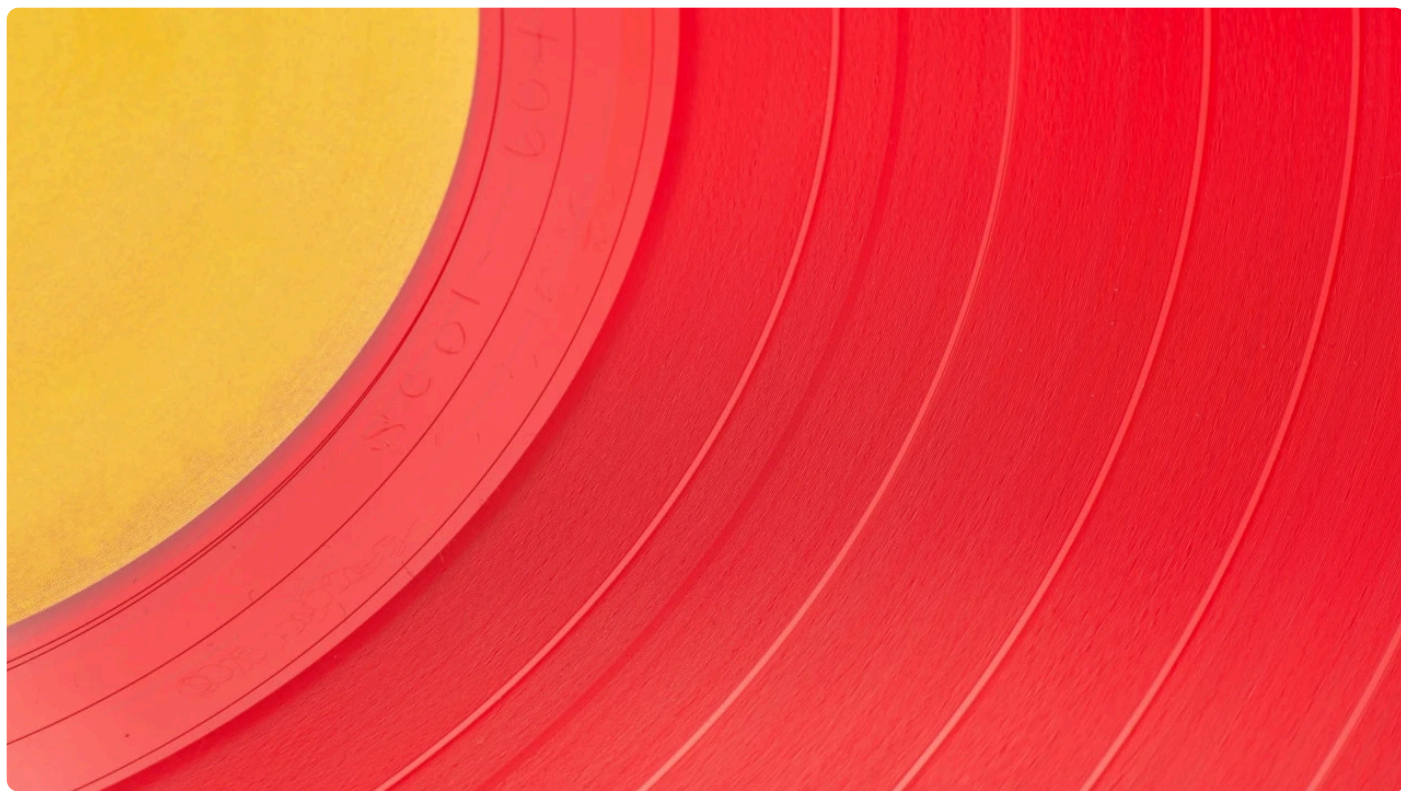


## 37 | 实战二（下）：重构ID生成器项目中各函数的异常处理代码

王争 · 设计模式之美



平时进行软件设计开发的时候，我们除了要保证正常情况下的逻辑运行正确之外，还需要编写大量额外的代码，来处理有可能出现的异常情况，以保证代码在任何情况下，都在我们的掌控之内，不会出现非预期的运行结果。程序的 bug 往往都出现在一些边界条件和异常情况下，所以说，异常处理得好坏直接影响了代码的健壮性。全面、合理地处理各种异常能有效减少代码 bug，也是保证代码质量的一个重要手段。


