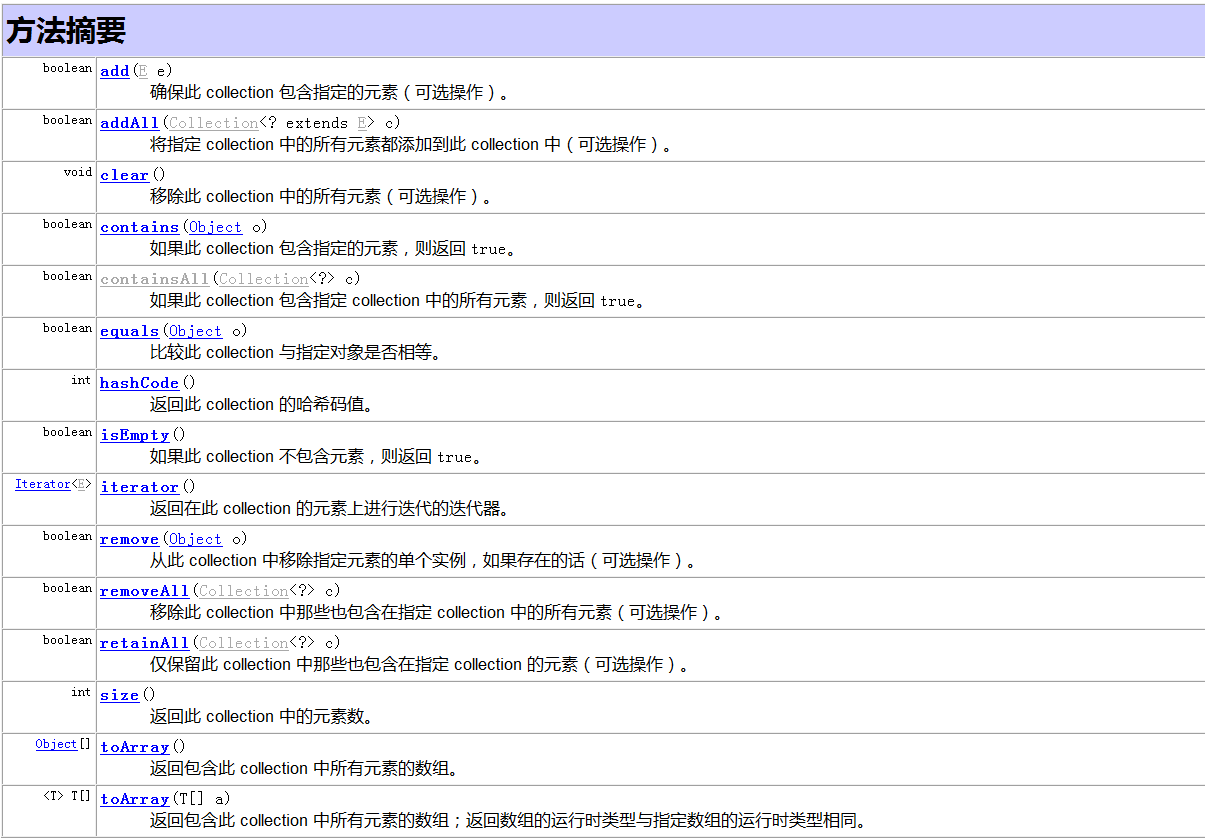
1. 集合框架

1 . 主要包括Collection接口和Map接口，其中Collection接口又包括List，Set，Queue三个子接口；Map又包含HashMap和HashTable两个子接口

1. 集合与数组的区别：（1）数组长度固定，只能保存定长的数据；集合可以保存长度不固 定的数据；（2）数组元素既可以是基本类型的值，也可以是对象类型；集合里只能放对象（实际是保存对象的引用变量）
2. Collection接口中的方法：



1. Set接口

Set包含HashSet，TreeSet和EnumSet三个重要的实现类。

Set的特点是无序的，元素不可重复的，元素可以为null。所谓无序，指的是Set中的元素在内存中存储的位置是无序的；

1. Set
2. HashMap底层实现原理

1. ArrayList底层实现原理

private static final Object[] *DEFAULTCAPACITY\_EMPTY\_ELEMENTDATA* = {};

private static final int *DEFAULT\_CAPACITY* = 10;

private static final int *MAX\_ARRAY\_SIZE* = Integer.*MAX\_VALUE* - 8;

public ArrayList() {  
 this.elementData = *DEFAULTCAPACITY\_EMPTY\_ELEMENTDATA*;  
}

public boolean add(E e) {  
 ensureCapacityInternal(size + 1); // Increments modCount!!  
 elementData[size++] = e;  
 return true;  
}

private void ensureCapacityInternal(int minCapacity) {  
 ensureExplicitCapacity(*calculateCapacity*(elementData, minCapacity));  
}

private void ensureExplicitCapacity(int minCapacity) {  
 modCount++;  
  
 // 如果添加元素后大于当前数组的长度，则进行扩容

if (minCapacity - elementData.length > 0)  
 grow(minCapacity);  
}

//扩容

private void grow(int minCapacity) {  
 // overflow-conscious code  
 int oldCapacity = elementData.length;

//将数组的长度增加原来数组的一半  
 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);  
}

1. HashMap底层实现原理

HashMap基于Map接口实现，元素以键值对的方式存储，并且允许使用null 建和null　值，　因为key不允许重复，因此只能有一个键为null,另外HashMap不能保证放入元素的顺序，它是无序的，和放入的顺序并不能相同。HashMap是线程不安全的。

HashMap采用Entry数组来存储key-value对，每一个键值对组成了一个Entry实体，Entry类实际上是一个单向的链表结构，它具有Next指针，可以连接下一个Entry实体，依次来解决Hash冲突的问题，因为HashMap是按照Key的hash值来计算Entry在HashMap中存储的位置的，如果hash值相同，而key内容不相等，那么就用链表来解决这种hash冲突。

//默认初始化的容量

static final int DEFAULT\_INITIAL\_CAPACITY = 1 << 4; // aka 16

//最大的容量

static final int MAXIMUM\_CAPACITY = 1 << 30; //2的30次方

//负载因子，当容量达到75%时就进行扩容操作

static final float DEFAULT\_LOAD\_FACTOR = 0.75f;

//阈值，用于判断是否需要扩容（threshold = 容量\*负载因子）

int threshold;

//无参构造方法，使用默认的容量大小和负载因子，并调用其他的构造方法

public HashMap() {

this(DEFAULT\_INITIAL\_CAPACITY, DEFAULT\_LOAD\_FACTOR);

}

//构造方法，指定初始容量和负载因子

public HashMap(int initialCapacity, float loadFactor) {

If (initialCapacity<0) {

throw new IllegalArgumentException("Illegal initial capacity: " +initialCapacity);

}

If (initialCapacity > MAXIMUM\_CAPACITY) {

initialCapacity = MAXIMUM\_CAPACITY;

}

If (loadFactor < 0 || Float.isNaN(loadFactor)) {

throw new IllegalArgumentException("Illegal load factor: " + loadFactor);

}

this.loadFactor = loadFactor; //设置负载因子

threshold = initialCapacity; //初始容量

init(); //不做任何操作

}