用程序实现Enigma报告

蒋程 2017060201009

原理:

使用者按下第一个字母后，字母会先经过三个可以转动的转盘，称为转字，每一个转子即一个字符映射表。如下:

输入	A	B	C	D	E
输出	D	E	A	B	C

一个转子的输出作为下一个转子的输入。当经过第三个转子后，输出会被送入到一个反射器。反射器和转子一样，区别在于不能转动，且映射是对称的，即如果输入c1会输出c2，那么输入c2将输出c1。经过反射器后，字母会依次反向进入三个转子，并最终显示在显示盘上。至此算完成了一个字母的加密。当加密下一个字母时，第一个转子会转动一格，转子的转动代表着映射表发生变化。如上表在转动后将变成。

输入	A	B	C	D	E
输出	E	A	B	C	D

当转子转动一周再次回到**初始位置**时，会带动第二个转子转动一格，同理第二个转动一周后，会带动第三个转子转动。

本次实转子和反射器如下:

- 转子1: m l x w c d r s o h e j z f i a k p v t u b n y g q
- 转子2: p y o a k v e h l x t m j d f z q c u n i s r b w g
- 转子3: o j b k f y e t m n c z g p q u h s w r i a d v x l
- 反射器: s q m p t n x u j i v o c f l d b w a e h k r g z y

使用的时候，三个转子能够在初始化的时候自由的调节顺序，即保证了加密操作前的初始的状态拥有足够的多。

代码:

```java
package cryptology;

public class Enigma {
    private Rotor mRotor;

    /**
     * @param max 转子的个数
     * @param keys 密钥
     */
    public Enigma(int max, int[] keys) {
        mRotor = new Rotor(0, max);
        if (keys.length == max) {
            Rotor ptr = mRotor;
            int pos = 0;
            while (pos < max - 1) {
                ptr.move(keys[pos++] % 26);
                ptr = (Rotor) ptr.mNext;
            }
        }
    }


    /**
     * 对外暴露的加密方法
     *
     * @param msg 明文
     * @return 密文
     * @see #encryptCharSequence(char[])
     */
    public String getEncryptMsg(String msg) {
        return encryptCharSequence(msg.toCharArray());
    }

    /**
     * 先对字符序列中的每一个元素进行编码，即a对应0，b对应1，z对应25，以方便操作
     * 在将每一个字符传入转子以获得映射后的密码
     *
     * @param chs 待加密的字符序列
     * @return 密文
     */
    private String encryptCharSequence(char[] chs) {
        StringBuilder builder = new StringBuilder();
        for (char ch : chs) {
            int code = enCode(ch);
            int result = mRotor.getCode(code);
            char newch = deCode(result);
            builder.append(newch);
        }
        return builder.toString();
    }

    /**
     * @param ch 输入的字符
     * @return 字符的编码
     */
    private int enCode(char ch) {
        return (int) ch - 'a';
    }
```

```
/**
 * @param num 编码
 * @return 字符
 */
private char deCode(int num) {
    return (char) (num + 'a');
}

/**
 * 反射器，实现了Encryptor接口
 */
private class Rotor implements Encryptor {
    private int[] mCodeSecuence;
    private Encryptor mNext;
    private int flag;
    private int count = 0;

    /**
     * 检测当前的输入次数，如足够，就转动
     */
    private void checkSelf() {
        count++;
        if (count == flag) {
            move(1);
            count = 0;
        }
    }

    /**
     * 转子转动的方法
     *
     * @param count 转子转动的次数
     */
    void move(int count) {
        for (int i = 0; i < count; i++) {
            for (int j = 0; j < mCodeSecuence.length; j++) {
                mCodeSecuence[j] = (mCodeSecuence[j] + 1) % 26;
            }
        }
    }

    /**
     * 转子的构造方法，采用递归创建，当深度触底的时候，创建反射器
     * 转子内部维护一个编码之间的映射序列{@link #mCodeSecuence}
     * 通过转子间映射序列相互传递从而达到加密的作用
     *
     * @param depth 当前转子的深度，即当前是第几个转子
     * @param max   总共的转子的数量
     */
    Rotor(int depth, int max) {
        flag = (int) Math.pow(26, depth);
        if (depth < max - 1) {
            mNext = new Rotor(depth + 1, max);
        } else {
            mNext = new Reflector();
        }
        mCodeSecuence = CharSequenceUtil.getInts();
    }
```

```
        /**
         * 实际获取加密编码的方法，此方法中会判断当前转子是否存在下一个
         * 若有，则将转化后的编码传递到下一个转子(反射器),并且自动处理
         * 下一个转子传递回来后的值。
         * <p>
         * 如果{@link #mNext}为空，那么处于当前的即是反射器{@link Reflector#getCode(int)}
         *
         * @param pos 传入反射器中的位置
         * @return 新的编码
         * @see #mNext#getCode(int)
         * @see #indexOf(int)
         */
        @Override
        public int getCode(int pos) {
            int result = mCodeSecuence[pos];
            if (mNext != null) {
                result = mNext.getCode(result);
                result = indexOf(result);
            }
            checkSelf();
            return result;
        }

        /**
         * 当转子对编码进行映射后会将结果传递给下一个转子(反射器)
         * 而下一个最终会回传它映射后的编码，这个时候需要对编码再做一次映射(再加密一次)
         *
         * @param result 由下一个转子回传的编码
         * @return 映射后的编码
         */
        private int indexOf(int result) {
            for (int i = 0; i < 26; i++) {
                if (mCodeSecuence[i] == result) {
                    return i;
                }
            }
            return 0;
        }

    }

    /**
     * 反射器
     */
    class Reflector implements Encryptor {
        private int[] nums = {18, 16, 12, 15, 19, 13, 23, 20, 9, 8, 21, 14, 2, 5, 11, 3, 1, 22,
0, 4, 7, 10, 17, 6, 25, 24};

        /**
         * {@link Rotor#getCode(int)}
         *
         * @param pos 转子传入的编码
         * @return 反射后的编码
         */
        @Override
        public int getCode(int pos) {
            return nums[pos];
        }
    }
```

```
    /**
     * 抽象接口，统一拥有getCode方法
     * {@link Rotor}
     * {@link Reflector}
     */
    interface Encryptor {
        int getCode(int pos);
    }
}
```

测试：

- 加密

> 输入：
>
> ```
> int[] keys = {2, 4, 6};
> Enigma enigma = new Enigma(3, keys);
> System.out.println(enigma.getEncryptMsg("helloworld"));
> ```
>
> 输出：zswxcbvdaa

- 解密

> 输入：
>
> ```
> int[] keys = {2, 4, 6};
> Enigma enigma = new Enigma(3, keys);
> System.out.println(enigma.getEncryptMsg("zswxcbvdaa"));
> ```
>
> 输出：helloworld