李政的专栏

欢迎交流!

muziazheng

关注 发私信

Throwable类的详细解释

2016-01-14 10:34 4256人阅读 评论(0) 收藏 举报

分类：

javase（176）

概述 软件包 类 使用 树 已过时 索引 帮助

JavaTM 2 Platform

Standard Ed. 5.0

上一个类 下一个类 框架 无框架 所有类

摘要： 嵌套 | 字段 | 构造方法 | 方法 详细信息： 字段 | 构造方法 | 方法

---

object 老祖宗

要点:是所有异常信息的祖宗类

java.lang

类 Throwable

java.lang.Object

继承者 java.lang.Throwable

所有已实现的接口：

java.io.Serializable

直接已知子类：

Error, Exception

---

public class Throwableextends Objectimplements java.io.SerializableThrowable 类是 Java 语言中所有错误或异常的超类。只有当对象是此类（或其子类之一）的实例时，才能通过 Java 虚拟机

或者 Java throw 语句抛出。类似地，只有此类或其子类之一才可以是 catch 子句中的参数类型。

两个子类的实例，Error 和 Exception，通常用于指示发生了异常情况。通常，
况的上下文中新近创建的，因此包含了相关的信息（比如堆栈跟踪数据）。

Throwable 包含了其线程创建时线程执行堆栈的快照。它还包含了给出有关错
符串。最后，它还可以包含 cause（原因）：另一个导致此 throwable 抛出的 thro
cause 设施在 1.4 版本中首次出现。它也称为异常链 设施，因为 cause 自身也会有 `` `` 类
推，就形成了异常链，每个异常都是由另一个异常引起的。

导致 throwable 可能具有 cause 的一个原因是，抛出它的类构建在低层抽象的基础上，从而高层操作由于低层操作的失败而失败。因此让低层抛出的 throwable 向外传播并非一种好的设计方法，因为它通常与高层提供的抽象不相关。此外，这样做会将高层 API 与其实现细节关联起来，并认为低层异常是经过检查的异常。抛出“经过包装的异常”（即包含 cause 的异常）允许高层与其调用方交流失败详细信息，而不会招致上述任何一个缺点。这种方式保留了改变高层实现而不改变其 API 的灵活性（尤其是，异常集合通过其方法抛出）。

导致 throwable 可能具有 cause 的另一个原因是，抛出它的方法必须符合不允许方法直接抛出 cause 的通用接口。例如，假定持久集合符合 Collection 接口，而其持久性在 java.io 的基础上实现。假定 put 方法的内部可以抛出 IOException。实现可以与其调用方交流 IOException 的详细消息，同时通过以一种合适的未检查的异常来包装 IOException，使其符合 Collection 接口。（持久集合的规范应该指示它能够抛出这种异常。）

Cause 可以通过两种方式与 throwable 关联起来：通过一个将 cause 看作参数的构造方法；或者通过 initCause(Throwable) 方法。对于那些希望将 cause 与其关联起来的新 throwable 类，应该提供带有 cause 的构造方法，并委托（可能间接）给一个带有 cause 的 Throwable 构造方法。例如：

```
try {
    lowLevelOp();
} catch (LowLevelException le) {
    throw new HighLevelException(le);  // Chaining-aware constructor
}
```

因为 initCause 方法是公共的，它允许 cause 与任何 throwable 相关联，甚至包括“遗留

throwable” ，它的实现提前将异常链机制的附件应用到 Throwable。例如：

```
try {
    lowLevelOp();
} catch (LowLevelException le) {
    throw (HighLevelException)
          new HighLevelException().initCause(le);  // Legacy constructor
}
```

在版本 1.4 之前，许多 throwable 有自己的非标准异常链机制（ ExceptionIn
ClassNotFoundException、 UndeclaredThrowableException、 InvocationTarg
WriteAbortedException、 PrivilegedActionException、 PrinterIOException、
RemoteException 和 javax.naming.NamingException ）。所有这些 throwable 都已经更新，
可以使用标准异常链机制，同时继续实现其“遗留”链机制，以保持兼容性。

此外，从版本 1.4 开始，许多通用的 Throwable 类（例如，Exception、RuntimeException、Error ）都已经更新，具有带 cause 的构造方法。由于有 initCause 方法存在，这并非严格的要求，但委托给一个带有 cause 的构造方法将更方便也更形象。