在上一节课中，我们讲解了几种异常情况的处理方式，比如返回错误码、NULL 值、空对象、异常对象。针对最常用的异常对象，我们还重点讲解了两种异常类型的应用场景，以及针对函数抛出的异常的三种处理方式：直接吞掉、原封不动地抛出和包裹成新的异常抛出。

除此之外，在上一节课的开头，我们还针对 ID 生成器的代码，提出了 4 个有关异常处理的问题。今天，我们就用一节课的时间，结合上一节课讲到的理论知识，来逐一解答一下这几个问题。

话不多说，让我们正式开始今天的内容吧！

## 重构 generate() 函数

首先，我们来看，对于 generate() 函数，如果本机名获取失败，函数返回什么？这样的返回值是否合理？


 复制代码

```
1 public String generate() {
2     String substrOfHostName = getLastFieldOfHostName();
3     long currentTimeMillis = System.currentTimeMillis();
4     String randomString = generateRandomAlphameric(8);
5     String id = String.format("%s-%d-%s",
6         substrOfHostName, currentTimeMillis, randomString);
7     return id;
8 }
```

ID 由三部分构成：本机名、时间戳和随机数。时间戳和随机数的生成函数不会出错，唯独主机名有可能获取失败。在目前的代码实现中，如果主机名获取失败，substrOfHostName 为 NULL，那 generate() 函数会返回类似“null-16723733647-83Ab3uK6”这样的数据。如果主机名获取失败，substrOfHostName 为空字符串，那 generate() 函数会返回类似“-16723733647-83Ab3uK6”这样的数据。

在异常情况下，返回上面两种特殊的 ID 数据格式，这样的做法是否合理呢？这个其实很难讲，我们要看具体的业务是怎么设计的。不过，我更倾向于明确地将异常告知调用者。所以，这里最好是抛出受检异常，而非特殊值。

按照这个设计思路，我们对 generate() 函数进行重构。重构之后的代码如下所示：


 复制代码

```
1 public String generate() throws IdGenerationFailureException {
2     String substrOfHostName = getLastFieldOfHostName();
3     if (substrOfHostName == null || substrOfHostName.isEmpty()) {
4         throw new IdGenerationFailureException("host name is empty.");
5     }
6     long currentTimeMillis = System.currentTimeMillis();
7     String randomString = generateRandomAlphameric(8);
8     String id = String.format("%s-%d-%s",
9         substrOfHostName, currentTimeMillis, randomString);
10 }
```

```
10     return id;
11 }
```

## 重构 getLastFieldOfHostName() 函数

对于 getLastFieldOfHostName() 函数，是否应该将 UnknownHostException 异常在函数内部吞掉（try-catch 并打印日志），还是应该将异常继续往上抛出？如果往上抛出的话，是直接把 UnknownHostException 异常原封不动地抛出，还是封装成新的异常抛出？


 复制代码

```
1  private String getLastFieldOfHostName() {
2      String substrOfHostName = null;
3      try {
4          String hostName = InetAddress.getLocalHost().getHostName();
5          substrOfHostName = getLastSubstrSplittedByDot(hostName);
6      } catch (UnknownHostException e) {
7          logger.warn("Failed to get the host name.", e);
8      }
9      return substrOfHostName;
10 }
```

现在的处理方式是当主机名获取失败的时候，getLastFieldOfHostName() 函数返回 NULL 值。我们前面讲过，是返回 NULL 值还是异常对象，要看获取不到数据是正常行为，还是异常行为。获取主机名失败会影响后续逻辑的处理，并不是我们期望的，所以，它是一种异常行为。这里最好是抛出异常，而非返回 NULL 值。

至于是直接将 UnknownHostException 抛出，还是重新封装成新的异常抛出，要看函数跟异常是否有业务相关性。getLastFieldOfHostName() 函数用来获取主机名的最后一个字段，UnknownHostException 异常表示主机名获取失败，两者算是业务相关，所以可以直接将 UnknownHostException 抛出，不需要重新包裹成新的异常。

