'.equals(c)){
            if(node.next.get(c) == null)
                return false;
            return match(node.next.get(c),word,index+1);
        }else{
            for(char c:node.next.keySet()){
                Node cur = node.next.get(c);
                if(match(cur,word,index+1))
                    return true;//有一个匹配上了就返回true
            }
            return false;
        }
    }
}
```

添加一个当前节点的值，或许会更好理解，因为Map中存放的都是下一节点的值，默认为NULL

- 扩展内容

  Trie最大的局限性在于其**空间利用率不高**





更多字符串问题

子串查询

KMP　　Boyer-Moore　　Rabin-Karp