ArrayList

黄智辉

Agenda

- 线性表
- Java泛型
- ArrayList的实现和使用细节
- 迭代器模式
- Java中各种List类的性能对比

求时间复杂度

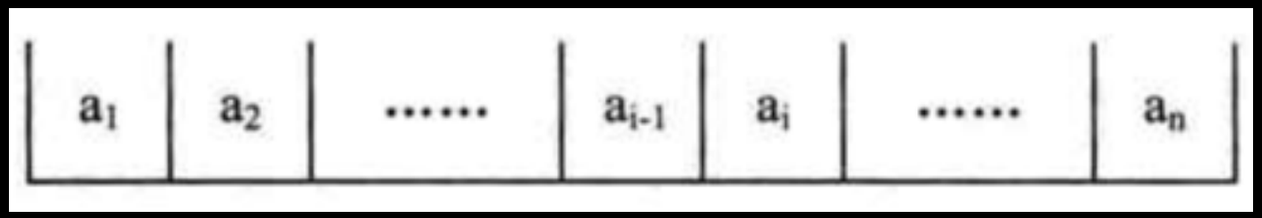
```
int count = 1;
while (count < n) {

    count  = count * 2;
}
```

```
int i, j;
for (i = 0; i < n; i ++) {

    for (j = i; j < n; j ++) {

        /**
         时间复杂度为O(1)的运算
         */
    }
}
```