按照上面的设计思路，我们对 getLastFieldOfHostName() 函数进行重构。重构后的代码如下所示：

 复制代码

```
1 private String getLastFieldOfHostName() throws UnknownHostException{
2     String substrOfHostName = null;
3     String hostName = InetAddress.getLocalHost().getHostName();
4     substrOfHostName = getLastSubstrSplittedByDot(hostName);
5     return substrOfHostName;
6 }
```

getLastFieldOfHostName() 函数修改之后，generate() 函数也要做相应的修改。我们需要在 generate() 函数中，捕获 getLastFieldOfHostName() 抛出的 UnknownHostException 异常。当我们捕获到这个异常之后，应该怎么办呢？

按照之前的分析，ID 生成失败的时候，我们需要明确地告知调用者。所以，我们不能在 generate() 函数中，将 UnknownHostException 这个异常吞掉。那我们应该原封不动地抛出，还是封装成新的异常抛出呢？

我们选择后者。在 generate() 函数中，我们需要捕获 UnknownHostException 异常，并重新包裹成新的异常 IdGenerationFailureException 往上抛出。之所以这么做，有下面三个原因。


调用者在使用 generate() 函数的时候，只需要知道它生成的是随机唯一 ID，并不关心 ID 是如何生成的。也就是说，这是依赖抽象而非实现编程。如果 generate() 函数直接抛出 UnknownHostException 异常，实际上是暴露了实现细节。

从代码封装的角度来讲，我们不希望将 UnknownHostException 这个比较底层的异常，暴露给更上层的代码，也就是调用 generate() 函数的代码。而且，调用者拿到这个异常的时候，并不能理解这个异常到底代表了什么，也不知道该如何处理。

UnknownHostException 异常跟 generate() 函数，在业务概念上没有相关性。

按照上面的设计思路，我们对 generate() 的函数再次进行重构。重构后的代码如下所示：


```
1 public String generate() throws IdGenerationFailureException {
2     String substrOfHostName = null;
3     try {
4         substrOfHostName = getLastFieldOfHostName();
```

 复制代码

```
5     } catch (UnknownHostException e) {
6         throw new IdGenerationFailureException("host name is empty.");
7     }
8     long currentTimeMillis = System.currentTimeMillis();
9     String randomString = generateRandomAlphameric(8);
10    String id = String.format("%s-%d-%s",
11        substrOfHostName, currentTimeMillis, randomString);
12    return id;
13 }
```

## 重构 getLastSubstrSplittedByDot() 函数

对于 getLastSubstrSplittedByDot(String hostName) 函数，如果 hostName 为 NULL 或者空字符串，这个函数应该返回什么？

 复制代码

```
1  @VisibleForTesting
2  protected String getLastSubstrSplittedByDot(String hostName) {
3      String[] tokens = hostName.split("\\.");
4      String substrOfHostName = tokens[tokens.length - 1];
5      return substrOfHostName;
6  }
```


理论上讲，参数传递的正确性应该有程序员来保证，我们无需做 NULL 值或者空字符串的判断和特殊处理。调用者本不应该把 NULL 值或者空字符串传递给 getLastSubstrSplittedByDot() 函数。如果传递了，那就是 code bug，需要修复。但是，话说回来，谁也保证不了程序员就一定不会传递 NULL 值或者空字符串。那我们到底该不该做 NULL 值或空字符串的判断呢？

如果函数是 private 类私有的，只在类内部被调用，完全在你自己的掌控之下，自己保证在调用这个 private 函数的时候，不要传递 NULL 值或空字符串就可以了。所以，我们可以不在 private 函数中做 NULL 值或空字符串的判断。如果函数是 public 的，你无法掌控会被谁调用以及如何调用（有可能某个同事一时疏忽，传递进了 NULL 值，这种情况也是存在的），为了尽可能提高代码的健壮性，我们最好是在 public 函数中做 NULL 值或空字符串的判断。

那你可能会说，getLastSubstrSplittedByDot() 是 protected 的，既不是 private 函数，也不是 public 函数，那要不要做 NULL 值或空字符串的判断呢？

之所以将它设置为 protected，是为了方便写单元测试。不过，单元测试可能要测试一些 corner case，比如输入是 NULL 值或者空字符串的情况。所以，这里我们最好也加上 NULL 值或空字符串的判断逻辑。虽然加上有些冗余，但多加些检验总归不会错的。