根据惯例，Throwable 类及其子类有两个构造方法，一个不带参数，另一个带有 String 参数，此参数可用于生成详细消息。此外，对于那些与其相关联的 cause 的子类，它们应有两个构造方法，一个带 Throwable(cause)，一个带 String（详细消息）和 Throwable(cause)。

在版本 1.4 中还引入了 getStackTrace() 方法，它允许通过各种形式的 printStackTrace() 方法编程访问堆栈跟踪信息，这些信息以前只能以文本形式使用。此信息已经添加到该类的序列化表示形式，因此 getStackTrace 和 printStackTrace 将可在反序列化时获得的 throwable 上正确操作。

从以下版本开始：

JDK1.0

另请参见：

序列化表格

构造方法摘要

Throwable()

构造一个将 null 作为其详细消息的新 throwable。

Throwable(String message)

构造带指定详细消息的新 throwable。

Throwable(String message, Throwable cause)

构造一个带指定详细消息和 cause 的新 throwable。

Throwable(Throwable cause)

构造一个带指定 cause 和 (cause==null ? null :cause.toString()) (它通常包含类和 e 的详细消息) 的详细消息的新 throwable。

方法摘要

Throwable fillInStackTrace()

记录异常堆栈跟踪。

Throwable getCause()

返回此 throwable 的 cause；或者如果 cause 不存在或未知，则返回 null。

String getLocalizedMessage()

创建此 throwable 的本地化描述。

String getMessage()

返回此 throwable 的详细消息字符串。

StackTraceElement[] getStackTrace()

提供编程访问由 printStackTrace() 输出的堆栈跟踪信息。

Throwable initCause(Throwable cause)

将此 throwable 的 cause 初始化为指定值。

void printStackTrace()

将此 throwable 及其追踪输出至标准错误流。

void printStackTrace(java.io.PrintStream s)

将此 throwable 及其追踪输出到指定的输出流。

void printStackTrace(java.io.PrintWriter s)

将此 throwable 及其追踪输出到指定的 PrintWriter。

void setStackTrace(StackTraceElement[] stackTrace)

设置将由 getStackTrace() 返回，并由 printStackTrace() 和相关方法输出的堆栈跟踪元素。

String toString()

返回此 throwable 的简短描述。

从类 java.lang.Object 继承的方法

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

构造方法详细信息

Throwable

public Throwable()构造一个将 null 作为其详细消息的新 throwable。Cause i

在以后通过调用 initCause(java.lang.Throwable) 来初始化。

调用 fillInStackTrace() 方法来初始化新创建的 throwable 中的堆栈跟踪数据。

Throwable

public Throwable(String message)构造带指定详细消息的新 throwable。Cause 尚未进行初始化，可在以后通过调用 initCause(java.lang.Throwable) 来初始化。

调用 fillInStackTrace() 方法来初始化新创建的 throwable 中的堆栈跟踪数据。

参数：

message - 详细消息。保存这个详细消息，以便以后通过 getMessage() 方法对其进行检索。

Throwable

public Throwable(String message,

Throwable cause)构造一个带指定详细消息和 cause 的新 throwable。

注意，与 cause 相关的详细消息不会 被自动合并到此 throwable 的详细消息中的。

调用 fillInStackTrace() 方法来初始化新创建的 throwable 中的堆栈跟踪数据。

参数：

message - 详细消息（保存此消息，以便以后通过 getMessage() 方法检索它）

cause - 原因（保存此 cause，以便以后通过 getCause() 方法检索它）。（允许

cause 是不存在的或是未知的。）

从以下版本开始：

1.4

Throwable

public Throwable(Throwable cause)构造一个带指定 cause 和 (cause==null ? null :cause.toString()) （它通常包含类和 cause 的详细消息）的详细消息的新 throwable。此构造方法对于那些与其他 throwable（例如，PrivilegedActionException）的包装器相同的 throwable 来说是有用的。

调用 fillInStackTrace() 方法来初始化新创建的 throwable 中的堆栈跟踪数据。

参数：

cause - 原因（保存此 cause，以便以后通过 getCause() 方法检索它）。（允许 null 值，指出 cause 是不存在的或是未知的。）

从以下版本开始：

1.4

方法详细信息

getMessage

public String getMessage()返回此 throwable 的详细消息字符串。

返回：

此 Throwable 实例的详细消息字符串（可以为 null）。

---

getLocalizedMessage

public String getLocalizedMessage()创建此 throwable 的本地化描述。子类便生成特定于语言环境的消息。对于不重写此方法的子类，默认实现返回与 get 结果。

