

Cucumber 使用进阶

概览

吕双涛, 刘娟, 和 柴瑞亚

2016 年 4 月 11 日发布

如何使用 Cucumber

如何避免 Cucumber 测试用例报告失真

Cucumber 是什么

如何与持续集成工具集成

Cucumber 是 BDD 模式下实现可执行规范（Executable Specifications）的开源工具，但是它的试，更加重要的在于其能够在团队成员之间构建统一的交流基础（feature 文件）、规范交流用 Language）、提高各个利益相关方（Business Stakeholders）沟通效率和效果，从而达到提升产品这一最终目标。

如何使用 Cucumber

Cucumber 有很多种语言的实现版本，例如 Java、Ruby、.NET、JavaScript 等等，并且 Cucumber 地集成，常见的 Selenium、SpringFramework、Ruby on Rails 等，能够方便地引入到您的测试本文以一个 Java 测试项目为例，介绍如何使用 Cucumber 的 Java 语言实现版本：Cucumber-

将 Cucumber-JVM 依赖加入到项目中

如果您的项目是使用 Maven 管理所依赖的第三方依赖 jar 包，那么引入 Cucumber-JVM 将是一单的将如下的 Code Snippet 加入到项目的 pom.xml 的“dependencies”下即可：

清单 1. pom.xml 中 dependencies 下添加内容

```
1 <dependency>
2 <groupId>info.cukes</groupId>
3 <artifactId>cucumber-java</artifactId>
4 <version>${cucumber.version}</version>
5 <scope>test</scope>
6 </dependency>
```

```
7 <dependency>
8   <groupId>info.cukes</groupId>
9   <artifactId>cucumber-junit</artifactId>
10  <version>${cucumber.version}</version>
11  <scope>test</scope>
12 </dependency>
13 <dependency>
14   <groupId>junit</groupId>
15   <artifactId>junit</artifactId>
16   <version>${junit.version}</version>
17   <scope>test</scope>
18 </dependency>
```

关于 version，请尽量选择适合自己项目的，如果您使用的是 Java 7，没有特殊的要求可以与本用’<cucumber.version>1.2.2</cucumber.version>’。

编写 Executable Specification

Cucumber 之所以受到如此的推崇，与其 Executable Specification 这个特性不无关系。顾名思义的意义：

- 可执行性 (Executable)：您可以像执行代码 (Java、Ruby...) 一样运行这些规范，来验证是从技术人员的视角来看的；
- 规范性 (Specification)：从非技术人员的视角触发，相比验证本身，他们更加关心系统功能能够做什么样的事情。

这看似简单的两方面似乎关联并不是很大，但是如何能够在同一个基础 (feature files) 之上做大的妙处。从项目管理人员的角度来看，Cucumber 是技术人员和非技术人员交流的桥梁，从而能够增加各个利益相关方的沟通，因为只有深入的沟通，在各方都理解了真正期望的功能这一基础 Executable Specification！

回归到工具这一层面，Cucumber 是以 feature 文件来组织测试的，相信大家都很清楚这里之所以是为了凸显用户在使用系统中所能够享受到的服务和功能。以 ATM 取钱场景为例子，需要如下

1. 创建 feature 文件；
2. 生成测试 Step Definitions；
3. 运行测试用例。

下面来具体说明如何在测试工作中使用 Cucumber。

创建 Feature 文件

自动柜员机（ATM）大家都非常熟悉，现在假设您在为某一个银行所提供的固定金额取款功能讨论之后，针对这一功能，你们得出了如下的场景定义，此处以 Feature 文件的形式写出来：

清单 2. src/main/resources/features/FixedAmountWithdraw.feature

```

1  # language: zh-CN
2  功能：使用 ATM 固定金额方式取款
3      通常“取款”菜单包含了几个固定金额，使用这些固定金额取款可以避免从键盘输入提取金额，从而可以
4
5      场景大纲：固定金额取款
6      假如 我的账户中有余额"<accountBalance>"元
7      当 我选择固定金额取款方式取出"<withdrawAmount>"元
8      那么 我应该收到现金"<receivedAmount>"元
9      而且 我账户的余额应该是"<remainingBalance>"元
10     例子：
11     | accountBalance | withdrawAmount | receivedAmount | remainingBalance |
12     | 1000.00 | 100.00 | 100.00 | 900.00 |
13     | 500.00 | 500.00 | 500.00 | 0.00 |

```

上述文件中，需要说明的是：

1. `# language: zh-CN` 表明 feature 文件中所使用的描述语言是中文简体，Cucumber 本身支持 Spoken Language 而非 Programming Language)

通过 `java cucumber.api.cli.Main --i18n help` 查看所支持的所有 Spoken Language；

如果您需要像这样的多层次列表项，可以通过项目符号按钮旁边的右箭头按钮（增加缩进量按钮）

- 通过 `java cucumber.api.cli.Main --i18n LANG` 查看所给定语言支持的所有关键字，比如查 Gherkin 关键字，可以通过执行命令 `java cucumber.api.cli.Main --i18n zh-CN` 查看，其输

清单 3. Cucumber 支持语言输出内容

```

1  | feature | "功能" |
2  | background | "背景" |
3  | scenario | "场景", "剧本" |
4  | scenario outline | "场景大纲", "剧本大纲" |
5  | examples | "例子" |
6  | given | "★", "假如", "假设", "假定" |
7  | when | "★", "当" |
8  | then | "★", "那么" |
9  | and | "★", "而且", "并且", "同时" |
10 | but | "★", "但是" |
11 | given (code) | "假如", "假设", "假定" |
12 | when (code) | "当" |
13 | then (code) | "那么" |
14 | and (code) | "而且", "并且", "同时" |
15 | but (code) | "但是" |

```

2. 采用中文描述 feature 文件，首先得益于 Cucumber 本身的支持，但是另外一个最重要的原因利益相关方清楚地读懂，使用利益相关方的 Spoken Language 来撰写规范可以使沟通更加方便，

