TreeMap	不允许为 null	允许为 null	AbstractMap	线程不安全
ConcurrentHashMap	不允许为 null	不允许为 null	AbstractMap	锁分段技术（JDK8:CAS）
HashMap	允许为 null	允许为 null	AbstractMap	线程不安全

反例：由于 HashMap 的干扰，很多人认为 ConcurrentHashMap 是可以置入 null 值，而事实上，存储 null 值时会抛出 NPE 异常。

20. 【参考】合理利用好集合的有序性（sort）和稳定性（order），避免集合的无序性（unsort）和不稳定性（unorder）带来的负面影响。
说明：有序性是指遍历的结果是按某种比较规则依次排列的，稳定性指集合每次遍历的元素次序是一定的。如：ArrayList 是 order / unsort；HashMap 是 unorder / unsort；TreeSet 是 order / sort。

21. 【参考】利用 Set 元素唯一的特性，可以快速对一个集合进行去重操作，避免使用 List 的 contains() 进行遍历去重或者判断包含操作。

(七) 并发处理

1. 【强制】获取单例对象需要保证线程安全，其中的方法也要保证线程安全。
说明：资源驱动类、工具类、单例工厂类都需要注意。

2. 【强制】创建线程或线程池时请指定有意义的线程名称，方便出错时回溯。
正例：自定义线程工厂，并且根据外部特征进行分组，比如，来自同一机房的调用，把机房编号赋值给 whatFeatureOfGroup：

```
public class UserThreadFactory implements ThreadFactory {
    private final String namePrefix;
    private final AtomicInteger nextId = new AtomicInteger(1);
    // 定义线程组名称，在利用 jstack 来排查问题时，非常有帮助
    UserThreadFactory(String whatFeatureOfGroup) {
        namePrefix = "FromUserThreadFactory's" + whatFeatureOfGroup + "-Worker-";
    }
    @Override
    public Thread newThread(Runnable task) {
        String name = namePrefix + nextId.getAndIncrement();
        Thread thread = new Thread(null, task, name, 0, false);
        System.out.println(thread.getName());
        return thread;
    }
}
```

3. 【强制】线程资源必须通过线程池提供，不允许在应用中自行显式创建线程。
说明：线程池的好处是减少在创建和销毁线程上所消耗的时间以及系统资源的开销，解决资源不足的问题。如果不使用线程池，有可能造成系统创建大量同类线程而导致消耗完内存或者“过度切换”的问题。

4. 【强制】线程池不允许使用 Executors 去创建，而是通过 ThreadPoolExecutor 的方式，这样的处理方式让写的同学更加明确线程池的运行规则，规避资源耗尽的风险。
说明：Executors 返回的线程池对象的弊端如下：
1）FixedThreadPool 和 SingleThreadPool：
允许的请求队列长度为 Integer.MAX_VALUE，可能会堆积大量的请求，从而导致 OOM。
2）CachedThreadPool：
允许的创建线程数量为 Integer.MAX_VALUE，可能会创建大量的线程，从而导致 OOM。
3）ScheduledThreadPool：

六、工程结构

(一) 应用分层

1. 【推荐】根据业务架构实践，结合业界分层规范与流行技术框架分析，推荐分层结构如图所示，默认上层依赖于下层，箭头关系表示可直接依赖，如：开放 API 层可以依赖于 Web 层（Controller 层），也可以直接依赖于 Service 层，依此类推：

- 开放 API 层：可直接封装 Service 接口暴露成 RPC 接口；通过 Web 封装成 http 接口；网关控制层等。
- 终端显示层：各个端的模板渲染并执行显示的层。当前主要是 velocity 渲染，JS 渲染，JSP 渲染，移动端展示等。
- Web 层：主要是对访问控制进行转发，各类基本参数校验，或者不复用的业务简单处理等。
- Service 层：相对具体的业务逻辑服务层。
- Manager 层：通用业务处理层，它有如下特征
  1）对第三方平台封装的层，预处理返回结果及转化异常信息，适配上层接口。
  2）对 Service 层通用能力的下沉，如缓存方案、中间件通用处理。
  3）与 DAO 层交互，对多个 DAO 的组合复用。
- DAO 层：数据访问层，与底层 MySQL、Oracle、Hbase、OceanBase 等进行数据交互。
- 第三方服务：包括其它部门 RPC 服务接口，基础平台，其它公司的 HTTP 接口，如淘宝开放平台、支付宝付款服务、高德地图服务等。
- 外部数据接口：外部（应用）数据存储服务提供的接口，多见于数据迁移场景中。

2. 【参考】（分层异常处理规约）在 DAO 层，产生的异常类型有很多，无法用细粒度的异常进行 catch，使用 catch(Exception e) 方式，并 throw new DAOException(e)，不需要打印日志，因为日志在 Manager 或 Service 层一定需要捕获并打印到日志文件中去，如果同台服务器再打日志，浪费性能和存储。在 Service 层出现异常时，必须记录出错日志到磁盘，尽可能带上参数和上下文信息，相当于保护案发现场。Manager 层与 Service 同机部署，日志方式与 DAO 层处理一致，如果是单独部署，则采用与 Service 一致的处理方式。Web 层绝不应该继续往上抛异常，因为已经处于顶层，如果意识到这个异常将导致页面无法正常渲染，那么就应该直接跳转到友好错误页面，尽量加上友好的错误提示信息。开放接口层要将异常处理成错误码和错误信息方式返回。

3. 【参考】分层领域模型规约：
   - DO（Data Object）：此对象与数据库表结构一一对应，通过 DAO 层向上传输数据源对象。
   - DTO（Data Transfer Object）：数据传输对象，Service 或 Manager 向外传输的对象。

Java 开发手册