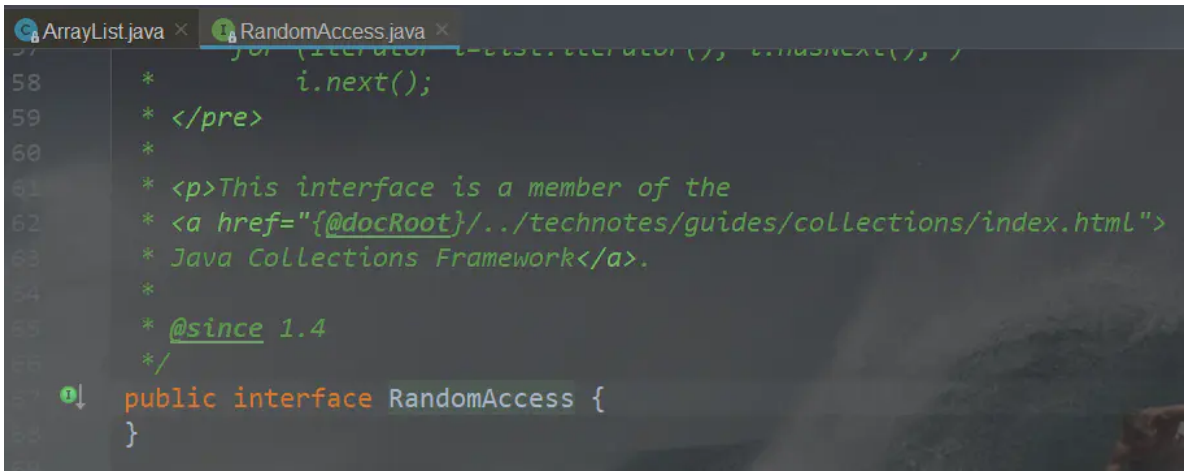
RandomAccess 这个空架子有何用?

在学习 `Java` 集合时， 最先学习的便是 `List` 中的 `ArrayList` 和 `LinkedList`， 学习集合很关键的是学习其源码， 了解底层实现方式， 那么今天就讲讲 `ArrayList` 实现的一个接口 `RandomAccess`。

好奇心的产生

查看 `ArrayList` 的源码， 发现它实现了 `RandomAccess` 这个接口， 出于好奇点进去看看， 结果发现这接口是空的， 这当然引发了更大的好奇心：这空架子到底有何用?



深入探究

JDK 官方文档是不可少的工具， 先看看它是怎么说的： `RandomAccess` 是 `List` 实现所使用的**标记接口**，用来表明其**支持快速（通常是固定时间）随机访问。**此接口的主要目的是允许一般的算法更改其行为，从而在将其应用到随机或连续访问列表时能提供良好的性能。

标记接口（Marker）：这就说明了 `RandomAccess` 为空的原因，这个接口的功能仅仅起到标记的作用。

这不是与序列化接口 `Serializable` 差不多吗？ 只要你认真观察， 其实不只这一个标记接口， 实际上 `ArrayList` 还实现了另外两个这样的空接口：

`Cloneable` 接口： 实现了 `Cloneable` 接口，以指示 `Object.clone()` 方法可以合法地对该类实例进行**按字段复制。** 如果在没有实现 `Cloneable` 接口的实例上调用 `Object` 的 `clone` 方法，则会导致抛出 `CloneNotSupportedException` 异常。

`Serializable` 接口： 类通过实现 `java.io.Serializable` 接口以**启用其序列化功能。**未实现此接口的类将无法使其任何状态序列化或反序列化。

继续探讨

标记接口都有什么作用呢？ 继续讨论 RandomAccess 的作用，其他两个在此不作讨论。

如果 `List` 子类实现了 `RandomAccess` 接口，那表示它能快速随机访问存储的元素， 这时候你想到的可能是数组， 通过下标 `index` 访问， 实现了该接口的 `ArrayList` 底层实现就是数组， 同样是通过下标访问， 只是我们需要用 `get()` 方法的形式， `ArrayList` 底层仍然是数组的访问形式。