返回：

此 throwable 的本地化描述。

从以下版本开始：

JDK1.1

---

getCause

public Throwable getCause()返回此 throwable 的 cause；或者如果 cause 不存在或未知，则返回 null。（该 Cause 是导致抛出此 throwable 的throwable。）

此实现返回由一个需要 Throwable 的构造方法提供的 cause，或者在创建之后通过 initCause(Throwable) 方法进行设置的 cause。虽然通常不需要重写此方法，但子类可以重写它，以返回一个通过某些其他方式设置的 cause。这适用于在异常链（异常嵌套）机制被加入到 Throwable 之前存在“遗留 Throwable 链机制”的情况。注意，不必 重写任何 PrintStackTrace 方法，所有方法都调用 getCause 方法来确定 throwable 的 cause。

返回：

此 throwable 的 cause，如果 cause 不存在或是未知的，则返回 null。

从以下版本开始：

1.4

---

initCause

public Throwable initCause(Throwable cause)将此 throwable 的 cause 初始化为指定值。（该 Cause 是导致抛出此 throwable 的throwable。）

此方法至多可以调用一次。此方法通常从构造方法中调用，或者在创建 throwa

果此 throwable 通过 Throwable(Throwable) 或 Throwable(String,Throwab

至一次也不能调用。

参数：

cause - 原因（保存此 cause，以便以后通过 getCause() 方法检索它）。（允许 null 值，指出 cause 是不存在的或是未知的。）

返回：

对此 Throwable 实例的引用。

抛出：

IllegalArgumentException - 如果 cause 是此 throwable。（throwable 不能是它自己的 cause。）

IllegalStateException - 如果此 throwable 通过 Throwable(Throwable) 或 Throwable(String,Throwable) 创建，或者此方法已经在此 throwable 上进行调用。

从以下版本开始：

1.4

---

toString

public String toString()返回此 throwable 的简短描述。如果此 Throwable 对象是利用非空详细消息字符串创建的，则结果是三个字符串的串联：

此对象的实际类的名称

": "（冒号和空格）

此对象的 getMessage() 方法的结果

如果此 Throwable 对象利用 null 详细消息字符串创建，则返回此对象的实际类的名称。

覆盖：

类 Object 中的 toString

返回：

该 throwable 的字符串表示形式。

---

printStackTrace

public void printStackTrace()将此 throwable 及其追踪输出至标准错误流。此
Throwable 对象的堆栈跟踪输出至错误输出流，作为字段 System.err 的值。输
象的 toString() 方法的结果。剩余行表示以前由方法 fillInStackTrace() 记录的数据。 式
取决于实现，但以下示例是最常见的：

```
java.lang.NullPointerException
        at MyClass.mash(MyClass.java:9)
        at MyClass.crunch(MyClass.java:6)
        at MyClass.main(MyClass.java:3)
```

本示例通过运行以下程序生成：

```
class MyClass {
    public static void main(String[] args) {
        crunch(null);
    }
    static void crunch(int[] a) {
        mash(a);
    }
    static void mash(int[] b) {
        System.out.println(b[0]);
    }
}
```

对于带初始化非空 cause 的 throwable 的追踪，通常应该包括 cause 的追踪。此信息的格式取决于实现，但以下示例是最常见的：

```
HighLevelException: MidLevelException: LowLevelException
        at Junk.a(Junk.java:13)
        at Junk.main(Junk.java:4)
Caused by: MidLevelException: LowLevelException
        at Junk.c(Junk.java:23)
        at Junk.b(Junk.java:17)
        at Junk.a(Junk.java:11)
        ... 1 more
Caused by: LowLevelException
        at Junk.e(Junk.java:30)
        at Junk.d(Junk.java:27)
        at Junk.c(Junk.java:21)
```

