07 迭代器模式、单例模式

C-3 创建: 张林伟, 最后修改: 张林伟 2018-10-23 17:39

迭代器模式

提供一个方法顺序访问一个聚合对象中各个元素，而又不暴露该对象的内部表示。

Demo

迭代接口

代码块 Java

```java
public interface Iterator<E> {
    boolean hasNext();
    E next();
}
```

容器

代码块 Java

```java
// Container.java
public interface Container<E> {
    Iterator<E> getIterator();
}

// NameRepository.java
public class NameRepository implements Container<String> {
    private String[] names = {"lewis", "tom", "john"};
    @Override public Iterator<String> getIterator() {
        return new NameIterator();
    }
    private class NameIterator implements Iterator<String> {
        int index = 0;
        @Override public boolean hasNext() {
            return index < names.length;
        }
        @Override public String next() {
            if (this.hasNext()) {
                return names[index++];
            } else {
                throw new NoSuchElementException();
            }
        }
    }
}
```

客户端

代码块 Java

```java
public class Client {
    public static void main(String[] args) {
        NameRepository nameRepository = new NameRepository();
        Iterator iterator = nameRepository.getIterator();

        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }
    }
}
```

单例模式

保证一个类仅有一个实例，并提供一个访问它的全局访问点。

Demo

代码块 Java

```java
/**
 * Desc: 懒汉式，线程不安全
 * ------------------------------------
 * Author:zhanglinwei
 * Date:2018/10/23
 * Time:17:23
 */
public class Singleton1 {

    private static Singleton1 instance;

    private Singleton1 (){}

    public static Singleton1 getInstance() {
        if (instance == null) {
            instance = new Singleton1();
        }
        return instance;
    }
}
```

代码块 Java

```java
/**
 * Desc: 懒汉式，线程安全
 * ------------------------------------
 * Author:zhanglinwei
 * Date:2018/10/23
 * Time:17:24
 */
public class Singleton2 {

    private static Singleton2 instance;

    private Singleton2 (){}

    public static synchronized Singleton2 getInstance() {
        if (instance == null) {
            instance = new Singleton2();
        }
        return instance;
    }
}
```

代码块 Java

```java
/**
 * Desc: 饿汉式
 * ------------------------------------
 * Author:zhanglinwei
 * Date:2018/10/23
 * Time:17:25
 */
public class Singleton3 {

    private static Singleton3 instance = new Singleton3();

    private Singleton3 (){}

    public static Singleton3 getInstance() {
        return instance;
    }
}
```

代码块 Java

```java
/**
 * Desc: 双重检验锁
 * ------------------------------------
 * Author:zhanglinwei
 * Date:2018/10/23
 * Time:17:27
 */
public class Singleton4 {

    // 声明成 volatile
    private volatile static Singleton4 instance;

    private Singleton4 (){}

    public static Singleton4 getSingleton4() {
        if (instance == null) {
            synchronized (Singleton4.class) {
                if (instance == null) {
                    instance = new Singleton4();
                }
            }
        }
        return instance;
    }
}
```

代码块 Java

```java
/**
 * Desc: 静态内部类
 * ------------------------------------
 * Author:zhanglinwei
 * Date:2018/10/23
 * Time:17:31
 */
public class Singleton5 {

    private static class SingletonHolder {
        private static final Singleton5 INSTANCE = new Singleton5();
    }

    private Singleton5 (){}

    public static final Singleton5 getInstance() {
        return SingletonHolder.INSTANCE;
    }
}
```

一般用第三种（饿汉式）。