使用位运算，非常巧妙，代码如下：

```
public boolean isValidSudoku2(char[][] board){
        int[] row = new int[9];
        int[] column = new int[9];
        int[] cell = new int[9];
        int binary;
        for (int i = 0; i < 9; i++) {
            for (int j = 0; j < 9; j++) {
                char value = board[i][j];
                //如果还没有填数字，直接跳过
                if (value == '.')
                    continue;
                binary = 1 << (value - '0');
                int k = (i / 3) * 3 + j / 3;
                //如果对应的位置只要有一个大于0，说明有冲突，直接返回false
                if ((row[i] & binary) > 0 || (column[j] & binary) > 0 || (cell[k] & binary)
> 0){
                    return false;
                }
                row[i] |= binary;
                column[j] |= binary;
                cell[k] |= binary;
            }
        }
        return true;
}
```

因为Java的int类型，是用4个字节表示的，也就是说一共32位，那我们可以使用其中的9位来表示每一行、 每一列、每一个子单元格中出现的数字。下面我们用数组来解释一下。

```
{'8', '3', '.', '.', '3', '.', '.', '.', '.'},
{'6', '.', '.', '1', '9', '5', '.', '.', '.'},
{'.', '9', '8', '.', '.', '.', '.', '6', '.'},
{'8', '.', '.', '.', '6', '.', '.', '.', '3'},
{'4', '.', '.', '8', '.', '3', '.', '.', '1'},
{'7', '.', '.', '.', '2', '.', '.', '.', '6'},
{'.', '6', '.', '.', '.', '.', '2', '8', '.'},
{'.', '.', '.', '4', '1', '9', '.', '.', '5'},
{'.', '.', '.', '.', '8', '.', '.', '7', '9'}
```

这里先只关注行的情况，结合上面的代码，第一行第一个字符为8，binary = 1 << (value - '0')，是说遇到值value，就向左数value位置1，（为啥置1.想必不需要解释了，因为2进制最大只能是1..）,也就是说，1
向左移8位，转换成2进制也就是100000000,即256，那么row[0]=256,再走到3这个字符的时候，左移三位，得到1000，
，即8. 256&8=0,所以判断不成立，row[0]=256|8. 即100001000. 那么走到第二个3的时候，得到binary仍旧是8，即1000，
这时 row[0]=256. 256&8= 100001000&1000=1000>0。不符合条件，结束。行和子单元格同理。

如果遇到位运算，我们一定要站在二进制的角度上思考问题，不然，像256&8这样的10进制操作，很难看出其中端倪。

另外这道题还有一个精妙的地方，即如何划分子单元格.9*9的单元格,划分成三个3*3的，横向划分可以以下公式：k=(i/3)*3+j/3,那么竖向划分，想必也很好理解了. 16X16的单元格，划分成4个4X4的，应该也很好理解了