按照这个设计思路，我们对 getLastSubstrSplittedByDot() 函数进行重构。重构之后的代码如下所示：

 复制代码

```
1  @VisibleForTesting
2  protected String getLastSubstrSplittedByDot(String hostName) {
3      if (hostName == null || hostName.isEmpty()) {
4          throw IllegalArgumentException("..."); //运行时异常
5      }
6      String[] tokens = hostName.split("\\.");
7      String substrOfHostName = tokens[tokens.length - 1];
8      return substrOfHostName;
9  }
```

按照上面讲的，我们在使用这个函数的时候，自己也要保证不传递 NULL 值或者空字符串进去。所以，getLastFieldOfHostName() 函数的代码也要作相应的修改。修改之后的代码如下所示：


 复制代码

```
1  private String getLastFieldOfHostName() throws UnknownHostException{
2      String substrOfHostName = null;
3      String hostName = InetAddress.getLocalHost().getHostName();
4      if (hostName == null || hostName.isEmpty()) { // 此处做判断
5          throw new UnknownHostException("...");
6      }
7      substrOfHostName = getLastSubstrSplittedByDot(hostName);
8      return substrOfHostName;
9  }
```



## 重构 generateRandomAlphameric() 函数

对于 generateRandomAlphameric(int length) 函数，如果 length < 0 或 length = 0，这个函数应该返回什么？

 复制代码


```
1  @VisibleForTesting
2  protected String generateRandomAlphameric(int length) {
3      char[] randomChars = new char[length];
4      int count = 0;
5      Random random = new Random();
6      while (count < length) {
7          int maxAscii = 'z';
8          int randomAscii = random.nextInt(maxAscii);
9          boolean isDigit= randomAscii >= '0' && randomAscii <= '9';
10         boolean isUppercase= randomAscii >= 'A' && randomAscii <= 'Z';
11         boolean isLowercase= randomAscii >= 'a' && randomAscii <= 'z';
12         if (isDigit|| isUppercase || isLowercase) {
13             randomChars[count] = (char) (randomAscii);
14             ++count;
15         }
16     }
17     return new String(randomChars);
18 }
19 }
```

我们先来看 length < 0 的情况。生成一个长度为负值的随机字符串是不符合常规逻辑的，是一种异常行为。所以，当传入的参数 length < 0 的时候，我们抛出 IllegalArgumentException 异常。

我们再来看 length = 0 的情况。length = 0 是否是异常行为呢？这就看你自己怎么定义了。我们既可以把它定义为一种异常行为，抛出 IllegalArgumentException 异常，也可以把它定义为一种正常行为，让函数在入参 length = 0 的情况下，直接返回空字符串。不管选择哪种处理方式，最关键的一点是，要在函数注释中，明确告知 length = 0 的情况下，会返回什么样的数据。

## 重构之后的 RandomIdGenerator 代码

对 RandomIdGenerator 类中各个函数异常情况处理代码的重构，到此就结束了。为了方便查看，我把重构之后的代码，重新整理之后贴在这里了。你可以对比着看一下，跟你的重构思路是否一致。