3. 如果不使用`# language: zh-CN`这个 header，默认情况下，Cucumber 会以英语解析 feature Gherkin 的关键词必须使用英文。

清单 4. 英文关键字描述的 Feature 文件

```

1 Feature: 使用 ATM 固定金额方式取款
2 通常“取款”菜单包含了几个固定金额，使用这些固定金额取款可以避免从键盘输入提取金额，从而可以加
3
4 Scenario Outline: 固定金额取款
5   Given 我的账户中有余额"<accountBalance>"元
6   When 我选择固定金额取款方式取出"<withdrawAmount>"元
7   Then 我应该收到现金"<receivedAmount>"元
8   And 我账户的余额应该是"<remainingBalance>"元
9   Examples:
10  | accountBalance | withdrawAmount | receivedAmount | remainingBalance |
11  | 1000.00 | 100.00 | 100.00 | 900.00 |
12  | 500.00 | 500.00 | 500.00 | 0.00 |

```

4. 使用 Cucumber dry run 命令`java cucumber.api.cli.Main -d src/main/resources/features/FixedAmountWithdraw.feature`可以验证对应的 feature 文件。这个命令也会打印出 dummy 的 steps definitions，参考如下的输出内容：

图 1. Cucumber Dry Run 输出示例



```

➔ dw git:(master) X cucumber -d src/main/resources/features/FixedAmountWithdraw.feature
UUUUUUU

2 Scenarios (2 undefined)
8 Steps (8 undefined)
@n0.000s

You can implement missing steps with the snippets below:

@假如("我的账户中有余额\$(".*?")元$")
public void 我的账户中有余额_元(String arg1) throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@当("我选择固定金额取款方式取出\$(".*?")元$")
public void 我选择固定金额取款方式取出_元(String arg1) throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@那么("我应该收到现金\$(".*?")元$")
public void 我应该收到现金_元(String arg1) throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@那么("我账户的余额应该是\$(".*?")元$")
public void 我账户的余额应该是_元(String arg1) throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

```

注：命令`java cucumber.api.cli.Main`事实上要求加入对应的 jar 包到 classpath 中，为了行式 classpath。建议读者在使用时，先把对应的 jar 包下载到本地（推荐使用 maven，执行 mvn compile dependency 到本地的 m2 目录下），然后根据所使用的操作系统不同，可以按照：

- 对于 Linux 或者 OSX，可以使用 alias 创建一个 cucumber 命令，参考如下的内容：

```
alias cucumber='java -classpath ".:./target/classes:/home/admin/.m2/repository/info/cukes/cucumber-java-1.2.2.jar:/home/admin/.m2/repository/info/cukes/cucumber-core-1.2.2.jar:/home/admin/.m2/repository/info/cukes/cucumber-jvm-deps/1.0.3/cucumber-jvm-deps-1.0.3.jar:/home/admin/.m2/repository/info/cukes/gherkin/2.12.2/gherkin-2.12.2.jar:/home/admin/.m2/repository/info/cukes/cucumber-junit/1.2.2/cucumber-junit-1.2.2.jar:/home/admin/.m2/repository/junit/junit/4.12/junit-4.12.jar:/home/admin/.m2/repository/org/html/0.2.3/cucumber-html-0.2.3.jar" cucumber.api.cli.Main'
```

- 对于 Windows，可以在 PATH 指定的任意一个目录（比如 C:\Windows\System32\）下，内容可以参考如下：

```
java -classpath ".;|target\classes;C:\Users\admin\.m2\repository\info\cukes\cucumber-java-1.2.2.jar;C:\Users\admin\.m2\repository\info\cukes\cucumber-core-1.2.2.jar;C:\Users\admin\.m2\repository\info\cukes\cucumber-jvm-deps\1.0.3\cucumber-jvm-deps-1.0.3.jar;C:\Users\admin\.m2\repository\info\cukes\gherkin\2.12.2\gherkin-2.12.2.jar;C:\Users\admin\.m2\repository\info\cukes\cucumber-junit\1.2.2\cucumber-junit-1.2.2.jar;C:\Users\admin\.m2\repository\junit\junit\4.12\junit-4.12.jar;C:\Users\admin\.m2\repository\org\html\0.2.3\cucumber-html-0.2.3.jar" cucumber.api.cli.Main %*
```

生成 Step definitions

当然，通过上述的 dry run 产生的 dummy steps definitions 可以作为创建对应 Step definitions 生成 Step definitions 的前提下，我推荐大家使用 dry run 的输出作为基础，创建自己的 Step definitions。集成开发环境所提供的 Step definitions 工具来实现上述操作，其原因是：

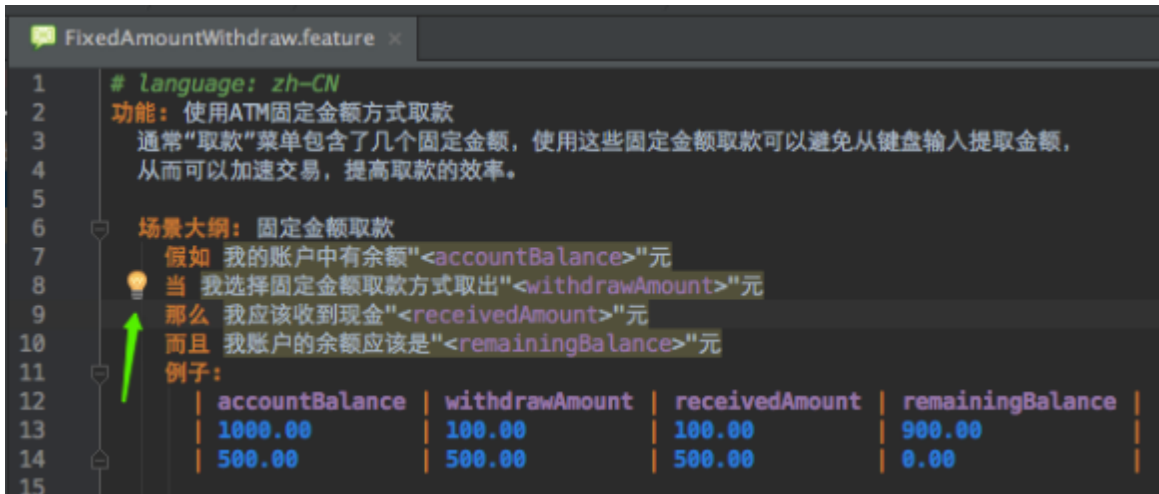
- 集成开发工具能够生成较为完善的代码，不仅仅是 steps definitions 本身，而且包含了对应的测试数据。
- 对于刚接触 Cucumber 不久且充满好奇心的使用者而言，能够接触到 Cucumber 的实现代码而集成开发环境在这一点上能够提供很大的帮助。

