Java 异常处理从入门到实战

异常是指程序在运行过程中由于外部问题（如硬件错误、输入错误）等导致的程序异常事件。

异常处理是编程语言或计算机硬件里的一种机制，用于处理软件或信息系统中出现的异常状况（即超出程序正常执行流程的某些特殊的情况）。

本场 Chat 将从以下几个方面介绍异常处理相关知识。

1. 为什么需要异常处理？
2. 你了解异常处理吗？
3. 异常处理有哪些原则？
4. Java 异常处理机制介绍；
5. 异常处理对项目质量的影响；
6. 异常处理对工资待遇的影响。

为什么需要异常处理？

在项目开发的过程中，即使程序员把代码写得尽善尽美，在系统的运行过程中仍然会遇到一些问题。因为很多问题不是靠代码能够避免的，比如：客户输入数据的格式，读取文件是否存在，网络是否始终保持通畅等等。

程序运行时，发生的不被期望的事件，它阻止了程序按照程序员的预期正常执行，或者说程序未按照期望的流程执行，这就导致了异常的发生。

异常发生时，如果不进行任何处理，势必导致程序不能正常运行，不能达到我们所期望的结果。

因此在项目开发过程中，异常处理机制是非常重要而且也是非常必要的，它能使我们对程序运行的结果进行控制及异常发生时进行有效地处理，以达到程序的预期效果。

下面通过“用户注册”的例子说明为什么需要异常处理。

用户注册的用户名称要求长度不能超过32位。

我们考虑一种情况：用户张三，输入的用户名称为“zhonghuarenmingongheguo_zhangsan_666”，长度超过32位。

假设数据库中用户表的用户名称字段最大长度为32位，那么如果不考虑异常情况，势必在用户提交注册后，保存到数据库的时候，数据库一定会报错“value too large for

column”。

这样的话，程序没有做异常处理的话，程序产生的异常会直接反馈到前端界面上，用户体验极差。因此我们需要对这类异常进行有效的处理，以保证良好的用户体验。

你了解异常处理吗？

异常处理是每个项目里非常重要的一部分，项目质量的高低好坏，与异常处理水平的高低好坏紧密相连。最近发现有很多开发者对 Java 异常处理没有自己的认识和标准，有的具备2-3年开发经验却对异常处理仍然没有深刻地认识。

下面我列举几种程序的场景，供大家参考。

1. 通篇的程序没有任何异常处理，任程序自生自灭，不做任何处理。
2. 程序使用 throw 输出错误给用户。
3. 程序使用 try…catch 进行一小段代码的包括，catch 里只打印了堆栈信息，或者 catch 里什么都没做，或者把异常 throws 出去。
4. 程序使用 try…catch 进行整段代码的包括，catch 里只打印了堆栈信息，或者 catch 里什么都没做，或者把异常 throws 出去。
5. 程序使用 try…catch 进行整段代码的包括，catch 里只打印了堆栈信息，并且输出了相应的错误日志。
6. 程序使用 try…catch 进行整段代码的包括，catch 里只打印了堆栈信息，输出了相应的错误日志，并且 throws 了自定义异常，选择友好的提示信息给用户展示。
7. 程序使用 try…catch 进行整段代码的包括，输出了相应的错误日志，如果不是最外层调用则 catch 里不打印堆栈信息，并且 throws 了自定义异常，同时把堆栈信息带出去，最外层调用方，根据自定义异常，选择友好的提示信息给用户展示，并且打印出异常的堆栈信息。即程序根据实际情况对异常进行了综合处理，并通过日志进行了异常的详细记录，以便核查问题。

毫无疑问，第7种是程序进行异常处理的最有效手段。

异常处理有哪些原则

既然我们知道异常处理对项目或者程序意义重大，那么在面对具体的项目情况，面对异常场景时，想要有效地进行异常处理，这里面有哪些原则呢？

我总结为以下**“三大异常处理原则”**。

1.程序层面异常处理原则

- 要避免使用异常处理代替错误处理；

有的人写代码会用异常处理来做判断逻辑或者做业务逻辑处理，这样会降低程序的清晰性，并且效率低下。

- 处理异常不可以代替简单测试，只能在异常情况下使用异常机制；
- 不要进行小粒度的异常处理，应该将整个业务代码包装在一个 try 语句块中；
- 异常往往在高层处理，且不能忽视每一个异常

