    当你在定义方法时，应该保证此方法的正确性，但是你不能保证你的方法的调用者，他们的输入数据，或者  
    使用环境的正确性，当客户端代码调用此方法时，我们应该向调用方声明可能出现的异常,这样当出现  
    此种异常时，通过方法的文档，他们就清楚到底发生了什么，自己为什么错，（而不是不分青红皂白的埋怨他们调用的  
    代码的作者 “你们写的是    什么狗屁代码，怎么总报错" ,或者”强哥你的代码又有错误了",）合理的异常反馈是 代码库作者  
    ，代码库调用者 之间信息交流的通道，或者说是，代码库作者推卸责任证明自己没有错的挡箭牌（当然你的代码明明有问题，却  
    故意向调用方 抛出输入数据异常，    就是你的职业道德问题了）  
      
    当你在开发初期，设计代码时，可能考虑不清楚可能出现什么异常（然后你没有定义，或者仅仅定义一两个），于是洋洋洒洒写  
    了几周后，发现我需要给某某个方法声明抛出异常，因为用户的输入，太离奇古怪，或者这段代码依赖的环境太不稳定了，需要调  
    用方捕获异常，并做异常处理，这时你抛出了新的异常，改了代码，发现项目中多了很多的编译错误，因为调用方并没有对异常做处理，这时你要  
    求你的调用方作者改代码？当然答案在大部分情况下是否定的。  
      
    以上两种情况仅仅是开发过程中最常见的情景中的两个，还有很多，总之合理的异常声明是 逻辑分层，代码分工，沟通交流的  
    有效保证。  
      
    我需要另外说明的是，当你的代码总是变动，总需要声明很多异常的时候，（三个异常就很多），你首先应该考虑是不是应该对这个  
    方法进行重构了（提取方法，或者提取类），因为频繁的变动代码很大意义上说明，这个方法变动的元素太多，应该分离，或者  
    把变化部分提取出去。这才是正确合理的解决方案。而不是向用户抛出一堆异常。  
      
     那么我们如何向用户正确反馈异常呢？  
     首先我们们可以从JDK的设计学会一些，JDK中所有的异常都继承自Exception（除了Throwable），当我们向上层（调用者）提供服务时，  
     ，一个包应该是高内聚的，一个包下的类合作 完成一项大的任务，然后我们在这一项任务中抽象出来一个异常基类，  
     例如muppet其中一个包是 cn.bronzeware.muppet.sqlgenerate 这个包专门负责生成sql语句的，那么你就可以定义一个  
     异常基类 SqlGenerateException，那么以后这个包下的类，给包外调用方提供的核心方法抛出的异常都应该SqlGenerateException类型，我们  
     可以把这个类声明为抽象异常基类  
     public class SqlGenerateException extends Exception{  
  
            @Override  
            public final String getMessage() {  
                  
                return "{Sql语句生成出错->"+Message();  
            }  
  
            public abstract String Message();  
  
}  
    我们将getMessage方法设为final不能重写，同时其子类必须重写Message（）方法，我们在getMessage方法中有一个  
    默认字符串叫做 “sql语句生成出错”,这个字符串可以放关于异常基类的描述，然后这个字符串在加上 子类的Message方法  
    利用模板方法设计模式，实现基类错误信息和子类错误信息的整合。  
      
    然后子类应该怎么写呢，  
    public class SelectSqlGenerateException extends SqlGenerateException{  
          
        public SelectSqlGenerateException(String message){  
          
            this.message = message;  
        }  
        public SelectSqlGenerateException(){  
          
        }  
          
        private String message = "";  
        public String message(){  
              
            return "Select语句生成出错->"+message+"}";  
        }  
          
    }  
      
      
    当在SelectGenerate生成Select语句时，如果出现关于生成语句的异常可以抛出  
      new SelectGenerateException("这里在进一步指出Select语句具体出现什么异常")  
      这样当SelectGenerate类的调用者在获取到SelectGenerateGenerate异常时  
      通过getMessage()返回的字符串  是"{Sql语句出错-> Select语句出错> Select语句定义错误，缺少逗号}" （举个例子）  
        
      那么这时，SelectGenerate的调用者其实获取的异常类是 SelectSqlGenerateException 类型，它可以通过这种手段  
      获得具体的异常处理  
      } catch (SqlGenerateException e) {  
              
            if(e instanceof SelectSqlGenerateException){  
                //做具体的处理  
              
            }else if/其他处理  
              
              
        }  
    如果它需要针对具体的异常类型做处理，就可以用上述类型检查做特殊处理，否则如果不需要做特殊处理，再向上抛出  
      
          
       当有另外一个包下的类或者其他层次(Sql语句层之上)的类 调用 SqlGenerate层（sql语句生成层）的服务时，他们本身  
       也有一个主要的功能任务，这个功能任务，也需要与之对应的异常信息抽象基类，这个基类同样这样设计  
      muppet中调用Sql语句层的包是cn.bronzeware.muppet.context ,是Context层，这个层负责数据操作，获取实体类信息（注解处理层），生成Sql操作 描述，生成sql语句，调用JDBC进行数据操作，封装结果集，主要是这几个功能，他的异常基类是ContextException()  
      和SqlGenerate同样的设计，作为抽象基类 有一个模板方法 message(),  
      其子类 SelectContextException同样这样设计  
        
       public class ContextException extends Exception{  
  
            private static final long serialVersionUID = -2272596789720418995L;  
              
            @Override  
            public final String getMessage() {  
                  
                return "{数据操作Context层出错->"+Message();  
            }  
  
            public abstract String Message();  
  
}  
        
        
        
      public class SelectContextException extends ContextException{  
          
        public SelectContextException(String message){  
          
            this.message = message;  
        }  
        public SelectContextException(){  
          
        }  
          
        private String message = "";  
        public String message(){  
              
            return "Select操作出错->"+message+"}";  
        }  
          
    }  
     当SelectContext中出现异常时，你可以抛出SelectContext异常，  
     throw new SelectContextException("具体的Select操作可能出现的异常");  
       
     此时可能出现什么情况，即SelectContext中出现的异常是因为SqlGenerate层出现了异常  
     这时你应该捕获SqlGenerateException异常，捕获到这个异常后，可以这样操作  
    } catch (SqlGenerateException e) {  
            throw new SelectContextException(e.getMessage());  
          
    }  
      
      
    应该是 "{数据操作Context层出错->Select操作出错->{Sql语句出错-> Select语句出错> Select语句定义错误，缺少逗号 }}"  
    这样上一层知道了原来SelectContext层出错是因为Sql语句生成的错误，同时又是因为Select语句定义错误，缺少逗号  
      
    这样的错误信息应该是很详细的，更详细的信息，需要你在合适的正确的地方,提供更详细的异常描述，然后做异常抛出，捕获，再抛出，再捕获等等  
    这样，在长长的一套语句的最后(可以调整在最前，读者可以想一想)，总会有错误信息的最准确描述  
      
    当然你依然可以声明第四级异常子类，（Exceptinon ,ContextException,SelectContextException）但是我认为三级足够了，  
    首先，ContextException中有最基本的描述，SelectContextException中还会有描述，你还可以通过SelectContextException的构造方法  
    在传入更具体的错误信息，我觉得足够了，当然你仍然有需求的话，考虑下是不是有必要在划分一个包，或者再声明一个与之前的  
    包内的基类独立的另外一个包内基类。然后再做决定不迟。