static <T> Collection<T>

synchronizedCollection(Collection<T> c)

返回指定 collection 支持的同步（线程安全的）collection。

static <T> List<T>

synchronizedList(List<T> list)

返回指定列表支持的同步（线程安全的）列表。

static <K,V> Map<K,V>

synchronizedMap(Map<K,V> m)

返回由指定映射支持的同步（线程安全的）映射。

static <T> Set<T>

synchronizedSet(Set<T> s)

返回指定 set 支持的同步（线程安全的）set。

static <K,V> SortedMap<K,V>

synchronizedSortedMap(SortedMap<K,V> m)

返回指定有序映射支持的同步（线程安全的）有序映射。

static <T> SortedSet<T>

synchronizedSortedSet(SortedSet<T> s)

返回指定有序 set 支持的同步（线程安全的）有序 set。

synchronizedCollection

public static <T> Collection<T> synchronizedCollection(Collection<T> c)返回指定 collection 支持的同步（线程安全的）collection。为了保证按顺序访问，必须通过返回的 collection 完成所有对底层实现 collection 的访问。

在返回的 collection 上进行迭代时，用户必须手工在返回的 collection 上进行同步：

Collection c = Collections.synchronizedCollection(myCollection);

...

synchronized(c) {

Iterator i = c.iterator(); // Must be in the synchronized block

while (i.hasNext())

foo(i.next());

}

不遵从此建议将导致无法确定的行为。

返回的 collection 不会 将 hashCode 和 equals 操作传递给底层实现 collection，但这依赖于 Object 的 equals 和 hashCode 方法。在底层实现 collection 是一个 set 或一个列表的情况下，有必要遵守这些操作的协定。

如果指定 collection 是可序列化的，则返回的 collection 也将是可序列化的。

参数：

c - 被“包装”在同步 collection 中的 collection。

返回：

指定 collection 的同步视图。

--------------------------------------------------------------------------------

synchronizedSet

public static <T> Set<T> synchronizedSet(Set<T> s)返回指定 set 支持的同步（线程安全的）set。为了保证按顺序访问，必须通过返回的 set 完成对所有底层实现 set 的访问。

在返回的 set 上进行迭代时，用户必须手工在返回的 set 上进行同步：

Set s = Collections.synchronizedSet(new HashSet());

...

synchronized(s) {

Iterator i = s.iterator(); // Must be in the synchronized block

while (i.hasNext())

foo(i.next());

}

不遵从此建议将导致无法确定的行为。

如果指定 set 是可序列化的，则返回的 set 也将是可序列化的。

参数：

s - 被“包装”在同步 set 中的 set。

返回：

指定 set 的同步视图。

--------------------------------------------------------------------------------

synchronizedSortedSet

public static <T> SortedSet<T> synchronizedSortedSet(SortedSet<T> s)返回指定有序 set 支持的同步（线程安全的）有序 set。为了保证按顺序访问，必须通过返回的有序 set（或其视图）完成对所有底层实现有序 set 的访问。

在返回的有序 set 上或者其任意 subSet、headSet 或 tailSet 视图上进行迭代时，用户必须手工在返回的有序 set 上进行同步。

SortedSet s = Collections.synchronizedSortedSet(new TreeSet());

...

synchronized(s) {

Iterator i = s.iterator(); // Must be in the synchronized block

while (i.hasNext())

foo(i.next());

}

或者：

SortedSet s = Collections.synchronizedSortedSet(new TreeSet());

SortedSet s2 = s.headSet(foo);

...

synchronized(s) { // Note: s, not s2!!!

Iterator i = s2.iterator(); // Must be in the synchronized block

while (i.hasNext())

foo(i.next());

}

不遵从此建议将导致无法确定的行为。

如果指定的有序 set 是可序列化的，则返回的有序 set 也将是可序列化的。

参数：

s - 被“包装”在同步有序 set 中的有序 set。

返回：

指定有序 set 的同步视图。

--------------------------------------------------------------------------------

synchronizedList

public static <T> List<T> synchronizedList(List<T> list)返回指定列表支持的同步（线程安全的）列表。为了保证按顺序访问，必须通过返回的列表完成所有对底层实现列表的访问。

在返回的列表上进行迭代时，用户必须手工在返回的列表上进行同步：

List list = Collections.synchronizedList(new ArrayList());

...

synchronized(list) {

Iterator i = list.iterator(); // Must be in synchronized block

while (i.hasNext())

foo(i.next());

}

不遵从此建议将导致无法确定的行为。

如果指定列表是可序列化的，则返回的列表也将是可序列化的。

参数：

list - 被“包装”在同步列表中的列表。

返回：

指定列表的同步视图。

--------------------------------------------------------------------------------

synchronizedMap

public static <K,V> Map<K,V> synchronizedMap(Map<K,V> m)返回由指定映射支持的同步（线程安全的）映射。为了保证按顺序访问，必须通过返回的映射完成所有对底层实现映射的访问。

在返回映射的任意 collection 视图上进行迭代时，用户必须手工在返回的映射上进行同步：

Map m = Collections.synchronizedMap(new HashMap());

...