2.需求层面异常处理原则

- 搞清楚业务边界，用代码防止异常；

在写代码之前，就要明确业务需求，同时了解该需求功能的边界，写代码的时候，用专门的代码来防止异常的发生。

- 给用户友好的提示；

在发现异常情况后，代码进行特殊处理后，给用户友好的提示，不能直接抛出异常到用户界面。

- 程序员不定义业务异常流程。

业务需求是通过代码来实现的，程序员一般不了解业务，因此在写某个业务需求过程中，发现处理不了的异常，一定要与客户或者产品负责人来确定异常流程，从业务逻辑层面，考虑异常处理的业务逻辑，程序员不去定义业务异常流程。

3.接口层面异常处理原则

- 输入、输出参数校验；

接口服务提供方一般要对输入参数做校验，校验通过后，才会提供服务。接口调用方一般要对输出参数做校验，校验通过后，才会使用服务。

- 请求和响应的日志记录；

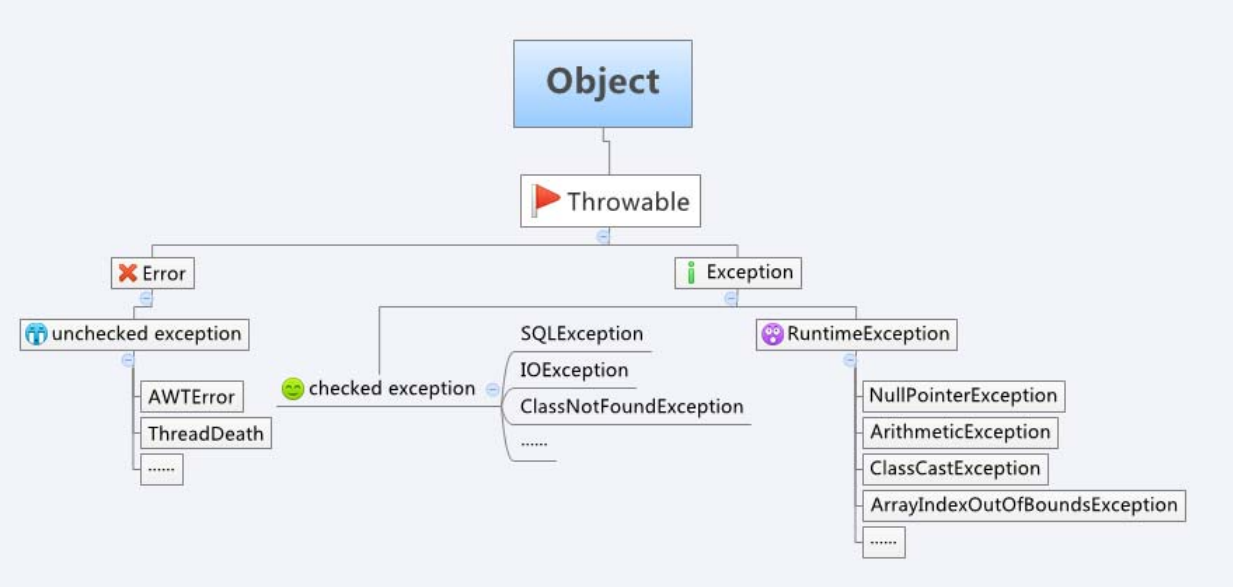
接口要对请求和响应做好日志记录，这在后期遇到问题反查的时候特别重要。

- 接口超时等异常的业务处理。

接口开发时，经常会遇到接口超时等网络问题，因此在开发的过程中，尤其要处理好这些异常。比如：超时反查机制，超时重发机制等。

Java 异常处理机制介绍

Java 异常类层次如下图（ Java 是采用面向对象的方式来处理异常的）。



从上图，可以看出所有的异常跟错误都继承于 Throwable 类，也就是说所有的异常都是一个对象。Exception 类是 Throwable 类的子类，除了 Exception 类外，Throwable 还有一个子类 Error 。

1.Error（错误）

Java 程序通常不捕获错误。错误一般发生在严重故障时，它们在 Java 程序处理的范畴之外。Error 用来指示运行时环境发生的错误。

例如：JVM 运行时出现的 OutOfMemoryError 以及 Socket 编程时出现的端口占用等程序无法处理的错误。程序一般不会从错误中恢复。

2.Exception（异常）

所有的异常类是从 java.lang.Exception 类继承的子类，异常可分为运行时异常跟编译异常。

- 运行时异常（即 RuntimeException 及其之类的异常）

这类异常在代码编写的时候不会被编译器所检测出来，是可以不需要被捕获即可以不处理，但是我们可以编写代码处理（使用 try…catch…finally）这样的异常。对于这些异常，我们应该修正代码，而不是去通过异常处理器处理，这样的异常发生的原因多半是代码写的有问题（一般的做法是进行判断后处理）。

常见的 RUNtimeException 有：NullpointException（空指针异常）、ClassCastException（类型转换异常）、IndexOutOfBoundsException（数组越界异常）等。