由于 Eclipse 暂时并不支持 Steps definitions 的生成操作，下面本文以 JetBrains 的优秀 Java IDE IntelliJ IDEA 为例，介绍如何从 feature 生成 Steps definitions。

在 IntelliJ IDEA 下生成 Steps definitions

在编写 feature 文件的过程中，IDEA 会提示目前文件中哪些步骤（steps）是没有对应的 Java 代码实现的。IntelliJ IDEA 会以黄色的小灯泡这个提示标志来提醒作者：

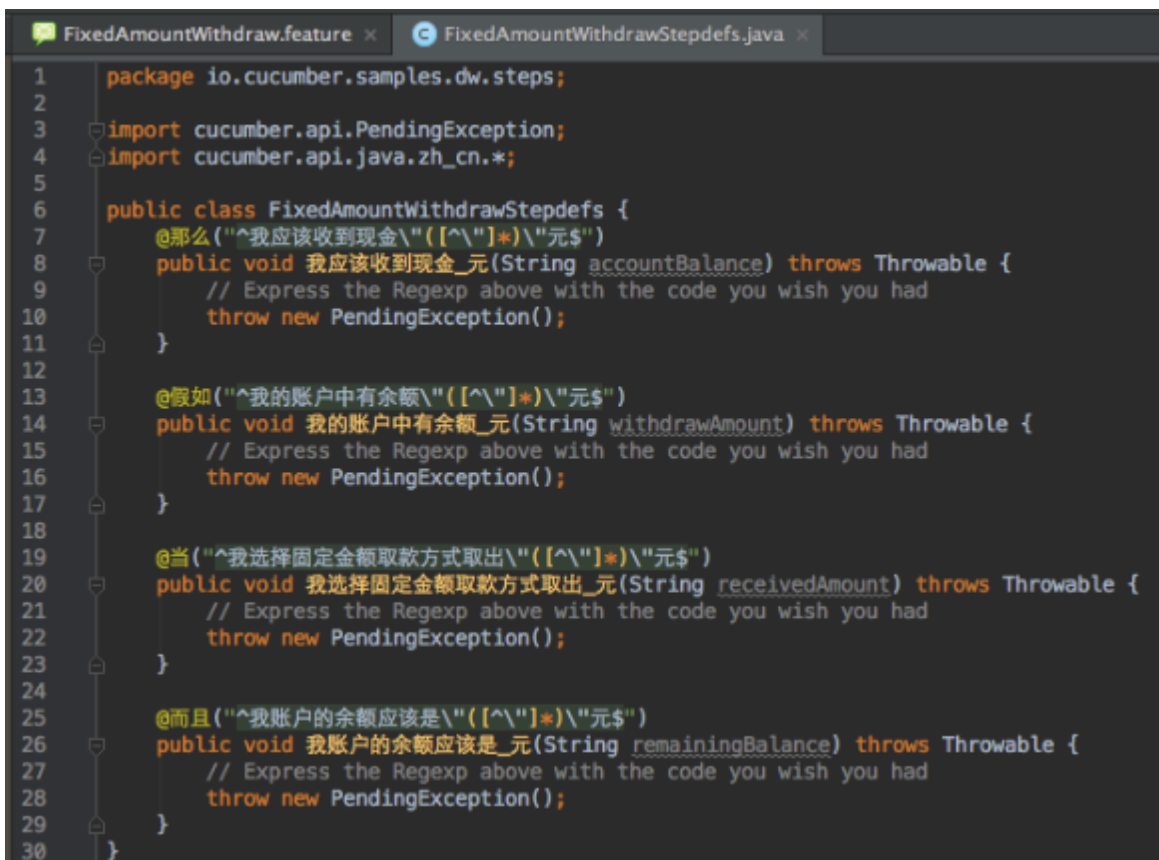
图 2. IDEA 提示 No Step Definition Found 示例



读者只需要按照如下的步骤来生成对应的 Steps definitions 即可：

1. 点击该提示图标，并从弹出的菜单项中选择“Create Step Definition”或者“Create All Steps
2. 在弹出的“Create New Step Definition File”模式窗口中填写文件名称、实现语言以及文件位
3. 重复步骤 1 和 2 直到所有的 step 都生成对应的 Java step definition 即可，参考图 3 所生成

图 3. IDEA 生成 Steps Definitions 示例



在 Eclipse 下生成 Steps definitions

在 Eclipse 中，仍然可以通过 Dry run 的方式来实现部分 steps definitions code 的生成，这一点命令。

调试和运行测试用例

在 IDEA 中调试运行测试用例

Cucumber测试用例有两种方式可以启动，debug mode下，两种方式都可以对Steps definitions代码过

a.Run with JUnit方式；

这种方式要求必须有 JUnit Test 来触发，常规的做法是：

1. 创建一个空白的 JUnit Test：所谓空白就是在 JUnit Test 类中没有任何方法；
2. 按照 Cucumber 所提供的 Options annotation 来引入所需要使用的 feature 文件和 Steps de
3. 按照需要设置报告格式等，下面是以 JUnit 方式触发 FixedAmountWithdraw.feature 文件的