同时你应该想到链表， `LinkedList` 底层实现是链表， `LinkedList` 没有实现 `RandomAccess` 接口，发现这一点就是突破问题的关键点。

数组支持随机访问， 查询速度快， 增删元素慢； 链表支持顺序访问， 查询速度慢， 增删元素快。所以对应的 `ArrayList` 查询速度快，`LinkedList` 查询速度慢， `RandomAccess` 这个标记接口就是标记能够随机访问元素的集合， 简单来说就是底层是数组实现的集合。

为了提升性能，在遍历集合前，我们便可以通过 `instanceof` 做判断， 选择合适的集合遍历方式，当数据量很大时， 就能大大提升性能。

随机访问列表使用循环遍历，顺序访问列表使用迭代器遍历。

先看看 `RandomAccess` 的使用方式

```
import java.util.*;
public class RandomAccessTest {
    public static void traverse(List list){

        if (list instanceof RandomAccess){
            System.out.println("实现了RandomAccess接口，不使用迭代器");

            for (int i = 0;i < list.size();i++){
                System.out.println(list.get(i));
            }

        }else{
            System.out.println("没实现RandomAccess接口，使用迭代器");

            Iterator it = list.iterator();
            while(it.hasNext()){
                System.out.println(it.next());
            }

        }
    }
    public static void main(String[] args) {
        List<String> arrayList = new ArrayList<>();
        arrayList.add("a");
        arrayList.add("b");
        traverse(arrayList);

        List<String> linkedList = new LinkedList<>();
        linkedList.add("c");
        linkedList.add("d");
        traverse(linkedList);
    }
}
复制代码
```

下面我们加入大量数据进行性能测试：

```
import java.util.*;
public class RandomAccessTimeTest {
```

```
    //使用for循环遍历
    public static long traverseByLoop(List list){
        long startTime = System.currentTimeMillis();
        for (int i = 0;i < list.size();i++){
            list.get(i);
        }
        long endTime = System.currentTimeMillis();
        return endTime-startTime;
    }

    //使用迭代器遍历
    public static long traverseByIterator(List list){
        Iterator iterator = list.iterator();
        long startTime = System.currentTimeMillis();
        while (iterator.hasNext()){
            iterator.next();
        }
        long endTime = System.currentTimeMillis();
        return endTime-startTime;
    }

    public static void main(String[] args) {
        //加入数据
        List<String> arrayList = new ArrayList<>();
        for (int i = 0;i < 30000;i++){
            arrayList.add("" + i);
        }
        long loopTime = RandomAccessTimeTest.traverseByLoop(arrayList);
        long iteratorTime = RandomAccessTimeTest.traverseByIterator(arrayList);
        System.out.println("ArrayList:");
        System.out.println("for循环遍历时间:" + loopTime);
        System.out.println("迭代器遍历时间:" + iteratorTime);

        List<String> linkedList = new LinkedList<>();
        //加入数据
        for (int i = 0;i < 30000;i++){
            linkedList.add("" + i);
        }
        loopTime = RandomAccessTimeTest.traverseByLoop(linkedList);
        iteratorTime = RandomAccessTimeTest.traverseByIterator(linkedList);
        System.out.println("LinkedList:");
        System.out.println("for循环遍历时间:" + loopTime);
        System.out.println("迭代器遍历时间:" + iteratorTime);
    }
}
复制代码
```

结果：

ArrayList: for 循环遍历时间: 3 迭代器遍历时间: 7

LinkedList: for 循环遍历时间: 2435 迭代器遍历时间: 3

结论

根据结果我们可以得出结论： `ArrayList` 使用 for 循环遍历优于迭代器遍历 `LinkedList` 使用 迭代器遍历优于 for 循环遍历

根据以上结论便可利用 `RandomAccess` 在遍历前进行判断，根据 `List` 的不同子类选择不同的遍历方式， 提升算法性能。

学习阅读源码， 发现底层实现的精妙之处， 改变自己的思维， 从每一个小细节提升代码的性能。