- 编译异常（RuntimeException 以外的异常）

这类异常在编译时编译器会提示需要捕获，如果不进行捕获则编译错误。在方法中要么用 try…catch 语句捕获它并处理，要么用 throws 子句声明抛出它，否则编译不会通过。这样的异常一般是由程序的运行环境导致的。因为程序可能被运行在各种未知的环境

下，而程序员无法干预用户如何使用他编写的程序，于是程序员就应该为这样的异常时刻准备着。

常见编译异常有：IOException（流传输异常）、SQLException（数据库操作异常）等。

3.Java处理异常的机制

抛出异常以及捕获异常 ，一个方法所能捕捉的异常，一定是 Java 代码在某处所抛出的异常。简单地说，异常总是先被抛出，后被捕捉的。

- 抛出异常

在执行一个方法时，如果发生异常，则这个方法生成代表该异常的一个对象，停止当前执行路径，并把异常对象提交给 JRE。

- 捕获异常

JRE 得到异常后，寻找相应的代码来处理该异常。JRE 在方法的调用栈中查找，从生成异常的方法开始回溯，直到找到相应的异常处理代码为止。

- 五个关键字：try、catch、finally、throws、throw。

异常处理是通过 try...catch...finally 语句实现的。代码结构如下。

```
try{
    ......   //可能产生异常的代码
}
catch( ExceptionName1 e ){
    ......   //当产生ExceptionName1型异常时的处置措施
}
catch( ExceptionName2 e ){
    ......   //当产生ExceptionName2型异常时的处置措施
}
[ finally{
   ......     //无论是否发生异常，都无条件执行的语句
 }  ]
```

异常处理对项目质量的影响

如果一个项目里的各个功能在异常处理方面下的功夫不够，势必会极大地影响项目的质量。

下面我列举几个例子，供大家参考。

1.空指针异常。

我见过5年开发经验的程序员对空指针居然不做任何处理，空指针的处理是 Java 程序员的基本功，如果连这个都处理不了，真的不能算是合格的 Java 程序员。

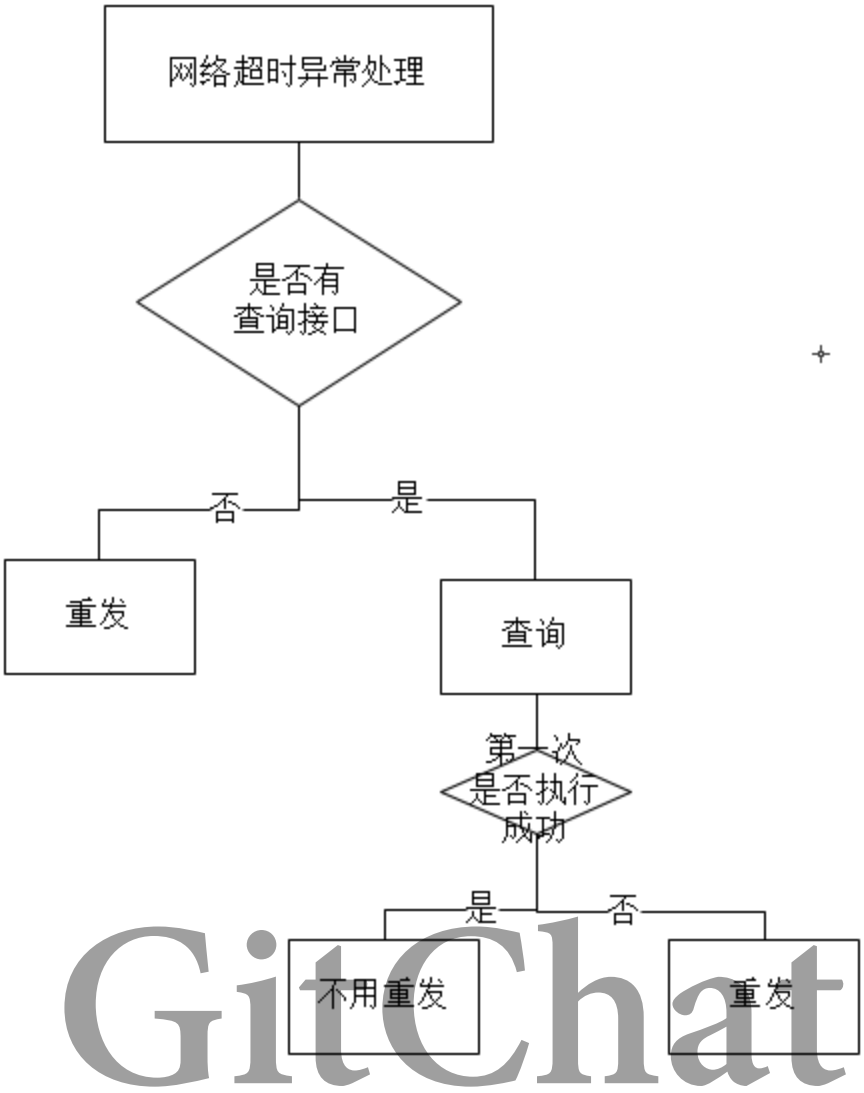
空指针的处理很简单，就是在使用一个对象的时候，记得加一层非空判断就ok了，代码如下所示。

```
if (list != null && list.size() > 0) {
    user = list.get(0);
}
```