清单 5. JUnit 方式触发 Cucumber Feature 文件示例

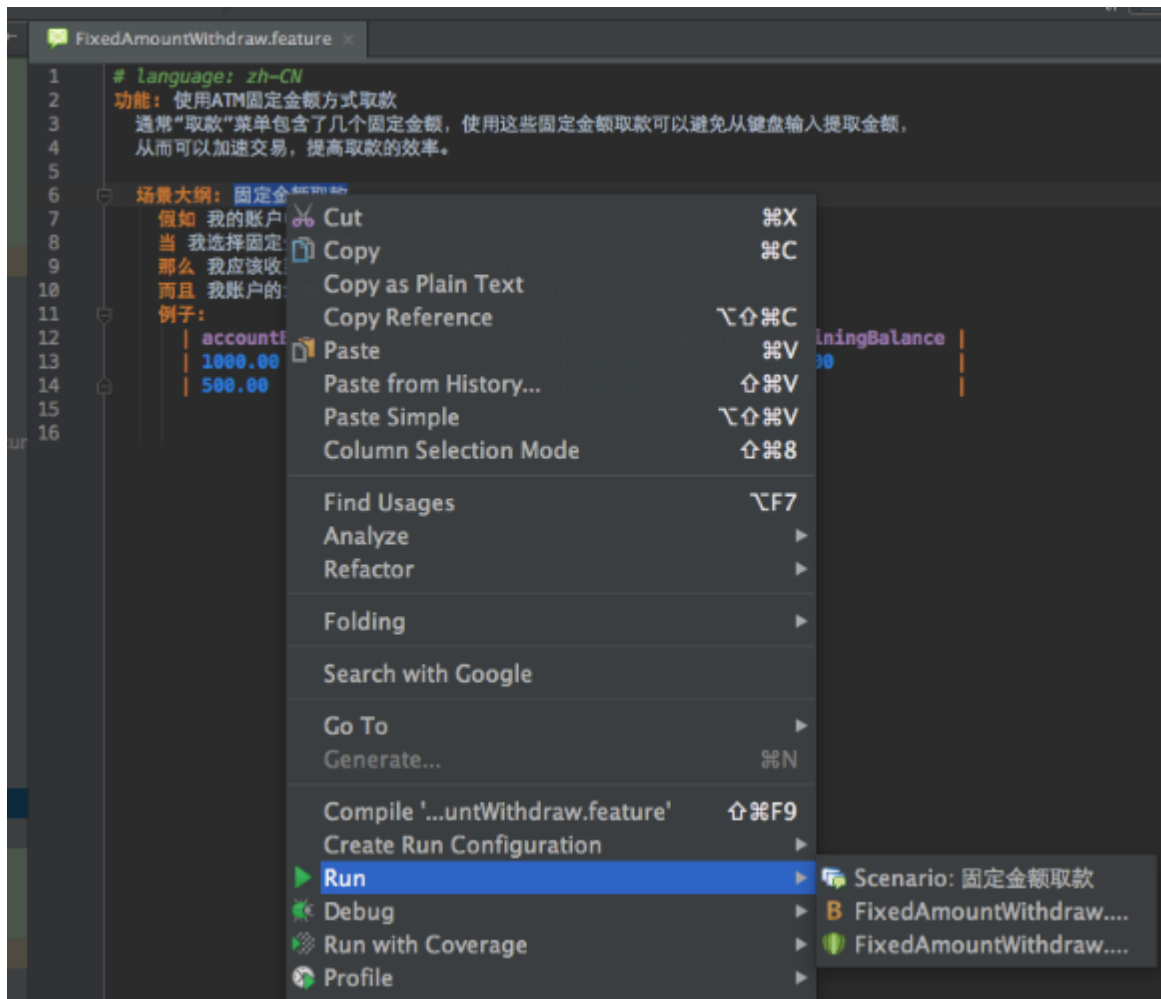
```
1 package io.cucumber.samples.dw;  
2  
3 import cucumber.api.CucumberOptions;  
4 import cucumber.api.junit.Cucumber;  
5 import org.junit.runner.RunWith;  
6  
7 @RunWith(Cucumber.class)  
8 @CucumberOptions(  
9     format = {"json:target/json-report/dw.json"}  
10    , features = {"classpath:features/FixedAmountWithdraw.feature"}  
11    , glue = {"io.cucumber.samples.dw.steps"}  
12 )  
13 public class AppTest {  
14 }
```

4. 然后就可以以 JUnit 用例的方式运行或者调试测试用例的实现了，此处不再赘述。Eclipse 和 式。

b.IDEA 下直接运行 Scenario/Scenario Outline

1. IDEA 下可以直接选中所需要运行或者调试的 Scenario/Scenario Outline；
2. 然后右键打开弹出菜单选择“Run”或者“Debug”；
3. 并选择对应的 Scenario 名称即可运行或者调试该场景对应的 Steps definitions。

图 4. IDEA 中 Debug Scenario 示例



定制化运行 Cucumber 测试用例

Cucumber 是以 feature 文件来组织测试用例的，关于如何 customized 运行 Cucumber 测试用例，组织 feature 文件。Feature 文件通常是按照子文件夹来组织的，您可以按照功能归属、类别、种类是最好的，通常并没有准确的答案，一个可行的建议是先按照您认为最可行的方式组织起来，然后随着 iteration，后续如果没有发现明显的问题就可以坚持下来。通常您需要尝试几次才能确定哪一种组织方式就像是一本书的目录对应的章节，便于索引。

Cucumber 还支持为 Scenario 指定标签（tag），tag 是以“@”字符开头的一个单词，用来表述该 scenario 是属于哪个 scenario，可以是 scenario outline 甚至可以是 scenario outline 下的 examples）所属的多个，Cucumber 本身并不限制 tag 的个数。

清单 6. 带有 Tag 的 Feature 文件示例

```

1  # language: zh-CN
2  @withdraw
3  功能：使用 ATM 固定金额方式取款
4      通常“取款”菜单包含了几个固定金额，使用这些固定金额取款可以避免从键盘输入提取金额，
5      从而可以加速交易，提高取款的效率。
6
7  @fixedAmount
8  场景大纲：固定金额取款
9      假如 我的账户中有余额"<accountBalance>"元
10     当 我选择固定金额取款方式取出"<withdrawAmount>"元
11     那么 我应该收到现金"<receivedAmount>"元
  
