arXiv:2303.14104v1 [cs.DC] 24 Mar 2023

JepREST：分布式 REST 应用的功能测试

Sara Sim˜oes、Ana Ribeiro、Carla Ferreira 和 Nuno Pregui.ca

里斯本新大学科学与技术学院

摘要 移动和 Web 应用通常由带有 REST 接口的应用服务提供支持，这些服务由一组彼此交互的分布式组件实现。这种方法使服务能够以低于单体系统的成本实现高可用性和高性能。然而，多个组件的存在使此类系统的开发过程更加复杂，因此更容易出现错误。本文介绍 JepREST，这是一个能够自动使用 Jepsen 库，测试提供 REST 接口的分布式应用正确性的系统。JepREST 根据服务接口规范生成并执行测试集，由多个客户端并发执行操作，随后检查系统行为是否可线性化。初步评估表明，JepREST 能够简化 REST 应用的测试。

1 引言

移动和 Web 应用通常通过支持性的应用服务来实现，这些服务执行应用逻辑并维护其状态，同时提供 REST 接口，供客户端执行应用操作。对于非平凡应用，这些应用服务是复杂的分布式系统，由多个组件组成，包括应用缓存服务（例如 Memcached 、 Redis ）、事件传播服务（例如 Kafka ）、数据库以及复制数据存储服务。

这些系统的复杂性源于其目标：实现高性能、高可用性和容错能力。因此，这些系统也变得难以设计和编程，容易出现错误。因此，应用系统的验证和测试是一项重要问题，而复杂分布式应用的测试一直受到研究界和工业界的关注。

有许多工具可用于测试 REST 服务，尤其是允许执行负载测试、评估 REST 服务性能的工具。一些此类工具（例如 Artillery [4]）支持进行功能测试，以检查服务是否按预期运行。然而，这些工具对功能测试的支持有限，且

复杂。首先，工具的使用者必须指定某项操作执行后的预期结果。对于简单测试，这样做是可行的；但对于随机测试以及涉及并发执行操作的测试，这就很困难，甚至是不可能的。其次，这些工具没有能够干扰服务组件运行的机制，因此无法测试系统在发生故障时的行为。因此，人们日益认识到，有必要创建能够简化分布式应用测试创建过程的应用程序 [16]。

本文介绍 JepREST，这是一个用于简化分布式应用测试的系统。此类应用具有 REST 接口，JepREST 旨在分析 REST 应用在接受功能测试集测试时的正确性；测试过程中有多个客户端执行并发请求，应用中受测的各个组件也可能发生故障。

JepREST 由四个组件构成。第一个组件根据 REST 服务规范生成用于调用 REST 服务操作的代码。第二个组件生成用于执行程序员定义的工作负载的代码，其中包括在应用中创建新对象的代码。第三个组件执行测试，并控制 REST 应用和服务测试客户端的运行。最后一个组件负责验证应用的执行是否符合正确行为；正确行为定义为执行具有线性化性，即验证执行期间所有成功完成的操作是否都按照实时顺序以原子方式发生。例如，若操作 A 在操作 B 开始之前完成，那么 A 的执行必须始终先于 B 的执行；换言之，B 的执行必须始终观察到 A 的执行所产生的效果。JepREST 可以看作一个系统，用于简化在 REST 应用测试中使用 Jepsen 的过程。Jepsen 是一个用于分析分布式系统正确性的库。在设计 JepREST 时，需要应对多个挑战，包括如何根据 REST 接口规范自动生成测试执行所需的代码（包括创建新对象），以及如何定义验证执行正确性所需的模型。

JepREST´仍在开发中，目前版本˜仅支持测试在单个 ´服务器上运行的应用程序，且操作遵循 REST 的标准 语义。初步实验表明，该系统能够检测出大量正确性违规。

本文其余部分安排如下：第 2 节讨论相关工作；第 3 节介绍 JepREST 使用的库；第 4 节描述该系统；第 5 节介绍所进行的测试；第 6 节总结全文。

2 相关工作

分布式系统的正确性验证是一个复杂的过程，原因有多种，其中包括系统的去中心化运行方式——

系统的各个组件独立执行操作，不同组件之间以及每个组件内部执行的操作存在并发性，而且通信和组件本身都可能发生部分故障。在软件 测试中，可以将验证分解为一系列阶段 [24]。单元测试旨在单独验证每个组件的正确性。集成测试旨在验证多个分布式组件集成后的正确性，其中的组件可以运行在同一台物理机器上，也可以运行在不同的机器上。功能测试旨在验证整个系统是否正常运行，即是否符合规范。