2.网络超时异常。

在实际开发项目中，会使用到很多接口服务，比如 WebService 或者 HTTP 接口，这些服务的调用很有可能会导致网络超时等异常，只要发生超时异常，如果不进行处理的话，势必会影响所涉及的功能，进而影响项目质量。

网络超时异常，一般会设置接口超时时间，如果超时了，开发测试过程中去优化服务提供方的代码，看看是不是有不合理的查询或者低效率的查询。作为服务调用方来说，接口超时后，一般会进行接口重发调用，如果接口提供方提供了服务执行结果查询接口的话，那么服务调用方应该在超时处理时，先调用服务执行结果查询接口，根据查询的结果再来确定是否要重新发起服务调用。网络超时处理机制如下图所示。



3.友好的用户体验。

如果后台程序出现异常后，不做任何处理，直接把异常堆栈显示到用户界面里，这种项目的质量是极其差的，尤其是刚做项目的新人要特别注意。

用户体验还包括用户的输入项，及时做校验，及时提醒给用户，除了在前端界面使用 JavaScript 校验外，提交到后端的信息也要再做一次校验，防止程序提交。

4.异常处理与日志打印。

异常处理的同时一定要配合日志使用，用日志记录下来4个 W：What、Where、When、Why，即什么出了错（或者错误是什么），哪里出了错，什么时间出的错，错误产生的原因。

如果有异常处理，但是没有日志记录，那么我们遇到问题，反查的时候就很棘手，无从下手。记录日志一般用 log4j，把日志记录到特定的文件里，而不建议使用 e.printStackTrace() 打印堆栈。

另外最不可取的就是程序代码 catch 了异常，catch 块里什么都没写。这样功能发生异常后，我们根本无法察觉。如下代码所示。

```
try{
    Do something
}catch(Exception e){
   //此处无任何代码（最不可取的代码）
}
```

下面我们举个小例子，来看下 Java 代码，如何合理地处理异常。一个推荐的异常处理实例，如下面代码所示：

```
public User getUserByName(String userName) throws ServiceException {
        User user = null;

        try {
            String hql = "from User t where t.userName = ?";
            List<User> list = hibernateDao.findByHql(hql,
                    new Object[] { userName });
            if (list != null && list.size() > 0) {
                user = list.get(0);
            }
        } catch (ServiceException e) {
            LOGGER.error("getUserByName.ServiceException" +
e.getMessage());
            throw new
ServiceException("getUserByName.ServiceException"
                    + e.getMessage(), e);
        } catch (Exception e) {
            LOGGER.error("ServiceException.Exception" + e.toString());
            throw new ServiceException("ServiceException.Exception"
                    + e.toString(), e);
        }

        return user;
    }
```


在最外层调用处，处理 ServiceException，打印堆栈信息，并转化成友好的提示信息 message 返回给用户界面，请见下面代码。

```
catch (ServiceException e) {
    LOGGER.error("根据姓名查询用户失败" + e.toString());
    message = "根据姓名查询用户失败，姓名为：" + userName;
}
```

异常处理对工资待遇的影响

一个程序员的异常处理能力的高低与项目质量的高低紧密相连。因此，一个优秀的程序员，不仅要能完成项目需求和功能，还需对业务逻辑的异常处理要专业。

有的程序员写的代码只能在正常环境下，正常流程下，程序可以正常执行。凡是稍微与预定的流程不吻合，程序就会出现各种问题，导致程序的功能无法使用。也就是说，程序员在完成业务功能的同时，没有考虑清楚边界。

在项目开发中，一个优秀的程序员和一个新手的区别就是对业务的理解，然后业务转化成代码，并且能考虑各种异常流程和优良的用户体验。

如果能做到在项目中对异常的合理和全面地处理，那么就有理由要求提高自己的工资待遇（可以提高20-30%），因为只有优秀的程序员才能做到面面俱到。

到此，Java 异常处理的 Chat 就结束了，咱们下次再见，谢谢。