```



```
12 而且 我账户的余额应该是"<remainingBalance>"元
13 @positive
14 例子:
15 | accountBalance | withdrawAmount | receivedAmount | remainingBalance |
16 | 1000.00 | 100.00 | 100.00 | 900.00 |
17 | 500.00 | 500.00 | 500.00 | 0.00 |
18 @negative
19 例子:
20 | accountBalance | withdrawAmount | receivedAmount | remainingBalance |
21 | 1000.00 | 1100.00 | 0.00 | 1000.00 |
22 | 500.00 | 600.00 | 0.00 | 500.00 |
```

其中：

表 1. Tag 位置解析

Tag 所修饰对象	表述的意义
Feature	该 Feature 文件中的所有 Scenario 和 Scenario Outline 都会继承修饰在 @withdraw
Scenario / Scenario Outline	表示 tag 适用于 Scenario/Scenario Outline，Scenario Outline 下的 Examples 比如上例中的 @fixedAmount
Examples	只对 Scenario Outline 及当前的 Examples 有效，比如上例中的 @positive

如果说子文件夹是目录对应的章节，那么 tag 可以看做是添加到书中的便签，让您迅速找到对应另外一种维度的索引，可以为 Cucumber 测试用例集合添加不同的维度，从而便于检索和过滤，测试用例时显得尤其重要！

按照 tag expression 过滤测试用例

纯粹的讲 expression 本身会让读者觉得味同嚼蜡，下面本文以不同的例子为基础，讲述如何适用 Cucumber 测试用例。

- 运行指定的单个 tag 对应的测试用例：

命令：`java cucumber.api.cli.Main --tag @focus features` 只会运行 features 中那些被标记为一些或者某一个场景其实在场景调试过程中非常有用，尤其是在您没有可用的 IDE 环境中，比如 Unix/Linux 机器上调试测试用例时；

- 运行 @focus 或者 @key 对应的测试用例

命令：`java cucumber.api.cli.Main --tag @focus,@key features` 可以运行 features 中那些被标记为“@key”的场景。此处的逗号可以被理解为逻辑上的“OR”关系运算符，因此运行的结果是二者都包含的场景。

- 运行被 @fast 标记并且 @bvt 标记的测试用例

命令：`java cucumber.api.cli.Main --tag @fast --tags @bvt features` 可以运行 features 中那些被 @fast 和 @bvt 标记的测试用例。命令可以从字面上理解为运行 BVT 测试用例中那些运行速度快的测试用例，假设您一下所做的改动是否影响到主流程，上述命令会帮您快速运行 BVT 相关的用例，当然，前提是您的改动只影响到了 BVT 相关的用例。

- 不运行被 @slow 标记但是被 @bvt 标记的测试用例

命令：`java cucumber.api.cli.Main --tag ~@slow --tags @bvt features` 可以运行 features 中那些没有被 @slow 修饰标记的测试用例。理想情况下，我们期望测试用例运行的足够快，但是现实测试中，由于某些原因，不乏有运行慢的用例，因此，标记出那些运行速度慢的用例，然后只在适合的时间运行这些用例，效率也是非常必要的。此处的“~”表示否定的意思。

如果您担心用例中有同样的标记比如 @focus 也在被别人使用，不希望通过 tag 的方式来标记用例，那么您可以通过 Filter on lines 和 Filter on names 的方式来实现同样的功能：Filter on lines 和 Filter on names。

按照行号过滤测试用例

命令：`java cucumber.api.cli.Main src/main/resources/features/FixedAmountWithdraw.feature --lines 12-17` 可以运行 FixedAmountWithdraw.feature 文件中的第 12 和 17 行，如果需要运行更多的行，只需要在 --lines 后面指定行号范围即可。

按照场景名称过滤测试用例

命令：`java cucumber.api.cli.Main --name 固定金额取款 features` 可以运行名称为“固定金额取款”的场景。对于 debug 单个场景时，这个功能非常有用。

指定 Cucumber 运行结果报告

Cucumber 本身支持多种报告格式以适用于不同环境下调用的报告输出：

- pretty：用于在命令行环境下执行 Cucumber 测试用例所产生的报告，如果您的 console 支持颜色，那么报告将使用颜色来显示不同的运行结果；如下图所示的例子分别显示了用例执行通过和用例没有通过的情况。

图 5. Pretty 格式的 Passed 报告示例

```

➔ dw git:(master) X cucumber -p pretty src/main/resources/features/FixedAmountWithdraw.feature:17 -g io.cucumber.samples.dw.steps
# language: zh-CN
@withdraw
功能：使用ATM固定金额方式取款
通常“取款”菜单包含了几个固定金额，使用这些固定金额取款可以避免从键盘输入提取金额，
从而可以加速交易，提高取款的效率。

@FixedAmount
场景大纲：固定金额取款                                # src/main/resources/features/FixedAmountWithdraw.feature:8
假如我的账户中有余额"<accountBalance>"元
当我选择固定金额取款方式取出"<withdrawAmount>"元
那么我应该收到现金"<receivedAmount>"元
而且我账户的余额应该是"<remainingBalance>"元

例子：

@withdraw @FixedAmount
场景大纲：固定金额取款                                # src/main/resources/features/FixedAmountWithdraw.feature:17
假如我的账户中有余额"500.00"元                        # FixedAmountWithdrawStepdefs.我的账户中有余额_元(String)
当我选择固定金额取款方式取出"500.00"元                # FixedAmountWithdrawStepdefs.我选择固定金额取款方式取出_元(String)
那么我应该收到现金"500.00"元                            # FixedAmountWithdrawStepdefs.我应该收到现金_元(String)
而且我账户的余额应该是"0.00"元                        # FixedAmountWithdrawStepdefs.我账户的余额应该是_元(String)

1 Scenarios (1 passed)
4 Steps (4 passed)
0m0.148s

