 异常的处理是每个Java程序员时常面对的问题，但是很多人没有原则，遇到异常也不知道如何去处理，于是遇到检查异常就胡乱try...catch...一把，然后e.printStackTrace()一下了事，这种做法通常除了调试排错有点作用外，没任何价值。对于运行时异常，则干脆置之不理。原因是很多开发者缺乏对异常的认识和分析，首先应该明白Java异常体系结构，一种分层继承的关系，你必须对层次结构熟烂于心： Throwable(必须检查) Error（非必须检查） Exception（必须检查） RuntimeException（非必须检查） 一般把Exception异常及其直接子类（除了RuntimeException之外）的异常称之为检查异常。把RuntimeException以及其子类的异常称之为非检查异常，也叫运行时异常。 对于Throwable和Error，则用的很少，一般会用在一些基础框架中，这里不做讨论。 下面针对J2EE的分层架构：DAO层、业务层、控制层、展示层的异常处理做个分析，并给出一般处理准则。 一、DAO层异常处理 如果你用了Spring的DAO模板来实现，则DAO层没有检查异常抛出，代码非常的优雅。但是，如果你的DAO采用了原始的JDBC来写，这时候，你不能不对异常做处理了，因为难以避免的SQLException会如影随形的跟着你。对已这种DAO级别的异常，异常了你又能如何呢？与其这样胡乱try...catch...，囫囵吞枣消灭了异常不如让异常以另外一种非检查的方式向外传递。这样做好处有二： 1）、DAO的接口不被异常所污染，假设你抛出了SQLException，以后

要是换了Spring DAO模板，那DAO接口就不再抛出了SQLException，这样，你的接口抛出异常就是对接口的污染。 2）、DAO异常向外传播给更高层处理，以便异常的错误原因不丢失，便于排查错误或进行捕获处理。 这里还有一个设计上常常令人困扰的问题：很多人会问，那定义一个什么样的异常抛出呢，或者是直接抛出一个throw RuntimeException(e)? 对于这个问题，需要分场合，如果系统小，你可以直接抛出一个throw RuntimeException(e)，但对于一个庞大的多模块系统来说，不要抛这种原生的非检查异常，而要抛出自定义的非检查异常，这样不但利于排错，而且有利于系统异常的处理，通常针对每一个模块，粗粒度的定义一个运行时DAO异常。比如：throw new ModelXxxDAORuntimeException(".....",e)，对于msg信息，你可写也可不写，根据需要灵活抛出。 这里常见一个很愚昧的处理方式，为每个DAO定义一个异常，呵呵，这样累不累啊，有多大意义，在Service层中调用时候，如果要捕获，还要捕获出一堆异常。这样致命的问题是代码混乱，维护困难，阅读也困难，DAO的异常应该是粗粒度的。 二、业务层异常处理 习惯上把业务层称之为Service层或者服务层，Service层的代表的是业务逻辑，不要迷信分太多太多层有多大好处，除非需要，否则别盲目划分不必要的层，层越多，效率越差，根据需要够用就行了。 Service接口中的每个方法代表一个特定的业务，而这个业务一定是一个完整的业务，通常会看到一些傻X的做法，数据库事务配置在 Service层，而Service的实现就是DAO的直接调用，然后在控制层（Action）中，调用了好多Service去完成一个业务，你气得已经无语了，低头找

砖头去！来源：考试大 搞明白以上两个问题后再回过头看异常怎么处理，Service层通常依赖DAO，而Service层的通常也会因为调用别的非检查异常方法而必须面对异常处理的问题，这里和DAO层又有所不同，彼一时，此一时嘛！ 一般来说一个小模块对应一个Service，当然也许有两个或多个，针对这个模块的Service定义一个非检查异常，以应付那些不可避免的异常检查，这个自定义异常可以简单的命名为XxxServiceRuntimeException，将捕获到的异常顺势转译为非检查异常后抛出。我喜欢这么做，因为前台是J2EE应用，前台是web页面，它们的Struts2等框架会自动捕获所有Service层的异常，并把异常交给开发者去自由处理。 但是还有一种情况，由于一些特殊的限制，如果某个异常一旦发生，必须做什么什么处理，而这种处理时硬性要求，或者调用某个Service方法，必须检查处理什么异常，也可以抛出非检查的自定义异常，往往出现这种情况的是政治原因。不推崇这种做法，但也不排斥。 总之，对于接口，尽可能不去用异常污染她！ 三、控制层异常 控制层说的简单些就是常见的Action层，主要是控制页面请求的处理。控制层通常都依赖于Service层，现在比较流行的框架对控制层做得都相当的到位，比如Struts2、SpringMVC等等，他们的控制层框架会捕获业务层的所有异常，并在控制层中声明可能抛出Exception，因此控制层一般不处理什么异常。 如果是控制层中因为调用了一些非检查异常的方法，比如IO操作等，可以简单处理下异常，保证流的安全，这才是目的。 四、显示层异常处理 对于页面异常，处理的方式多种多样，一是不处理异常，一旦异常了，页面就报错。二是定义出错页面，根据异常的类

型以及所在的模块，导航到出错页面。 一般来说，出错页面是更友好的做法。 另外还有特殊的处理方式，展示页面的模板可以捕获异常，并根据情况将异常信息铺到相应的位置，这样就更友好了，不过复杂度较高。 怎么处理，就看需要了。 五、总结 1）、对于异常处理，应该从设计、需要、维护等多个角度综合考虑，有一个通用准则：千万别捕获了异常什么事情都不干，这样一旦出现异常了，你没法依据异常信息来排错。 2）、对于J2EE多层架构系统来说，尽可能避免（因抛出异常带来的）接口污染。 以上论述仅代表个人观点，欢迎交流。