Set s = m.keySet(); // Needn't be in synchronized block

...

synchronized(m) { // Synchronizing on m, not s!

Iterator i = s.iterator(); // Must be in synchronized block

while (i.hasNext())

foo(i.next());

}

不遵从此建议将导致无法确定的行为。

如果指定映射是可序列化的，则返回的映射也将是可序列化的。

参数：

m - 被“包装”在同步映射中的映射。

返回：

指定映射的同步视图。

--------------------------------------------------------------------------------

synchronizedSortedMap

public static <K,V> SortedMap<K,V> synchronizedSortedMap(SortedMap<K,V> m)返回指定有序映射支持的同步（线程安全的）有序映射。为了保证按顺序访问，必须通过返回的有序映射（或其视图）完成对所有底层有序映射的访问。

当在返回的有序映射的 collection 视图或者其任何 subMap、headMap 或 tailMap 视图进行迭代时，用户必须手工在该映射上进行同步：

SortedMap m = Collections.synchronizedSortedMap(new TreeMap());

...

Set s = m.keySet(); // Needn't be in synchronized block

...

synchronized(m) { // Synchronizing on m, not s!

Iterator i = s.iterator(); // Must be in synchronized block

while (i.hasNext())

foo(i.next());

}

或者：

SortedMap m = Collections.synchronizedSortedMap(new TreeMap());

SortedMap m2 = m.subMap(foo, bar);

...

Set s2 = m2.keySet(); // Needn't be in synchronized block

...

synchronized(m) { // Synchronizing on m, not m2 or s2!

Iterator i = s.iterator(); // Must be in synchronized block

while (i.hasNext())

foo(i.next());

}

不遵从此建议将导致无法确定的行为。

如果指定的有序映射是可序列化的，则返回的有序映射也将是可序列化的。

参数：

m - 被“包装”在同步有序映射中的有序映射。

返回：

指定有序映射的同步视图。

--------------------------------------------------------------------------------

checkedCollection

public static <E> Collection<E> checkedCollection(Collection<E> c,

Class<E> type)返回指定 collection 的一个动态类型安全视图。试图插入一个错误类型的元素将导致立即抛出 ClassCastException。假设在生成动态类型安全视图之前，collection 不包含任何类型不正确的元素，并且所有对该 collection 的后续访问都通过该视图进行，则可以保证 该 collection 不包含类型不正确的元素。

一般的编程语言机制中都提供了编译时（静态）类型检查，但是一些未经检查的强制转换可能会使此机制无效。通常这不是一个问题，因为编译器会在所有这类未经检查的操作上发出警告。但有的时候，只进行单独的静态类型检查并不够。例如，假设将 collection 传递给一个第三方库，则库代码不能通过插入一个错误类型的元素来毁坏 collection。

动态类型安全视图的另一个用途是调试。假设某个程序运行失败并抛出 ClassCastException，这指示一个类型不正确的元素被放入已参数化 collection 中。不幸的是，该异常可以发生在插入错误元素之后的任何时间，因此，这通常只能提供很少或无法提供任何关于问题真正来源的信息。如果问题是可再现的，那么可以暂时修改程序，使用一个动态类型安全视图来包装该 collection，通过这种方式可快速确定问题的来源。例如，以下声明：

Collection<String> c = new HashSet<String>();

可以暂时用下面的声明代替：

Collection<String> c = Collections.checkedCollection(

new HashSet<String>(), String.class);

再次运行程序会造成它在将类型不正确的元素插入 collection 的地方失败，从而清楚地识别问题的来源。问题得以解决后，可以将修改后的声明转换回原来的声明。

返回的 collection 不会 将 hashCode 和 equals 操作传递给底层实现 collection，但这依赖于 Object 的 equals 和 hashCode 方法。在底层实现 collection 是一个 set 或一个列表的情况下，有必要遵守这些操作的协定。

如果指定 collection 是可序列化的，则返回的 collection 也将是可序列化的。

参数：

c - 将返回其动态类型安全视图的 collection

type - 允许 c 持有的元素类型

返回：

指定 collection 的动态安全类型视图

从以下版本开始：

1.5