```

图 6. Pretty 格式的 Undefined 报告示例

```

➔ dw git:(master) X cucumber -p pretty src/main/resources/features/FixedAmountWithdraw.feature:17
# language: zh-CN
@withdraw
功能：使用ATM固定金额方式取款
通常“取款”菜单包含了几个固定金额，使用这些固定金额取款可以避免从键盘输入提取金额，
从而可以加速交易，提高取款的效率。

@FixedAmount
场景大纲：固定金额取款                                # src/main/resources/features/FixedAmountWithdraw.feature:8
假如我的账户中有余额"<accountBalance>"元
当我选择固定金额取款方式取出"<withdrawAmount>"元
那么我应该收到现金"<receivedAmount>"元
而且我账户的余额应该是"<remainingBalance>"元

例子：

@withdraw @FixedAmount
场景大纲：固定金额取款                                # src/main/resources/features/FixedAmountWithdraw.feature:17
假如我的账户中有余额"500.00"元
当我选择固定金额取款方式取出"500.00"元
那么我应该收到现金"500.00"元
而且我账户的余额应该是"0.00"元

1 Scenarios (1 undefined)
4 Steps (4 undefined)
0m0.000s

```

- json：多用于在持续集成环境下的跨机器生成报告时使用，比如在用例执行的机器 A 上运行或报告机器 B 上生成用例执行报告，此时只需要把生成的 JSON 报告传输到机器 B 上即可。

图 7. JSON 格式报告示例

```

➔ dw git:(master) x cucumber -p json src/main/resources/features/FixedAmountWithdraw.feature:17 -g io.cucumber.samples.dw.steps
[
  {
    "id": "使用atm固定金额方式取款",
    "tags": [
      {
        "name": "@withdraw",
        "line": 2
      }
    ],
    "description": "通常“取款”菜单包含了几个固定金额，使用这些固定金额取款可以避免从键盘输入提取金额，\n从而可以加速交易，提高取款的效率。",
    "name": "使用ATM固定金额方式取款",
    "keyword": "功能",
    "line": 3,
    "elements": [
      {
        "id": "使用atm固定金额方式取款;固定金额取款;;3",
        "tags": [
          {
            "name": "@withdraw",
            "line": 2
          },
          {
            "name": "@fixedAmount",
            "line": 7
          }
        ],
        "description": "",
        "name": "固定金额取款",
        "keyword": "场景大纲",
        "line": 17,
        "steps": [
          {
            "result": {
              "duration": 176930000,
              "status": "passed"
            },
            "name": "我的账户中有余额'500.00'元",
            "keyword": "假知",
            "line": 9,
            "match": {
              "arguments": [
                {
                  "val": "500.00",
                  "offset": 9
                }
              ],
              "location": "FixedAmountWithdrawStepdefs.我的账户中有余额_元(String)"
            },
            "textedColumns": [

```

- html：用于生成简单的 HTML 格式的报告以便查看 Cucumber 测试用例运行的结果

图 8. 简单的 HTML 格式报告示例

```

▼
# language: zh-CN
@withdraw 功能: 使用ATM固定金额方式取款
通常“取款”菜单包含了几个固定金额，使用这些固定金额取款可以避免从键盘输入提取金额，从而可以加速交易，提高取款的效率。
▶ @fixedAmount 场景大纲: 固定金额取款
▼ @withdraw @fixedAmount 场景大纲: 固定金额取款
假如我的账户中有余额'1000.00'元
当我选择固定金额取款方式取出'100.00'元
那么我应该收到现金'100.00'元
而且我账户的余额应该是'900.00'元
▼ @withdraw @fixedAmount 场景大纲: 固定金额取款
假如我的账户中有余额'500.00'元
当我选择固定金额取款方式取出'500.00'元
那么我应该收到现金'500.00'元
而且我账户的余额应该是'0.00'元
▼ 例子:


| accountBalance | withdrawAmount | receivedAmount | remainingBalance |
|----------------|----------------|----------------|------------------|
| 1000.00        | 1100.00        | 0.00           | 1000.00          |
| 500.00         | 600.00         | 0.00           | 500.00           |


▼ @withdraw @fixedAmount 场景大纲: 固定金额取款
假如我的账户中有余额'1000.00'元
当我选择固定金额取款方式取出'1100.00'元
那么我应该收到现金'0.00'元
而且我账户的余额应该是'1000.00'元
▼ @withdraw @fixedAmount 场景大纲: 固定金额取款
假如我的账户中有余额'500.00'元

```

- junit：用于生成 JUnit 格式的报告：

清单 7. JUnit 格式报告示例

```

1 <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2 <testsuite failures="0" name="cucumber.runtime.formatter.JUnitFormatter"
3   skipped="0" tests="1" time="0.177755">

```

```

4   <testcase classname="使用 ATM 固定金额方式取款" name="固定金额取款" time="0.17775
5   <system-out>
6   <![CDATA[
7   假如我的账户中有余额"500.00"元.....passed
8   当我选择固定金额取款方式取出"500.00"元.....passed
9   那么我应该收到现金"500.00"元.....passed
10  而且我账户的余额应该是"0.00"元.....passed
11  ]]>
12  </system-out>
13  </testcase>
14  </testsuite>