第一部分：线性表

顺序存储结构

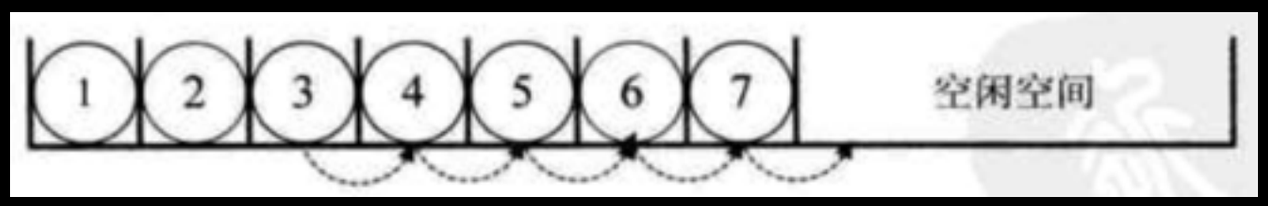
结构



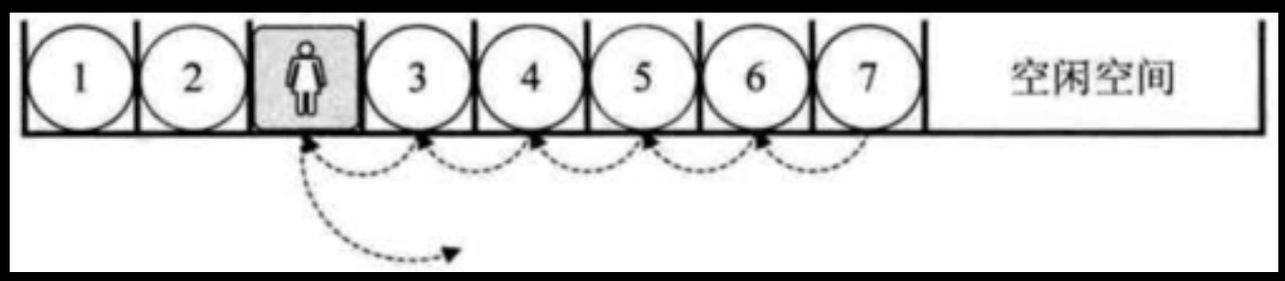
查询

$$LOC(a_i) = LOC(a_1) + (i-1) * c$$

插入

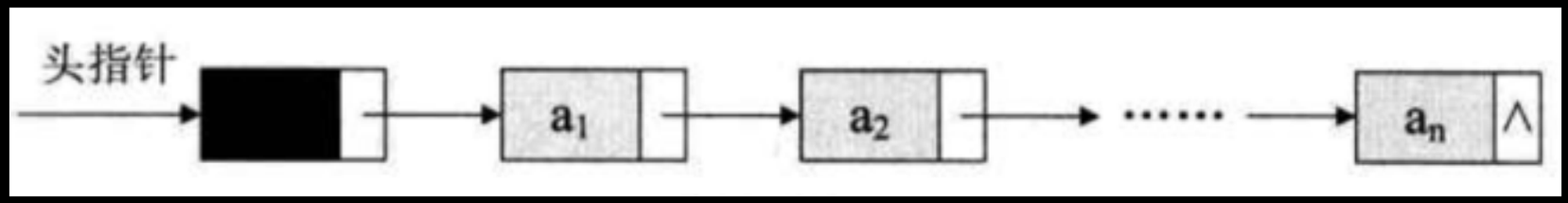


删除



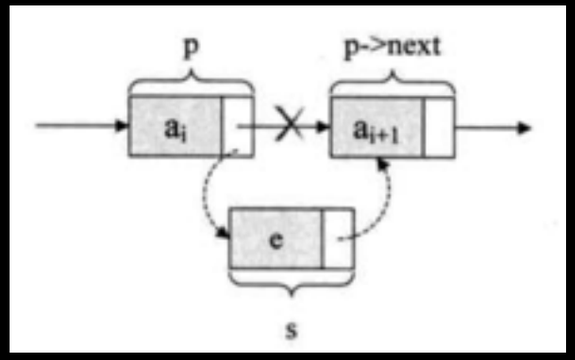
单链表

结构

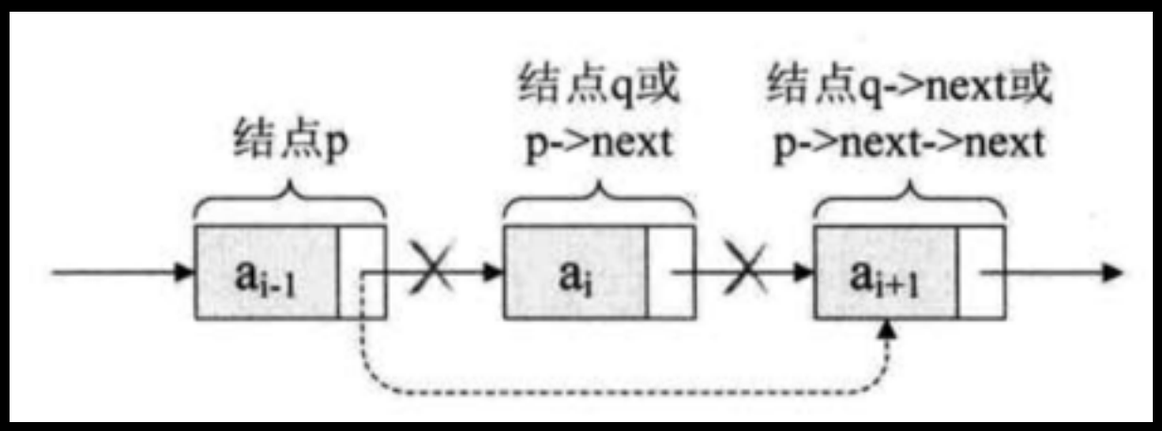


查询　O(n)