关闭

```
        ... 3 more
```

注意包含字符 "..." 的行。这些行指示此异常的椎栈跟踪的其余部分是从“终止”异常（由此异常引起的异常）的堆栈跟踪底部算起的指定数量帧。这种简写形式可以大大缩短通常情况下的输出长度，通常情况下抛出经过包装的异常所采用的方法与捕获“作为 cause 的异常”所述示例通过运行以下程序生成：

```
public class Junk {
    public static void main(String args[]) {
        try {
            a();
        } catch(HighLevelException e) {
            e.printStackTrace();
        }
    }
    static void a() throws HighLevelException {
        try {
            b();
        } catch(MidLevelException e) {
            throw new HighLevelException(e);
        }
    }
    static void b() throws MidLevelException {
        c();
    }
    static void c() throws MidLevelException {
        try {
            d();
        } catch(LowLevelException e) {
            throw new MidLevelException(e);
        }
    }
    static void d() throws LowLevelException {
       e();
    }
    static void e() throws LowLevelException {
        throw new LowLevelException();
    }
}


class HighLevelException extends Exception {
```

```
    HighLevelException(Throwable cause) { super(cause); }
}

class MidLevelException extends Exception {
    MidLevelException(Throwable cause)  { super(cause); }
}

class LowLevelException extends Exception {
}
```

---

printStackTrace
public void printStackTrace(java.io.PrintStream s)将此 throwable 及其追踪输出到指定的输出流。

参数：
s - 用于输出的 PrintStream

---

printStackTrace
public void printStackTrace(java.io.PrintWriter s)将此 throwable 及其追踪输出到指定的 PrintWriter。

参数：
s - 用于输出的 PrintWriter
从以下版本开始：
JDK1.1

---

fillInStackTrace

public Throwable fillInStackTrace()记录异常堆栈跟踪。此方法在此 Throwab
线程堆栈帧的当前状态有关的信息。

返回：

对此 Throwable 实例的引用。

另请参见：

printStackTrace()

---

getStackTrace

public StackTraceElement[] getStackTrace()提供编程访问由 printStackTrace() 输出的堆栈跟踪信息。返回堆栈跟踪元素的数组，每个元素表示一个堆栈帧。数组的第零个元素（假定数据的长度为非零）表示堆栈顶部，它是序列中最后的方法调用。通常，这是创建和抛出该 throwable 的地方。数组的最后元素（假定数据的长度为非零）表示堆栈底部，它是序列中第一个方法调用。

某些虚拟机在某些情况下可能会省略堆栈跟踪中的一个或多个堆栈帧。在极端情况下，没有该 throwable 堆栈跟踪信息的虚拟机可以从该方法返回一个零长度数组。一般说来，对于由 printStackTrace 输出的每个帧，此方法返回的数组都将包含一个对应的元素。

返回：

堆栈跟踪元素的数组，表示与此 throwable 相关的堆栈跟踪。

从以下版本开始：

1.4

---

setStackTrace

public void setStackTrace(StackTraceElement[] stackTrace)设置将由 getStackTrace() 返回，并由 printStackTrace() 和相关方法输出的堆栈跟踪元素。 此方法设计用于 RPC 框架和其他高级系

统，允许客户端重写默认堆栈跟踪，这些默认堆栈跟踪要么在构造 throwable 时由 fillInStackTrace() 生成，要么在从序列化流读取 throwable 时反序列化。

参数：

stackTrace - 要与此 Throwable 关联的堆栈跟踪元素。指定的数组由此调用复制后，指定数组中的改变将不会对此 Throwable 的堆栈跟踪产生影响。

抛出：

NullPointerException - 如果 stackTrace 为 null，或者如果 stackTrace 中的任何元

从以下版本开始：

1.4

概述 软件包 类 使用 树 已过时 索引 帮助

JavaTM 2 Platform

Standard Ed. 5.0

上一个类 下一个类 框架 无框架 所有类

摘要： 嵌套 | 字段 | 构造方法 | 方法 详细信息： 字段 | 构造方法 | 方法

提交错误或意见

有关更多的 API 参考资料和开发人员文档，请参阅 Java 2 SDK SE 开发人员文档。该文档包含更详细的、面向开发人员的描述，以及总体概述、术语定义、使用技巧和工作代码示例。

- 上一篇 类 System
- 下一篇 java160109DeadLockTest

- C++类overlode、override和overwrite详细解释
- 【转】Java中的异常、断言、日志

猜你在找

深度学习基础与TensorFlow实践
【在线峰会】一天掌握物联网全栈开发之道
机器学习40天精英计划
微信小程序开发实战
备战2017软考 系统集成项目管理工程师 学习套餐
【在线峰会】前端开发重点难点技
【在线峰会】如何高质高效的进行
Python数据挖掘与分析速成班
JFinal极速开发企业实战
Python大型网络爬虫项目开发实战

查看评论

暂无评论

您还没有登录,请[登录]或[注册]

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

关闭