```

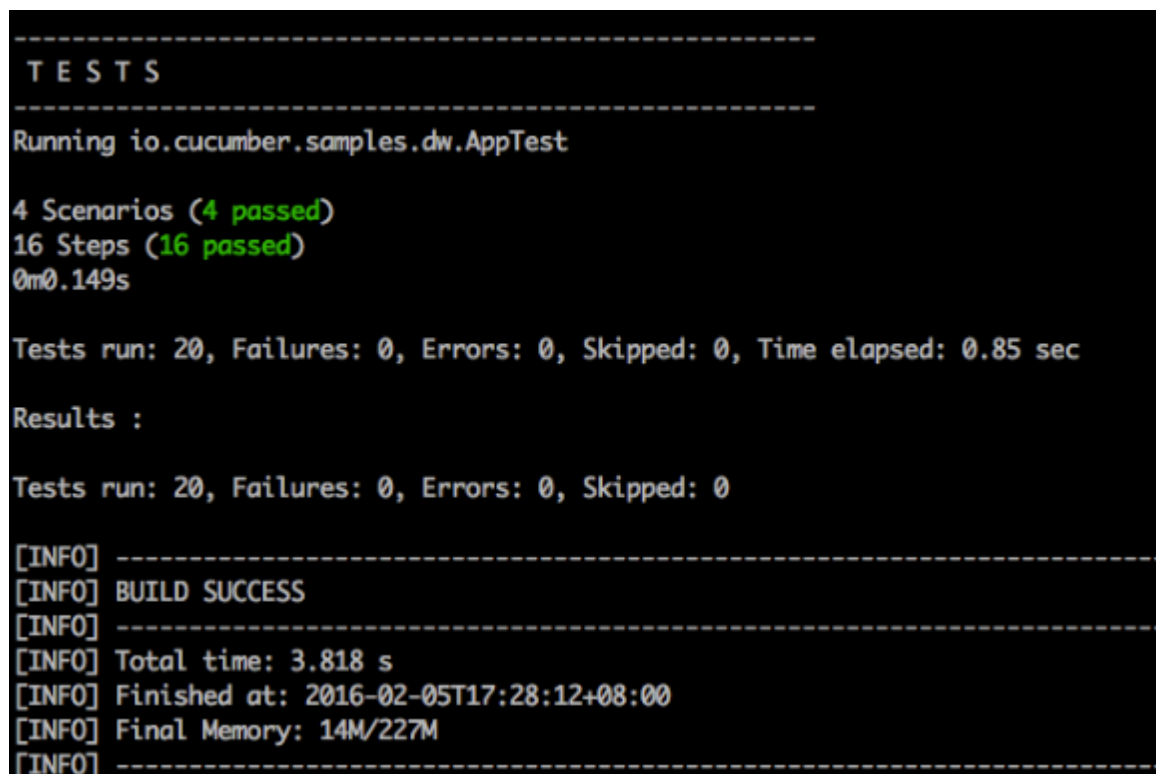
如何避免 Cucumber 测试用例报告失真

从上面描述的 Cucumber 支持的报告格式和样例，看不出有什么失真啊？错了就错了，对了就告会失真？

这里说的测试报告失真是以 Maven 调用的方式运行 Cucumber 测试用例时才会出现，但是这种中经常会用到的，因此，此处才对这种情况加以处理，以避免在持续集成环境下出现测试报告：

一般情况下我们在 Maven 中执行测试用例只需要运行命令`mvn test`即可，下面是一次运行中

图 9. Maven 给出的失真的测试报告示例



```

-----
TESTS
-----
Running io.cucumber.samples.dw.AppTest

4 Scenarios (4 passed)
16 Steps (16 passed)
0m0.149s

Tests run: 20, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.85 sec

Results :

Tests run: 20, Failures: 0, Errors: 0, Skipped: 0

[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 3.818 s
[INFO] Finished at: 2016-02-05T17:28:12+08:00
[INFO] Final Memory: 14M/227M
[INFO] -----

```

从中可以看出：

- Cucumber 提示运行了 4 个 Scenario 一共 16 个 Step 且全部通过了；
- 可以理解成一共执行了 4 个 JUnit 测试用例，没有失败或编译错误；

- 但是 Maven 给出的报告却提示运行了 20 个 Test，全部通过。

这就造成了运行报告的失真：

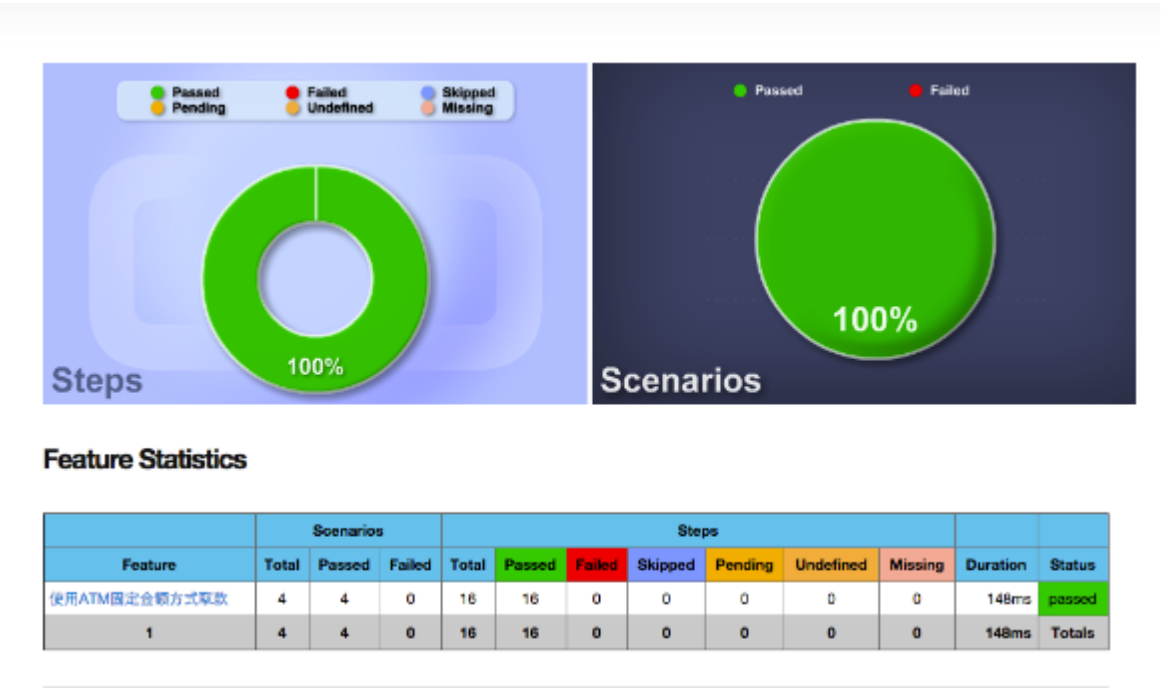
1. 事实上没有那么多测试用例，但是 Maven 却给出了数倍于真实的测试用例；
2. 如果有一个失败，其失败率明显会低于真实情况；
3. 如果以此为依据来 refactor 代码，很可能造成过于自信而导致后续工作量无法承受。