插入

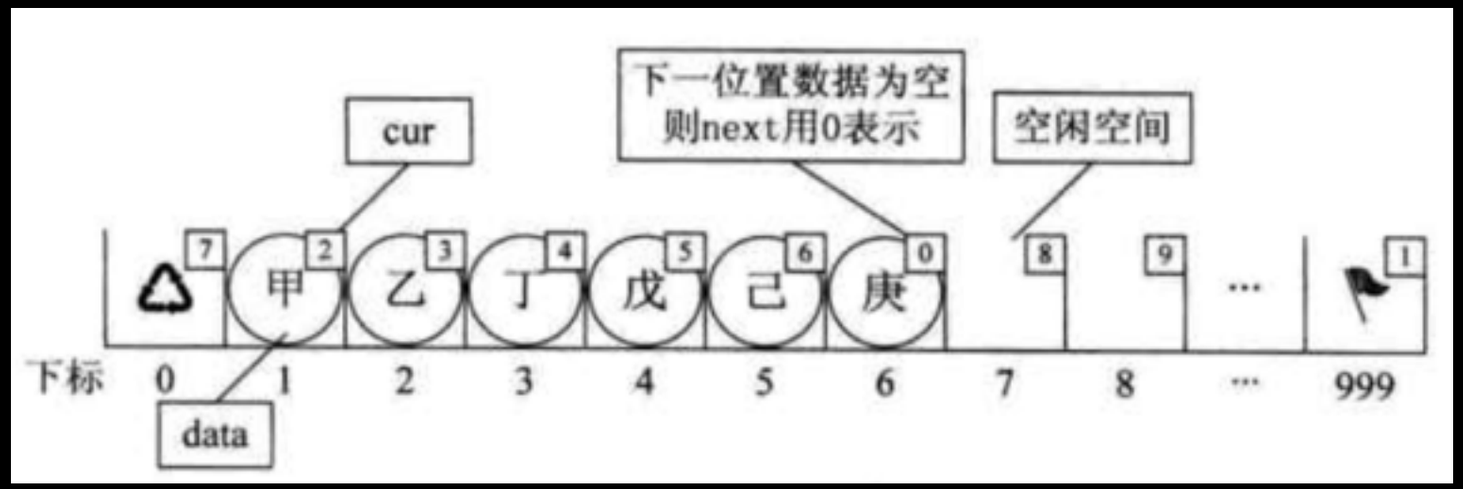


删除



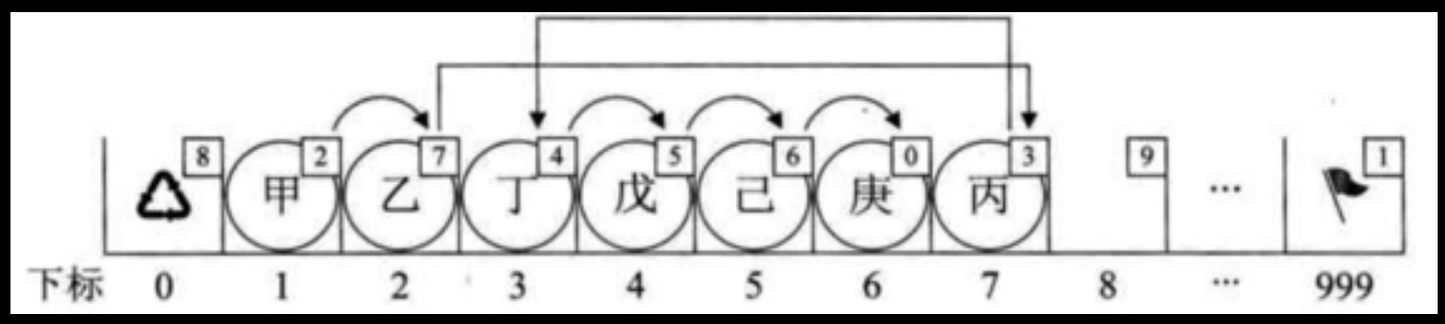
静态链表

结构

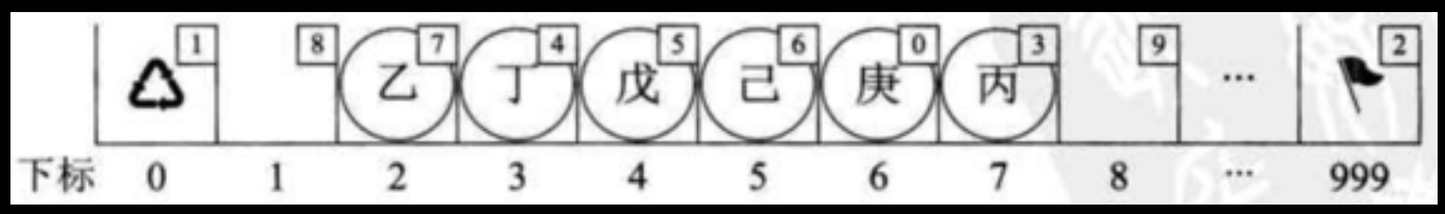


查询 O(n)