要使分布式系统的功能测试具有意义，必须考虑多个因素。一方面，应模拟符合预期使用场景及系统使用边界情况的工作负载。另一方面，必须考虑为系统定义的故障模型，即系统应能够容忍哪些故障，并通过模拟这些故障来验证系统确实能够容忍它们。

许多研究探讨了在分布式系统中引入故障的问题，包括在组件和通信中注入故障。这些研究与我们的工作互为补充，可用于在测试执行期间模拟故障。

此外，也存在多种系统用于执行负载测试 [19]„，可以评估延迟、性能和可扩展性等非功能属性，以及分布式应用程序。

REST API 测试 有多种工具和库可用于测试使用 REST API 的服务。cURL [11] 提供了一个库，可以通过 HTTP 发起调用，也可用于构建测试应用程序。Postman [15] 和 SoapUI [12] 等应用程序可以创建脚本 来测试 REST API，定义需要调用的操作顺序以及预期结果。不过，这些工具都无法执行并发请求。

Dredd [14]允许根据 API 描述生成测试，描述格式可以是*API Blueprint* 或*Swagger*，从而简化测试流程。然而，结果分析仅限于确认结果是否符合预期格式，既不处理可能存在的并发请求，也不验证响应内容是否正确。

ReqBin [13] ´是一款在线 工具，可用于测试 REST 服务，并支持从多个地理位置执行并发请求。不过，它不支持分析所得结果是否正确。

有多种工具可对 REST 服务进行负载测试。在这类应用中，程序员会指定要执行的测试，列出应执行的操作序列及各序列的相对频率。工具会执行测试，模拟多个并发客户端，执行所定义的各种操作。通常，程序员可以定义一次调用之后的预期结果，

但在结果取决于先前执行的操作，且可能受到并发操作影响的情况下，这项功能很难使用。我们的工作与这些工具有多方面的不同，尤其是支持验证结果的正确性，这通过 Knossos 库实现；此外，还可以在应用的各个组件中引入多种故障。

其他研究探讨了对现有 Web 服务,及数据库 [27]的正确性进行验证。与这些研究不同，我们的工作侧重于测试提供 REST 接口的通用分布式系统。

3 背景

本节介绍开展这项工作的背景，包括构建 JepREST 时使用的库。

OpenAPI 规范。˜ (OAS) [10] 定义了一种与开发 RESTful Web 服务所用语言无关的服务规范。通过一组注解，该规范可以指明服务提供的操作、这些操作的参数和结果，包括所处理对象的模式、不同对象属性之间的关联，以及其他信息。

有一些工具可以将 OpenAPI 规范转换为不同语言中使用的规范，例如 Java 中使用的 JAX-RS，反之亦然，从而简化了根据现有实现生成 OpenAPI 规范的过程。

PETIT PETIT 是一种微服务测试工具，它使用扩展后的 OpenAPI 规范，支持针对数据定义不变量，以及使用一阶逻辑为操作执行定义前置条件和后置条件。除了向服务的 API 发送请求并评估获得的结果外，PETIT 还可根据规范文档中定义的模式，生成适用于测试的数据。

Jepsen Jepsen [8] ´是一个用 Clojure 编写的库，用于创建测试分布式系统（尤其是数据库）正确性的程序。使用 Jepsen ´测试分布式数据库时，会在 Docker 环境中进行：测试程序在一个容器 中运行，数据库则在其他多个容器中运行。

测试程序会并发执行多个数据库客户端，这些客户端按照线程 指定的操作序列执行，而该线程负责协调测试。在测试期间，可以向系统注入一组故障；操作集和故障集均由程序员定义。

执行测试期间,会˜存储 ´执行´历史记录 [6] ，其中包含已执行的,操作˜及其结果。Jepsen 提供了两种工具：Knossos 和 Elle [20], ，用于验证执行的正确性,˜。Knossos 根据系统模型检查历史记录´ ´是否可线性化；系统模型包括对操作行为和系统状态的定义,˜。

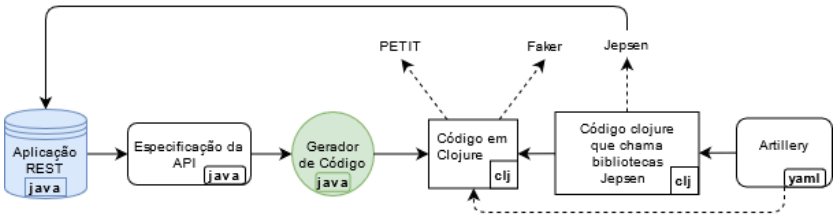

图 1. 使用 JepREST 测试应用的步骤。

Knossos 逐个分析操作，而 Elle 则能将历史记录正确性的验证范围扩展˛˜至由多个操作˛˜组成的事务的执行。

Faker Faker 是一个库，可用于生成逼真的数据，以便大规模测试采用 REST 接口的应用。使用该库的优势在于它能够生成相关数据，也就是说，生成类似于真实用户会提交的数据。例如，如果应用需要一个用户名，Faker 库就会生成一个真实的名字，即一个真实用户可能使用的名字。

