原子操作类

一，概述

- java.util.concurrent.atomic包中应该提供了13个类，属于4种类型的原子更新方式。

二，原子更新基本类型类

- AtomicBoolean：原子更新布尔类型
- AtomicInteger：原子更新整型
- AtomicLong：原子更新长整型
- 这三个类提供的方法几乎一模一样，以AtomicInteger为例
  - int addAndGet(int delta); 以原子的方式将输入的数值与实例中的值相加，并返回结果。
  - boolean compareAndSet(int expect,int update);如果输入的值等于预期值，则以原子的方式将该值设置为输入的值。
  - int getAndIncreament() ;以原子的方式加1，这里返回的是自增之前的值。
  - void lazySet(int newValue);最终会设置成newValue。

    代码样例：

```
import java.util.concurrent.atomic.AtomicInteger;

public class AtomicIntegerTest {
    private static AtomicInteger a = new AtomicInteger(1);

    public static void main(String[] args) {
        System.out.println(a.getAndIncrement()); // 自增+1然后返回加1之前的结果
        System.out.println(a.get()); // 获取到加1之后的结果
        System.out.println(a.addAndGet(2)); // 2+2=4
        System.out.println(a.compareAndSet(4, 3)); // 第一个参数是与a一样大小的，所以a的值被修改为3
        System.out.println(a.get()); // 获取到修改之后的结果。

    }
}
/*
1
2
4
true
3
*/
```

三，原子更新数组

- 通过原子的方式更新数组中的某个元素，Atomic包提供了以下类。

- AtomicIntegerArray：原子更新整型数组中的元素
- AtomicLongArray：原子更新长整型数组中的元素
- AtomicReferenceArray：原子更新引用类型数组中的元素

  以AtomicIntegerArray为例

```
import java.util.concurrent.atomic.AtomicIntegerArray;;

public class AtomicIntegetArrayTest {
    private static int[] value = new int[] { 1, 2 };
    private static AtomicIntegerArray a = new AtomicIntegerArray(value);
    public static void main(String[] srgs){
        a.getAndSet(0, 3);          //将第一个元素的值替换成3
        System.out.println(a.get(0));
        System.out.println(value[0]);   //原数组中的位置的值是不发生改变的
    }
}
/*
3
1
*/
```

四，原子类型更新引用类型

- 三个类
  - AtomicReference：原子更新引用类型
  - AtomicReferenceFieldUpdatater：原子更新引用类型的字段
  - AtomicMarkableReference：原子更新带有标记位的引用类型

  以AtomicReference类为例

```
import java.util.concurrent.*;
import java.util.concurrent.atomic.AtomicReference;

public class AtomicReferenceTest {
    private static AtomicReference<User> R = new AtomicReference<User>();

    public static void main(String[] args) {
        User user = new User("Tom", 16);
        R.set(user); // 将User对象传入R中,相当于将user赋值给了R
        User updata = new User("Jim", 17);
        System.out.println(R.compareAndSet(user, updata)); // R与第一个参数进行比
较，如果一样，则将第二个参数的值赋给R
        System.out.println(R.get().getName());
        System.out.println(R.get().getOld());
        System.out.println(user.getName() + "/" + user.getOld());  //并不会修改
原来的值
    }
}

class User {
```

```
    private String name;
    private int old;

    public User(String name, int old) {
        this.name = name;
        this.old = old;
    }

    public String getName() {
        return name;

    }

    public int getOld() {
        return old;
    }
}
/*
true
Jim
17
Tom/16
*/
```

五，原子更新字段类

- 用来更新某个类的某个字段
  - AtomicIntegerFieldUpdater：原子更新整型的字段的更新器
  - AtomicLongFieldUpdater：原子更新长整型字段的更新器
  - AtomicStampedReference：原子更新带有版本号的引用类型

  以AtomicIntegerFieldUpdater为例

```
import java.util.concurrent.*;
import java.util.concurrent.atomic.AtomicIntegerFieldUpdater;

public class AtomicIntegerFieldUpdaterTest {
    private static AtomicIntegerFieldUpdater<Student>
R=AtomicIntegerFieldUpdater.newUpdater(Student.class, "old");

    public static void main(String[] args) {
        Student a = new Student("Tom", 15);
        System.out.println(R.getAndIncrement(a));
        System.out.println(R.get(a));
    }
}

class Student {
    private String name;
    public volatile int old;    //字段必须是volatile类型的，在线程之间共享变量时保证立
即可见
```

```
    public Student(String name, int old) {
        this.name = name;
        this.old = old;
    }

    public String getName() {
        return name;
    }

    public int getOld() {
        return old;
    }

}
/*
15
16
*/
```