删除

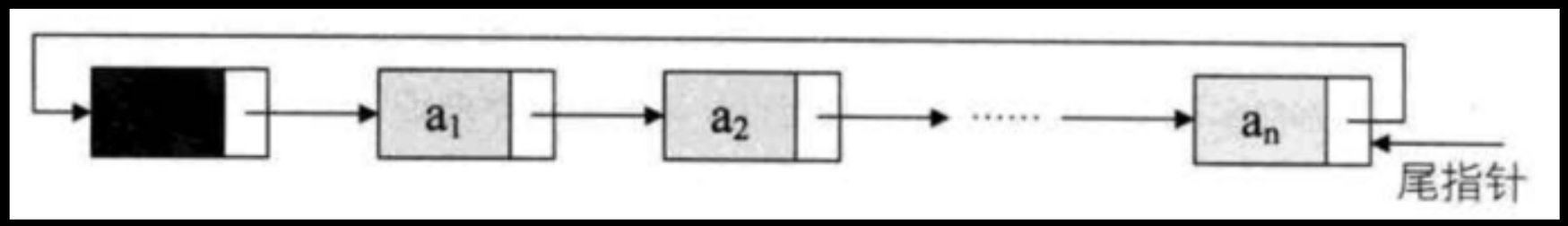


插入

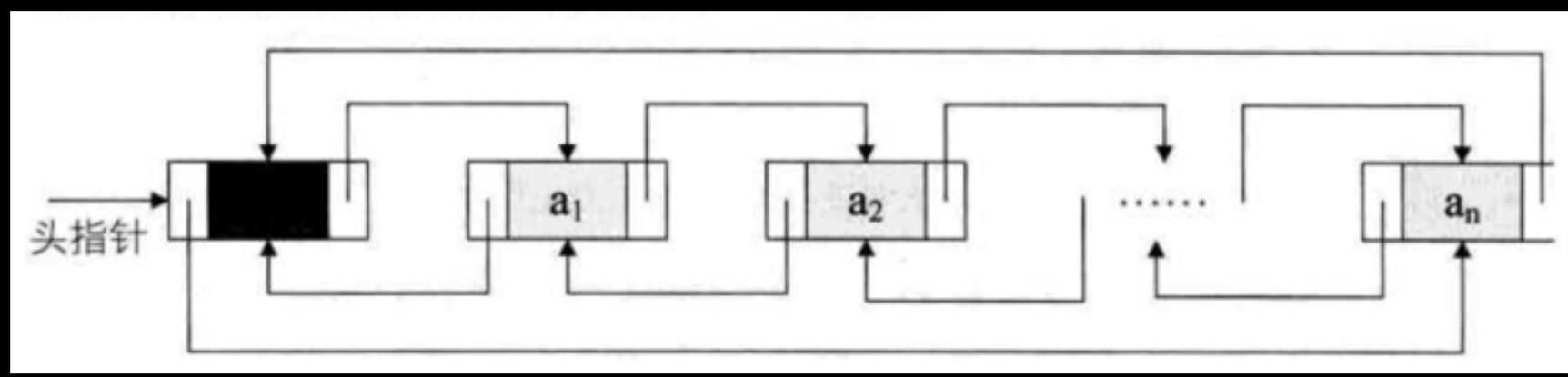


循环链表和双向链表

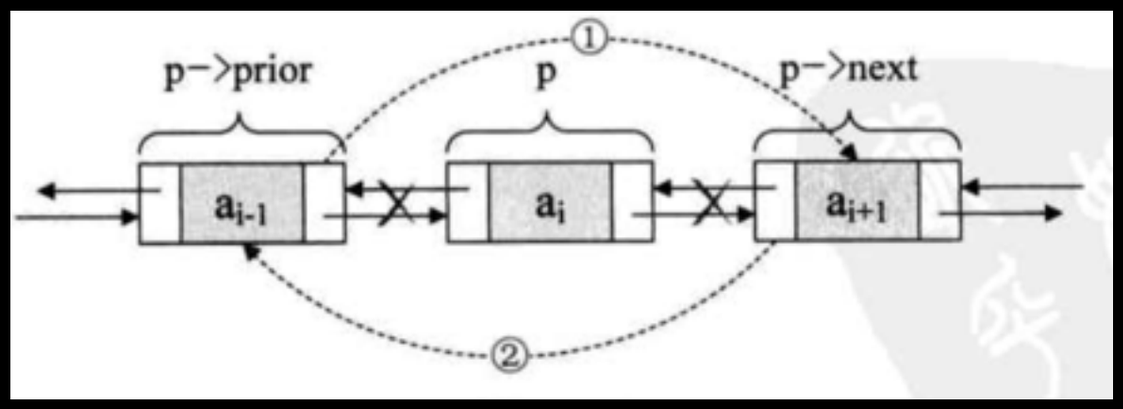
循环链表



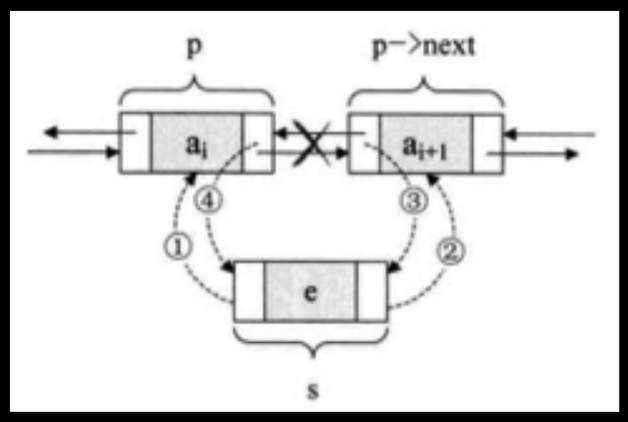
双向链表



删除



插入



栈？队列？

第二部分:Java中的泛型

1. 不能使用基本数据类型初始化泛型参数

2. Runtime类型检查不起作用

3.不能使用参数类型来创建Array

4. 可变参数警告

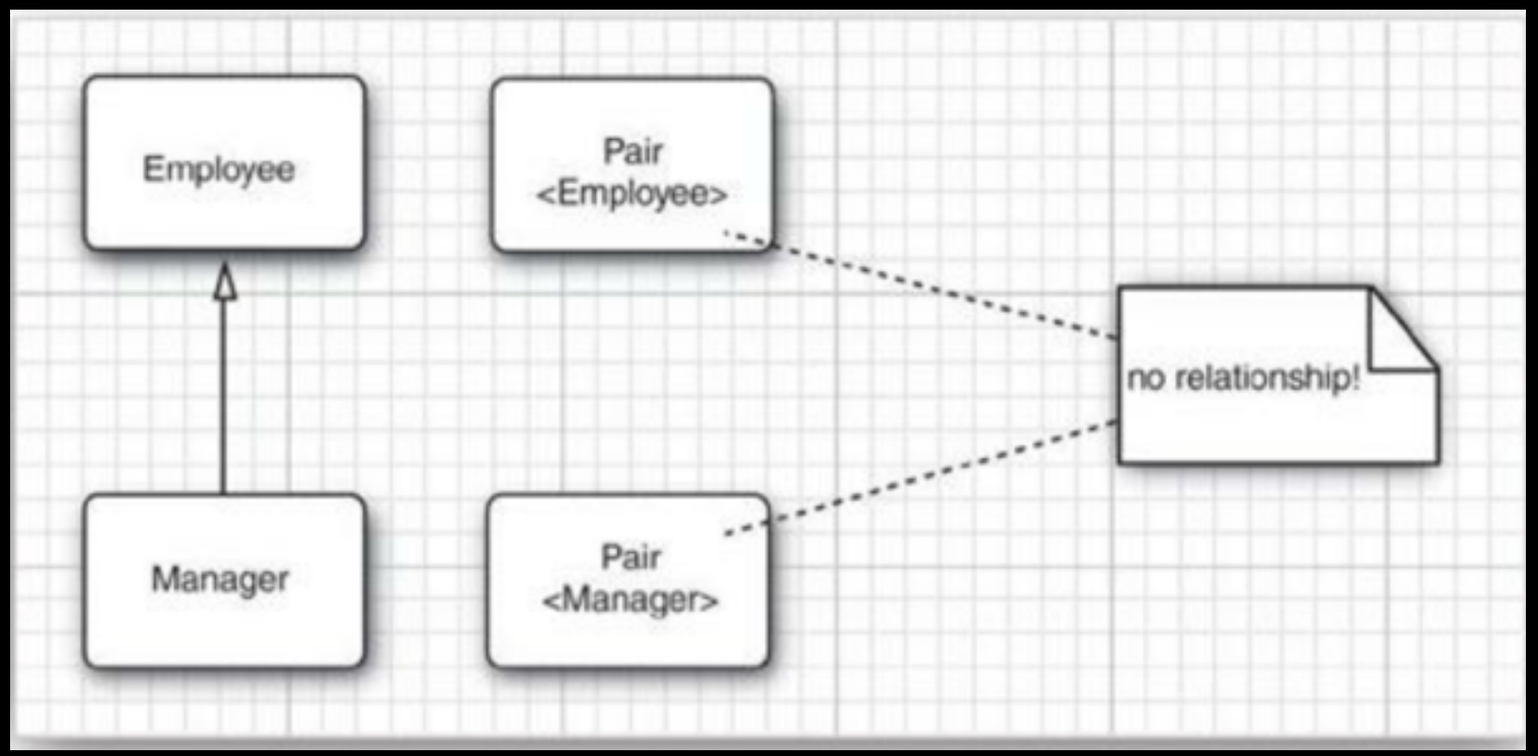
5.不能初始化类型变量

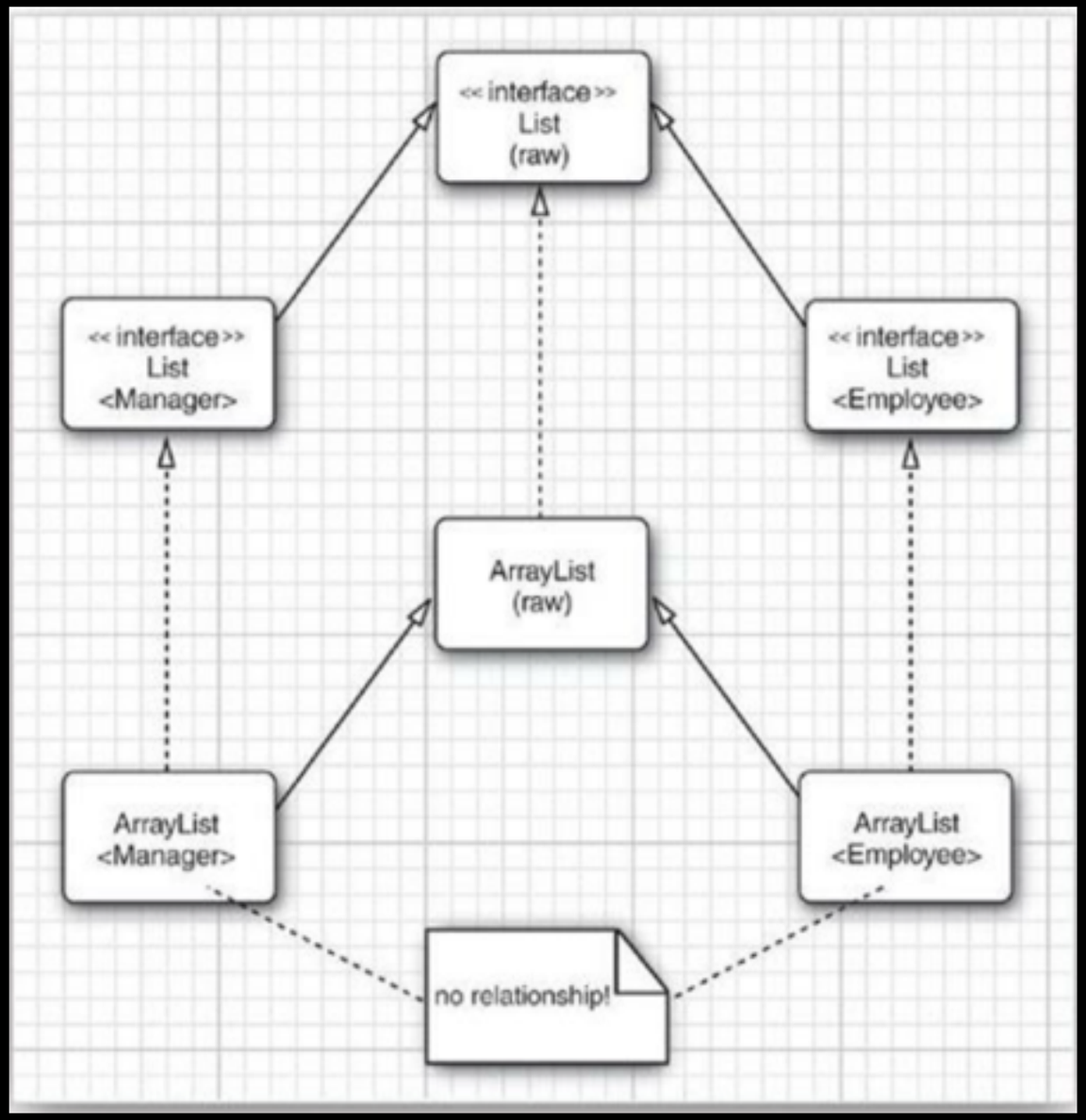
6.泛型类中的静态环境中不允许使用类型变量

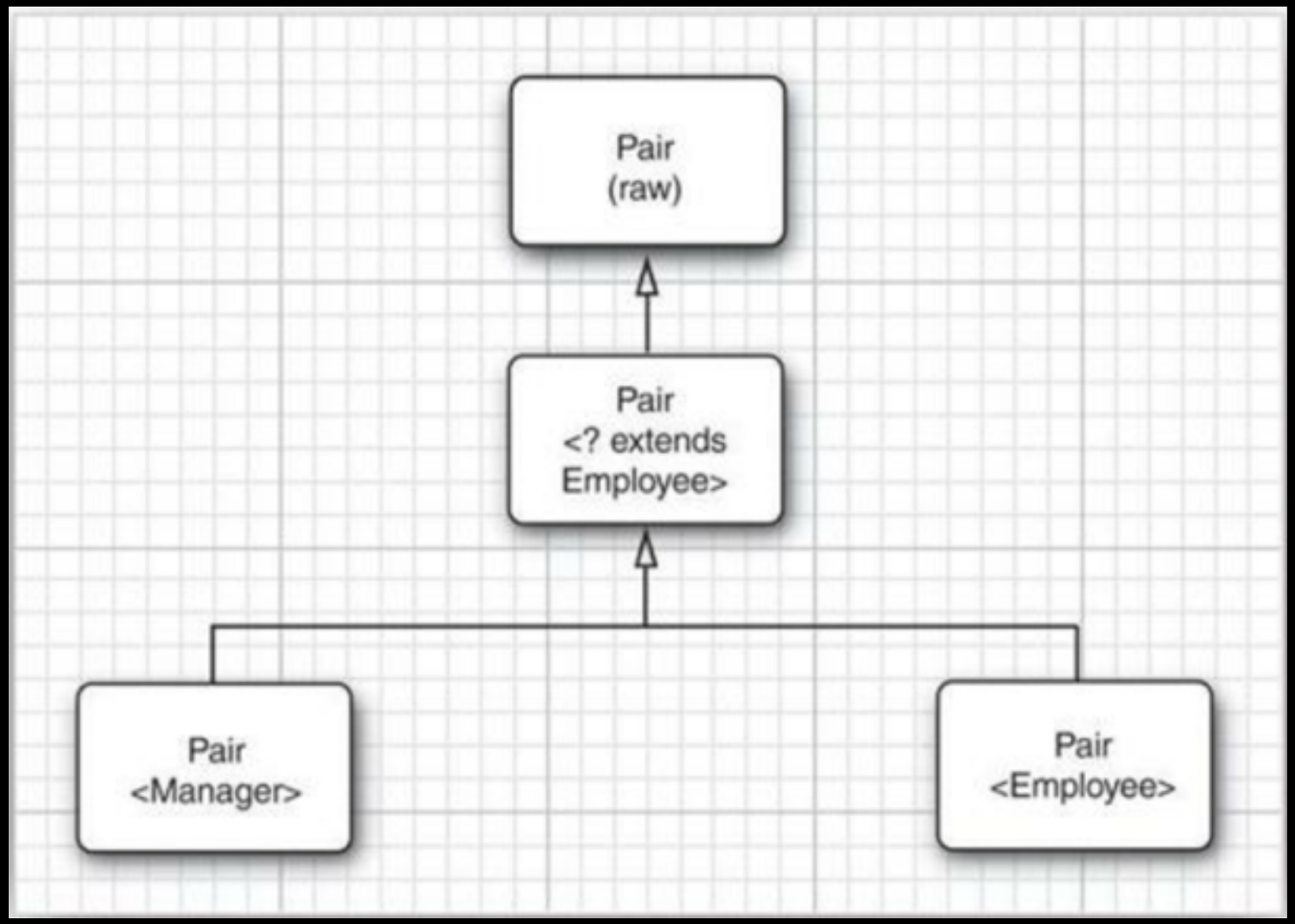
7.不能抛出或者捕获泛型类对象

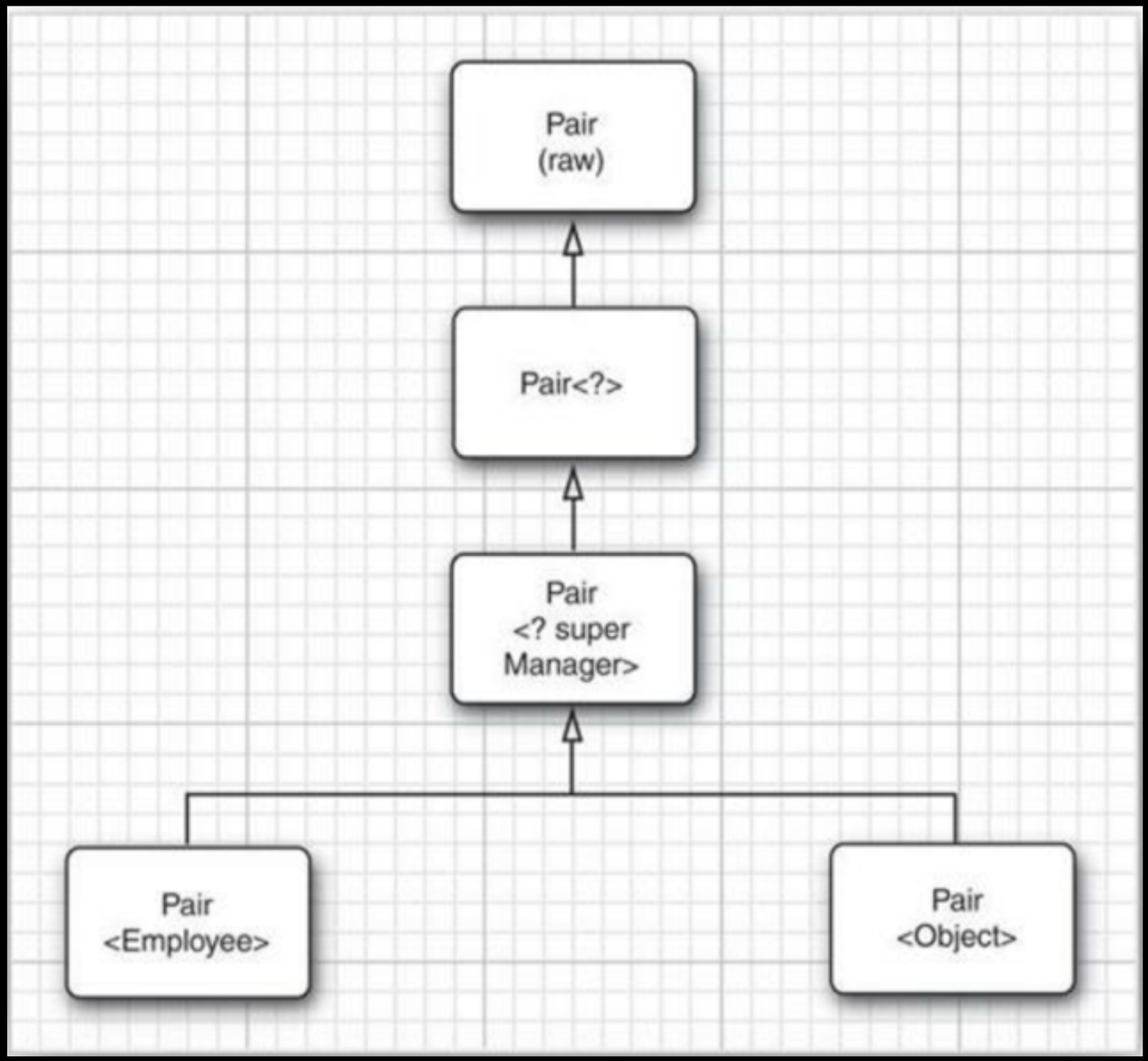
8.注意泛型擦除之后的冲突

泛型的继承规则


Employee
Pair
<Employee>
no relationship!
Manager
Pair
<Manager>

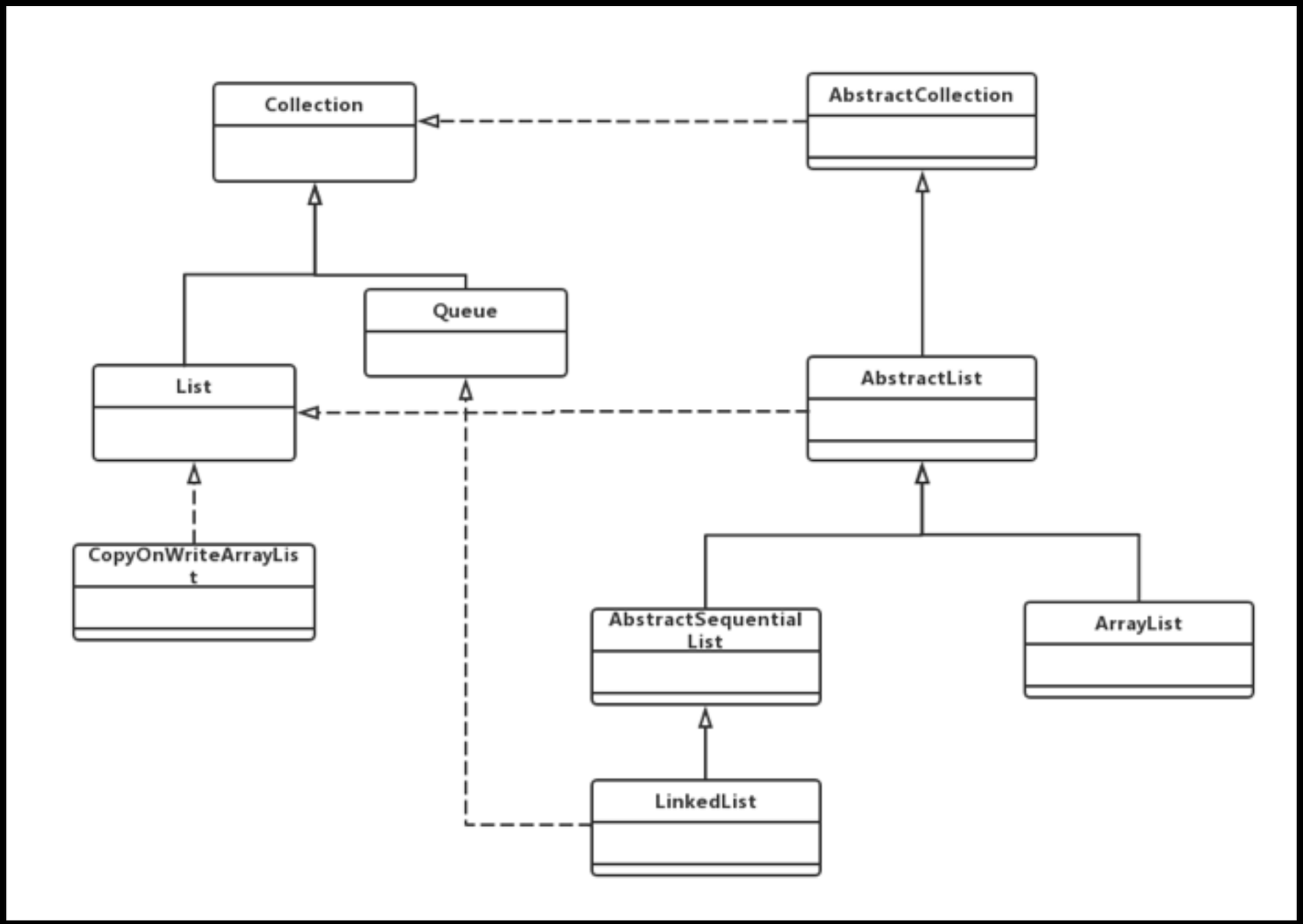

<< interface >>
List
(raw)
<< interface >>
List
<Manager>
<< interface >>
List
<Employee>
ArrayList
(raw)
ArrayList
