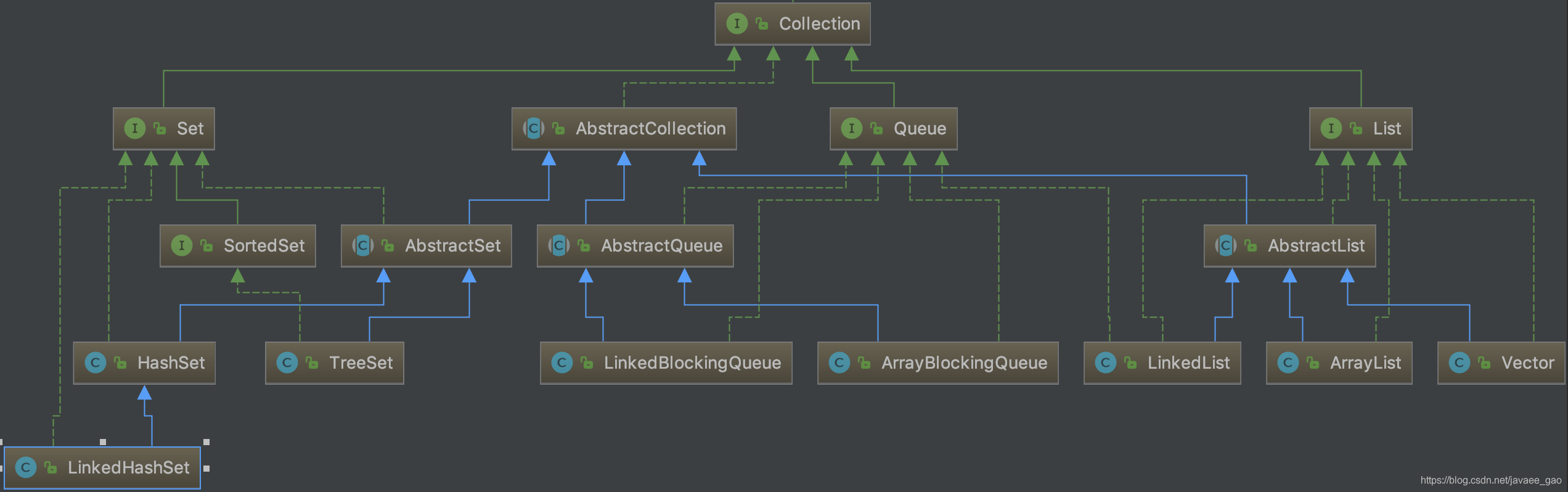
一、Java基础：

1、Collection：



List（有序存储）

Arraylist：基于数组实现，查询和遍历快，当插入的数据大于数组大小时，需要复制、移动，代价较高，删除和插入较慢

public class ArrayList<E> extends AbstractList<E>

implements List<E>, RandomAccess, Cloneable, java.io.Serializable

继承AbstractList，实现了list，是一个数组队列，提供了相关的增删改遍历，

RandomAccess接口标识着该类支持快速随机访问（只是一个定义了类型的接口，无作用），Cloneable即覆盖了函数clone（）,能被克隆，Serializable支持序列化，能通过序列化去传输，数组的默认大小是10

扩容方法：

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);

}

整个流程就是对旧数组位移运算得到新数组，然后把旧数组的整个数据复制到新数组上，操作代价很高，新容量大小为老容量的1.5倍

移位运算符简介：移位运算符就是在二进制的基础上对数字进行平移。按照平移的方向和填充数字的规则分为三种:<<(左移)、>>(带符号右移)和>>>(无符号右移)。作用：对于大数据的2进制运算,位移运算符比那些普通运算符的运算要快很多,因为程序仅仅移动一下而已,不去计算,这样提高了效率,节省了资源

添加和删除：

末尾添加：

public boolean add(E e) {

ensureCapacityInternal(size + 1); // Increments modCount!!

elementData[size++] = e;

return true;

}

指定位置添加（让数组自己复制自己实现让index开始之后的所有成员后移一个位置。）：

public void add(int index, E element) {

rangeCheckForAdd(index);

ensureCapacityInternal(size + 1); // Increments modCount!!

System.arraycopy(elementData, index, elementData, index + 1,

size - index);

elementData[index] = element; size++;}

删除：

public E remove(int index) {

rangeCheck(index);

modCount++;

E oldValue = elementData(index);

int numMoved = size - index - 1;

if (numMoved > 0)

System.arraycopy(elementData, index+1, elementData, index, numMoved);

elementData[--size] = null; // clear to let GC do its work

return oldValue;

}

流程就是把需要删除的元素右边的元素向左移动一位，覆盖了需要删除的元素，调用了arraycopy方法，操作代价高

Vector：基于数组实现，支持线程同步，可避免多线程引起的不一致性，但同步需要高花费，访问比arraylist慢

Linklist：用链表结构存储数据，适合动态的插入和删除，随机访问和遍历较慢，

LinkedList是由双向列表实现，使用Node存储节点信息，每个节点都有前节点（next），本节点（item），后节点（prev）。

Set（无序存储，数据唯一）

Hashset：基于哈希表实现，存入数据时按照哈希值排序，不是按照存入的顺序排序，为保证数据的唯一性，存入元素哈希值相同时，会使用equals方法比较，如果比较出不同就放入同一个哈希桶里

Treeset：基于红黑树实现，支持有序性操作，每增加一个对象都会进行排序，将对象插入到二叉树的指定位置

Linkhaskset（hashset+linkedhashmap）：继承于hashset，又是基于linkedhashmap来实现的，具有hashset的查找效率

Queue

1. Map

Hashmap：根据键的hashcode值来存储数据，大多数情况下可以直接定位到它的值，因而具有很快的访问速度，但是遍历顺序是不确定的，hashmap非线程安全，底层实现是数组+链表+红黑树

Concurrenthashmap：支持并发操作的hashmap，在jdk1.7和jdk1.8实现线程安全的方式不同

Hashtable：hashtable是遗留类，不建议使用，有很多常用的映射功能与hashmap类似，通过synchronized把整个（table）表锁住来实现线程安全，效率十分低下

Treemap：红黑树实现，可排序，需要对一个有序的key集合进行遍历时建议使用

Linkhashmap：是hashmap的一个子类，增加了一个双向链表，从而可以保存记录的插入顺序，在用iterator遍历linkhashmap时，先得到的记录肯定是先插入的，也可以在构造时带参数，按照访问次序排序