 复制代码

```
1 public class RandomIdGenerator implements IdGenerator {
2     private static final Logger logger = LoggerFactory.getLogger(RandomIdGenerator.class);
3
4     @Override
5     public String generate() throws IdGenerationFailureException {
6         String substrOfHostName = null;
7         try {
8             substrOfHostName = getLastFieldOfHostName();
9         } catch (UnknownHostException e) {
10             throw new IdGenerationFailureException("...", e);
11         }
12         long currentTimeMillis = System.currentTimeMillis();
13         String randomString = generateRandomAlphameric(8);
14         String id = String.format("%s-%d-%s",
15             substrOfHostName, currentTimeMillis, randomString);
16         return id;
17     }
18
19     private String getLastFieldOfHostName() throws UnknownHostException {
20         String substrOfHostName = null;
21         String hostName = InetAddress.getLocalHost().getHostName();
22         if (hostName == null || hostName.isEmpty()) {
23             throw new UnknownHostException("...");
24         }
25         substrOfHostName = getLastSubstrSplittedByDot(hostName);
26         return substrOfHostName;
27     }
28
29     @VisibleForTesting
30     protected String getLastSubstrSplittedByDot(String hostName) {
31         if (hostName == null || hostName.isEmpty()) {
32             throw new IllegalArgumentException("...");
33         }
34
35         String[] tokens = hostName.split("\\.");
36         String substrOfHostName = tokens[tokens.length - 1];
37         return substrOfHostName;
38     }
39
40     @VisibleForTesting
41     protected String generateRandomAlphameric(int length) {
```



```
42     if (length <= 0) {
43         throw new IllegalArgumentException("...");
44     }
45
46     char[] randomChars = new char[length];
47     int count = 0;
48     Random random = new Random();
49     while (count < length) {
50         int maxAscii = 'z';
51         int randomAscii = random.nextInt(maxAscii);
52         boolean isDigit= randomAscii >= '0' && randomAscii <= '9';
53         boolean isUppercase= randomAscii >= 'A' && randomAscii <= 'Z';
54         boolean isLowercase= randomAscii >= 'a' && randomAscii <= 'z';
55         if (isDigit|| isUppercase || isLowercase) {
56             randomChars[count] = (char) (randomAscii);
57             ++count;
58         }
59     }
60     return new String(randomChars);
61 }
62 }
```

## 重点回顾

好了，今天的内容到此就讲完了。我们一块来总结回顾一下，你需要重点掌握的内容。

今天的内容比较偏实战，是对上节课学到的理论知识的一个应用。从今天的实战中，你学到了哪些更高层的软件设计和开发思想呢？我这里抛砖引玉，总结了下面 3 点。

再简单的代码，看上去再完美的代码，只要我们下功夫去推敲，总有可以优化的空间，就看你愿不愿把事情做到极致。

如果你内功不够深厚，理论知识不够扎实，那你就很难参透开源项目的代码到底优秀在哪里。就像如果我们没有之前的理论学习，没有今天我给你一点一点重构、讲解、分析，只是给你最后重构好的 RandomIdGenerator 的代码，你真的能学到它的设计精髓吗？

对比 [第 34 节课](#) 最初小王的 IdGenerator 代码和最终的 RandomIdGenerator 代码，它们一个是“能用”，一个是“好用”，天壤之别。作为一名程序员，起码对代码要有追求啊，不然跟咸鱼有啥区别！

## 课堂讨论

我们花了 4 节课的时间，对一个非常简单的、不到 40 行的 ID 生成器代码，做了多次迭代重构。除了刚刚我在“重点回顾”中讲到的那几点之外，从这个迭代重构的过程中，你还学到哪些更有价值的东西？

欢迎在留言区写下你的思考和想法，和同学一起交流和分享。如果有收获，也欢迎你把这篇文章分享给你的朋友。

### AI智能总结

重构ID生成器项目中的异常处理代码是本文的核心内容。文章首先讨论了generate()函数中主机名获取失败时返回NULL值的合理性，并提出了抛出受检异常的重构方案。其次，对getLastFieldOfHostName()函数和getLastSubstrSplittedByDot()函数进行了重构，加入了对异常情况的处理逻辑。通过这些重构，文章强调了全面、合理地处理各种异常能有效减少代码bug，提高代码的健壮性。重构后的RandomIdGenerator代码展示了对各个函数异常情况处理代码的重构，提供了实际的重构思路和方法。读者可以从中学习到更高层的软件设计和开发思想，以及对代码优化的追求。通过对一个简单的ID生成器代码的多次迭代重构，读者可以领悟到代码优化的重要性和价值。整体而言，本文通过具体的代码示例和分析，为读者提供了实际的重构思路和方法，有助于读者在实际开发中更好地处理异常情况，提高代码质量。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

## 全部留言 (89)

最新 精选



Jxin

2020-01-27

还学到什么：

1.一下子想搞个例子讲这些真的太难了，拍着脑子想demo。栏主这个demo背景简单，也将要讲的内容串起来了，实属不易，辛苦栏主了。

个人见解：

1.按我的习惯，我会尽量把入参和中间不可靠变量的异常校验都放在public方法，所有私有方法都以契约的方式不再做参数校验。也就是说 public方法干 1.参数校验 2.系统一级流程编排 3.统一异常处理 这三件事。所以对private方法的提炼会和栏主有点出入。