<Manager>
ArrayList
<Employee>
no relationship!


Pair
(raw)
Pair
<? extends
Employee>
Pair
<Manager>
Pair
<Employee>


Pair
(raw)
Pair<?>
Pair
<? super
Manager>
Pair
<Employee>
Pair
<Object>

Coding Time

ArrayList类图



Fail-Fast的实现

```
private class Itr implements Iterator<E> {
    int cursor;       // index of next element to return
    int lastRet = -1; // index of last element returned; -1 if no such
    int expectedModCount = modCount;
```

```
    public void add(E e) {
        checkForComodification();

        try {
            int i = cursor;
            ArrayList.this.add(i, e);
            cursor = i + 1;
            lastRet = -1;
            expectedModCount = modCount;
        } catch (IndexOutOfBoundsException ex) {
            throw new ConcurrentModificationException();
        }
    }
```

```
    final void checkForComodification() {
        if (modCount != expectedModCount)
            throw new ConcurrentModificationException();
    }
```

ArrayCopy

```
 * @param      src      the source array.
 * @param      srcPos   starting position in the source array.
 * @param      dest     the destination array.
 * @param      destPos  starting position in the destination data.
 * @param      length   the number of array elements to be copied.
 * @exception  IndexOutOfBoundsException  if copying would cause
 *               access of data outside array bounds.
 * @exception  ArrayStoreException  if an element in the <code>src</code>
 *               array could not be stored into the <code>dest</code> array
 *               because of a type mismatch.
 * @exception  NullPointerException if either <code>src</code> or
 *               <code>dest</code> is <code>null</code>.
 */
public static native void arraycopy(Object src,  int  srcPos,
                                     Object dest, int destPos,
                                     int length);
```

扩容

老版本

```
public void ensureCapacity(int minCapacity) {
    modCount++;
    int oldCapacity = elementData.length;
    if (minCapacity > oldCapacity) {
        int newCapacity = (oldCapacity * 3)/2 + 1;
        if (newCapacity < minCapacity)
            newCapacity = minCapacity;
        // minCapacity is usually close to size, so this is a win:
        elementData = Arrays.copyOf(elementData, newCapacity);
    }

}
```