鉴于此，必须要避免这样的情况，有什么样的方法可以避免吗？

当然有，不知读者是否还记得 Cucumber report 支持 json 呢？Github 上有很多开源的插件或者 JSON 格式的报告生成 HTML 格式的报告。本文中推荐大家使用 [Cucumber-reporting](#)。Cucumber JSON 格式报告生成 HTML 格式报告，而且可以按照 tag 和 feature 以及 step 查看，不得不提的非常好看，下面就是以本文中所使用的 feature 文件为例，以 [Cucumber-reporting](#) 来生成的 H

按照 Features 方式查看

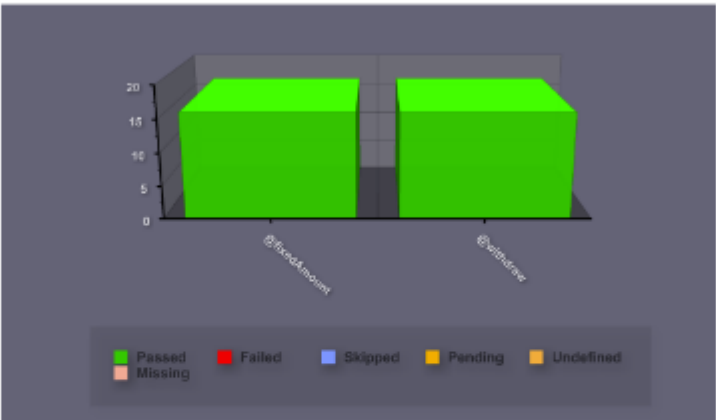
图 10. 按照 Features 方式查看报告示例



按照 Tags 方式查看

图 11. 按照 Tags 方式查看报告示例

The following graph shows passing and failing statistics for tags in this build.



Tag Statistics

Tag	Scenarios			Steps							Duration	Status
	Total	Passed	Failed	Total	Passed	Failed	Skipped	Pending	Undefined	Missing		
@fixedAmount	4	4	0	16	16	0	0	0	0	0	148ms	passed
@withdraw	4	4	0	16	16	0	0	0	0	0	148ms	passed
2	8	8	0	32	32	0	0	0	0	0	297ms	Totals

按照 Steps 方式查看

图 12. 按照 Steps 方式查看报告示例

The steps statistics in this build.

Step Statistics

Implementation	Total	Duration (ms)	Duration	Status
FixedAmountWithdrawStepdefs.我的账户中有余额_元(String)	4	147728000	147ms	100.0%
FixedAmountWithdrawStepdefs.我选择固定金额取款方式取出_元(String)	4	352000	000ms	100.0%
FixedAmountWithdrawStepdefs.我应该收到现金_元(String)	4	294000	000ms	100.0%
FixedAmountWithdrawStepdefs.我账户的余额应该是_元(String)	4	251000	000ms	100.0%
all steps	16	148636000	148ms	Totals

从上述报告中可以看出，Scenario 和 Step 是分别统计的，因此只需要关注 Scenario 失败的信息。

如何与持续集成工具集成

主流的持续集成工具有很多，被广泛采用的开源工具当推 Jenkins。Cucumber reporting 功能也有开源的 Jenkins Plugin：[Publish pretty cucumber-jvm reports on Jenkins](#)。对于其具体的用户尝试过可以按照其步骤成功集成 Cucumber Reporting 功能到 Jenkins，此处不再赘述。

结束语

本文从实际使用 Cucumber 这一工具的角度出发，以 Cucumber-JVM 实现为基础，采用了不同件，如何从 feature 文件生成对应的 Steps，如何生成不同格式的报告，如何定制化的运行测试中如何避免报告失真、如何与主流持续集成工具结合使用等，为大家在日常工作中使用 Cucum为自动化工具的层面，其在 API 测试中的使用也有很多可圈可点之处，本系列后续文章会对此

- 《活用 Cucumber 测试服务端开放 API》，讲述结合 BDD 风格测试工具 Rest-Assured 验证 Schema 验证 API 的返回结构等一系列实用的经验分享。

下载资源

[!\[\]\(e3f8612927870f2e0f9f5989e6dd3064_img.jpg\) 本文相关源代码 \(dw_art1.tar | 245 KB\)](#)

相关主题

- [The Cucumber Book: Behaviour-Driven Development for Testers and Developers \(Pragmatic\)](#) Cucumber 必读书目，Ruby 语言描述。
- [The Cucumber for Java Book: Behaviour-Driven Development for Testers and Developers](#) Java 语言描述。
- [Cucumber documents](#)，了解 Cucumber 不同语言的实现。
- [Cucumber-JVM](#)，使用 Cucumber 在 Java 中的实现。
- [Cucumber Reporting](#)，学习如何从 Cucumber JSON 格式报告生成样式好看的 HTML 格式。
- [developerWorks Java 技术专区](#)：这里有数百篇关于 Java 编程各个方面的文章。

评论

添加或订阅评论，请先[登录](#)或[注册](#)。

☐ 有新评论时提醒我

developerWorks

站点反馈

我要投稿

投稿指南

报告滥用

第三方提示

关注微博

加入

ISV 资源 (英语)

选择语言

English

中文

日本語

Русский

Português (Brasil)

Español

한글

技术文档库

订阅源

社区

dW 中国时事通讯

软件下载

[联系 IBM](#) [隐私条约](#) [使用条款](#) [信息无障碍选项](#) [反馈](#) [Cookie 首选项](#)