2.如果这个id生成器还要带有业务key，比如分表路由key之类的东西。那么这个实现就还得大

动干戈。但凡这种涉及持久数据的玩意，很可能需要考虑新老版本兼容的问题，也就是如何平滑过度老数据。所以需要在id生成算法上引入版本或者类型的标记，把标记打在持久化的数据上，以备平滑过度老数据。

作者回复: 我觉得你懂我~

共 13 条评论 >

👍 212



高源

2020-01-27

希望老师每节课举的代码有下载的地方，自己下载下来结合老师讲解的，自己理解体会其中的解决问题

作者回复: 好的，等我俩月，我整理好，一块放到github上：

<https://github.com/wangzheng0822>

共 6 条评论 >

👍 10



田园牧歌

2020-06-24

看了争哥专栏，受益匪浅，了解了如何评判一个项目、一段代码的好烂，如何写出高质量代码。但我有一个疑问，就是像这种基于接口而非实现的编程方式，在实际的业务项目中如何分包和模块呢？如果是静态方法的小算法我可以放到util包中，比如命名为IdGeneratorUtil.java

作者回复: 划分模块跟划分类的方法一样，做到高内聚低耦合，职责单一。你可以参考面向对象设计的4个步骤来做模块的定义和划分。



👍 3



Kelly.W

2020-11-26

开发能用的代码可能会很快也比较简单。但想要开发出优秀的代码，就需要投入比较多的精力，一轮轮来优化。

能用的代码和优质代码之间最大的区别就在于细节，像这节课中讲到的异常抛出，特殊值处理等等都是细节。

这就是60分和100分的差别。

作者回复: 嗯嗯



👍 2



**Clear**

2020-06-21

王老师好，我有一个问题： 根据函数是否关心异常类型，来判断异常是否需要转换抛出，会不会导致需要新增很多异常的类呢。

作者回复: 会，但为了清晰、易于理解，多几个异常也未尝不可啊



👍 1



**空白昵称**

2020-03-13

我觉得抛异常这件事，有点像开发经理的职责。下级有问题（异常）要反馈，然后自己能处理则处理（吞掉异常）。如果自己不能处理的，要向上级汇报，那么汇报的时候就要考虑，如果上级不关心底层研发问题，则上报自己汇总的问题（re-throw新的异常）。如果上级领导也是技术控，对底层很了解，那么可适当直接上报（直接re-throw下级反馈的异常）。

共 7 条评论 >

👍 160



**undefined**

2020-02-01

个人见解：如果 id 生成器需要应用到生产环境，类似 hostname 获取失败的问题，需要由生成器本身给出降级方案。一来为了 id 格式统一，二来假若抛给业务，业务对于这种系统底层的失败，也没有什么好的解决方法。

共 9 条评论 >

👍 58



**Geek\_kobe**

2020-01-27

果然还是看技术文章能让恐慌的心静下来

共 1 条评论 >

👍 26



**Harvey**

2020-01-27

设计之所以难是因为没有标准答案，很多权衡是依赖于具体业务的。这就是DDD的思想所在，要先想清楚问题域是什么在思考解决方案。很多开发讨论问题的时候没有层次，上来就陷

入技术细节，这就叫缺乏抽象。下游系统要想清楚哪些是上游系统给你提供的服务？哪些是人家的内部技术实现？比如ID生成，作为上游系统，ID生成服务提供的是有小概率重复的随机ID服务，至于随机算法，下游系统不必关心，这是上游系统的内部实现，这样上游系统才有空间更换算法而不影响下游系统。

共 3 条评论 >

 24



xk\_

2020-03-16

很多小伙伴说，generate方法不需要抛出异常。

对，如果这个业务是仅仅只是作为唯一id，那么不管怎么样一定要生成一个随机数的。

那么，放开来想这个业务并非是一个id生成器，而是生成一个id业务码，这个id后面会用个主机名，对流量进行识别。争哥这么抛出异常，我觉得是可行的。还是要看具体的业务

再说一句，主机名还是一个很重要的东西，hadoop集群获取不到主机名，是一个很严重的问题。

共 1 条评论 >

 13