新版本

```
private void grow(int minCapacity) {
    // overflow-conscious code
    int oldCapacity = elementData.length;
    int newCapacity = oldCapacity + (oldCapacity >> 1);
    if (newCapacity - minCapacity < 0)
        newCapacity = minCapacity;
    if (newCapacity - MAX_ARRAY_SIZE > 0)
        newCapacity = hugeCapacity(minCapacity);
    // minCapacity is usually close to size, so this is a win:
    elementData = Arrays.copyOf(elementData, newCapacity);
}
```

默认是容量是10，如果要创建的List数量大于10，
那么在创建时候尽量指定其容量，或者手动调用grow方法来指定

ArrayList的注意细节

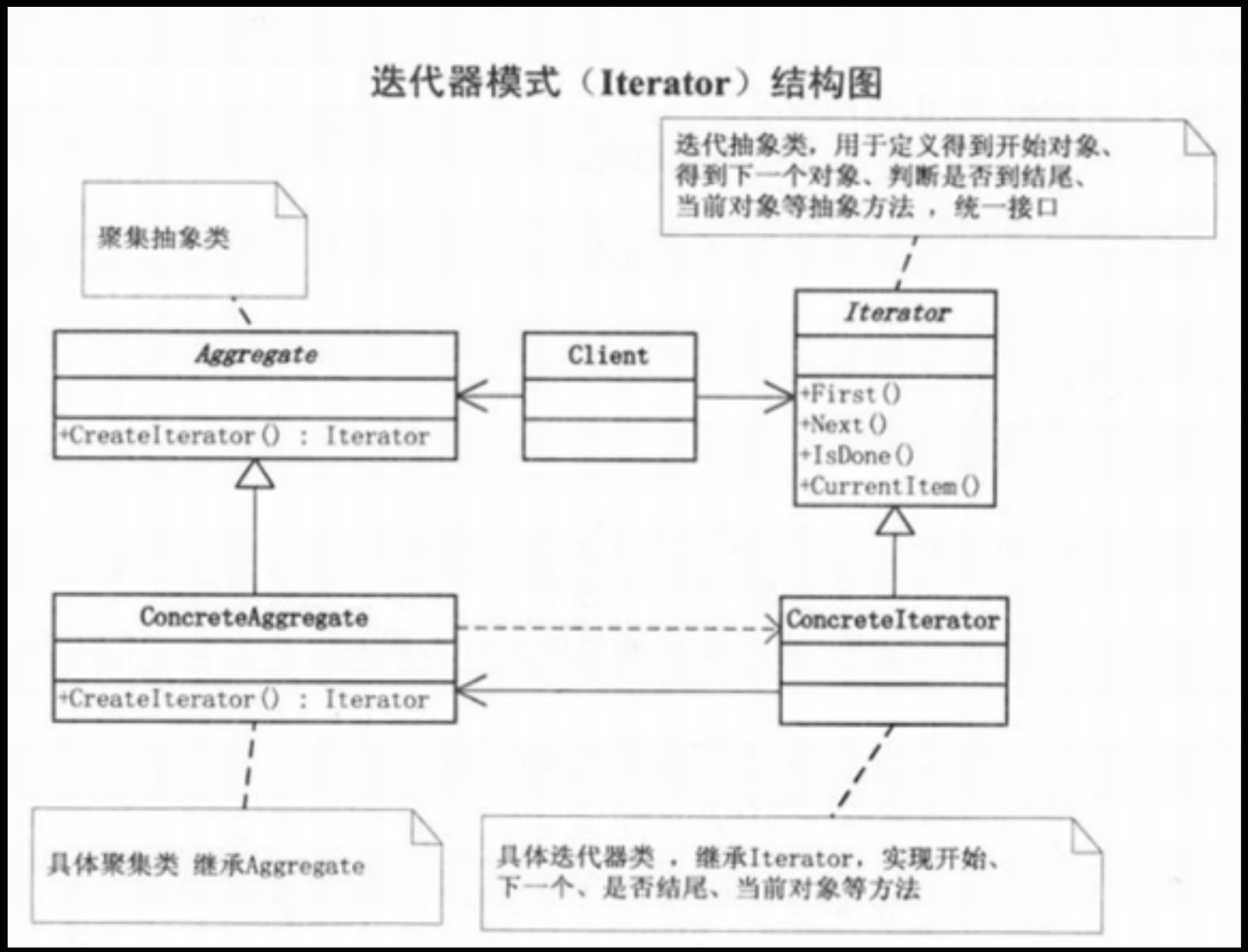
Coding Time

细节总结

1. 最好是使用Iterator的Remove方法，而不是Collection的Remove方法

2. 使用ArrayList，在数据插入完成之后要调用trimToSize以避免内存空间的浪费

迭代器模式


迭代器模式（Iterator）结构图
迭代抽象类，用于定义得到开始对象、
得到下一个对象、判断是否到结尾、
当前对象等抽象方法 ，统一接口
聚集抽象类
Aggregate
+CreateIterator() : Iterator
Client
Iterator
+First()
+Next()
+IsDone()
+CurrentItem()
ConcreteAggregate
+CreateIterator() : Iterator
ConcreteIterator
具体聚集类 继承Aggregate
具体迭代器类 ，继承Iterator，实现开始、
下一个、是否结尾、当前对象等方法

实战

Coding Time

几种List的性能对比

单位：ms

类型	Javas数组	ArrayList	LinkedList	Vector
get	16	23	63	31
iterate	31	47	33	48
insert	NO	1610	31	1625
remove	NO	6625	16	6750

Thx