- 解决线程安全问题
  - 1. 只知道使用并发工具，但并不清楚当前线程的来龙去脉，解决多线程问题却不了解线程。
    - 使用 ThreadLocal 来缓存数据，以为 ThreadLocal 在线程之间做了隔离不会有线程安全问题，没想到线程重用导致数据串了。请务必记得，在业务逻辑结束之前清理 ThreadLocal 中的数据。
    - 不能认为没有显式开启多线程就不会有线程安全问题。
    - 使用类似 ThreadLocal 工具来存放一些数据时，需要特别注意在代码运行完后，显式地去清空设置的数据。
  - 2. 误以为使用了并发工具就可以解决一切线程安全问题，期望通过把线程不安全的类替换为线程安全的类来一键解决问题。
    - 使用了 ConcurrentHashMap 就可以解决线程安全问题，没对复合逻辑加锁导致业务逻辑错误。如果你希望在一整段业务逻辑中，对容器的操作都保持整体一致性的话，需要加锁处理。
    - ConcurrentHashMap 对外提供的方法或能力的限制：
      - 多个操作之间不保证状态一致性，必要的时候需要手动加索
      - 诸如 size、isEmpty 和 containsValue 等聚合方法，在并发情况下可能会反映 ConcurrentHashMap 的中间状态。这些方法的返回值只能用作参考，而不能用于流程控制。显然，利用 size 方法计算差异值，是一个流程控制。
      - 诸如 putAll 这样的聚合方法也不能确保原子性，在 putAll 的过程中去获取数据可能会获取到部分数据。
  - 3. 没有充分了解并发工具的特性，还是按照老方式使用新工具导致无法发挥其性能。
    - 使用了 ConcurrentHashMap，但没有充分利用其提供的基于 CAS 安全的方法，还是使用锁的方式来实现逻辑。你可以阅读一下 ConcurrentHashMap 的文档，看一下相关原子性操作 API 是否可以满足业务需求，如果可以则优先考虑使用。

```
static final <K,V> boolean casTabAt(Node<K,V>[] tab, int i,
                                    Node<K,V> c, Node<K,V> v) {
    return U.compareAndSetObject(tab, ((long)i << ASHIFT) + ABASE, c, v);
}
```

    - ConcurrentHashMap 这样的高级并发工具的确提供了一些高级 API，只有充分了解其特性才能最大化其威力，而不能因为其足够高级、酷炫盲目使用。比如原子性方法 computeIfAbsent 等方法来做复合逻辑操作
  - 4. 没有了解清楚工具的适用场景，在不合适的场景下使用了错误的工具导致性能更差。
    - 没有理解 CopyOnWriteArrayList 的适用场景，把它用在了读写均衡或者大量写操作的场景下，导致性能问题。对于这种场景，你可以考虑是用普通的 List。

```
/**
 * Appends the specified element to the end of this list.
 *
 * @param e element to be appended to this list
 * @return {@code true} (as specified by {@link Collection#add})
 */
public boolean add(E e) {
    synchronized (lock) {
        Object[] elements = getArray();
        int len = elements.length;
        Object[] newElements = Arrays.copyOf(elements, len + 1);
        newElements[len] = e;
        setArray(newElements);
        return true;
    }
}
```

  - 思考
    - 是否可以把 ThreadLocalRandom 的实例设置到静态变量中，在多线程情况下重用呢?
      - 不可以
      - 基本原理：基本原理是，current()的时候初始化一个初始化种子到线程，每次nextseed再使用之前的种子生成新的种子：

```
UNSAFE.putLong(t = Thread.currentThread(), SEED,
               r = UNSAFE.getLong(t, SEED) + GAMMA);
```

      - 如果通过主线程调用一次current生成一个ThreadLocalRandom的实例保存起来，那么其它线程来获取种子的时候必然取不到初始种子，必须是每一个线程自己用的时候初始化一个种子到线程，可以在nextSeed设置一个断点看看：UNSAFE.getLong(Thread.currentThread(),SEED);
    - ConcurrentHashMap computeIfAbsent 和 putIfAbsent 方法的区别?
      - 1、当Key存在的时候，如果Value获取比较昂贵的话，putIfAbsent就白白浪费时间在获取这个昂贵的Value上（这个点特别注意）
      - 2、Key不存在的时候，putIfAbsent返回null，小心空指针，而computeIfAbsent返回计算后的值
      - 3、putIfAbsent可以存入null，computeIfAbsent计算结果是null只会返回null，不会写入。