4 JepREST

JepREST 是一个用于验证采用 REST 接口的应用正确性的系统，验证工作在提交功能测试后进行。为此，JepREST 系统会使用第 3 节介绍的 Jepsen 库，生成并提交一个测试集，由多个客户端并发地对待分析的 REST 应用执行操作。随后，JepREST 会分析应用生成的响应，以验证其是否符合正确的行为。

使用 JepREST 测试应用程序的步骤如下，图 1 展示了这一流程：

1. 定义所研究应用的 API，以及与各项操作相关的语义。根据 API 生成调用服务方法所需的代码，包括创建新对象。2. 定义测试系统要执行的测试集。JepRest 会生成代码，以执行 YAML脚本 中定义的工作负载。3. 使用此前生成的代码执行工作负载，并借助 Jepsen 的库来控制测试执行并获取执行日志 。4. 分析所得结果，检查执行是否符合正确行为；正确行为定义为根据所定义的 API 模型可线性化的执行。接下来将详细介绍该方案的每个步骤。

4.1 应用程序 API 的定义

由于 JepREST 需要使用 Jepsen 提供的方法和结构，而这些内容是用 Clojure 开发的，因此需要创建一个代码生成器，用于处理应用规范并生成由 Jepsen 执行的 Clojure 方法，以便对所研究的 REST 应用发起必要的调用。REST 服务规范应由程序员以 Java 代码提供，并使用 OpenAPI 和 JAX-RS 注解。

Jepsen 库提供了可执行简单读写操作的客户端，但不包含任何能够提交 REST 操作的客户端。因此，有必要创建一个新的 Jepsen 客户端，并为每种 REST 操作定义客户端的行为，以便执行相应的操作请求。对于 API 中定义的每种操作，都通过生成的 Clojure 方法来定义其行为；客户端提交相应操作的请求时，该方法便会执行。生成这些方法时，假设所研究的应用程序采用标准REST 语义：POST 用于创建新资源；GET 用于读取一个或多个资源；PUT 用于更新特定资源；PATCH 用于部分更新现有资源；最后，DELETE 用于删除特定资源。

4.2 测试集的定义

要确定 REST 应用程序是否正确，必须分析系统执行有代表性的测试集时给出的响应。这个过程中有两个挑战。首先，要确保调用操作时使用的参数具有相关性。其次，要定义适用于该系统的工作负载。

关于第一个挑战，如果定义的操作集只包含针对从未创建过的资源发出的 GET、PUT、PATCH 或 DELETE 操作，那么应用程序生成的结果将无法充分反映所研究应用程序的实现情况。为了解决这个问题，生成的代码会在客户端保存服务器上已创建对象的信息，并使用 OpenAPI 的 @Link 注释建立不同对象之间的关联。这些注释定义了不同操作之间的依赖关系。例如，若操作 A 带有指向操作 B 的 @Link 注释，如图2所示，则认为操作 B 依赖于操作 A，因为要成功执行操作 B，就需要将操作 A 的信息作为参数传入。

在处理每个操作的 JAX-RS 注解、生成用于描述客户端希望将该操作提交给应用程序时其行为的 Clojure 方法后，代码生成器将分析并处理应用程序规范中所有现有的 @Link 注解，以检测操作之间的所有依赖关系，并将特定操作的所有信息存储到列表中，供应用程序的其他操作用作输入 。接下来，生成器

```
@POST
@Operation(operationId = "创建学生", 响应 = {
    @ApiResponse(链接 = {@Link(名称 ="GetStudentByID",
    操作 ID = "获取学生",
    参数 = {@LinkParameter(名称 ="学生标识",
                表达式 = "$响应.body#/标识符")})})})
响应 创建学生(学生 student);
```

图2. 创建的 @Link 注解。˛˜

```
    @Values("faker.name/first –name")  @Pattern(正则表达式 ="[A–Z][a–z]+")
    private String 名字;
    @Min(0) @Max(100)
    private int 年龄;
    @Pattern（正则表达式 ="[A–Z][a–z]+") @Size(min = 20, max = 500)
    private 字符串 说明;
```

图3. 表示数据属性的注解示例。

负责生成 Clojure 方法，以定义对 REST 应用程序进行的调用。为此，生成器将利用创建的列表，选择依赖于其他操作的操作的输入 值，例如参数。

另一个与定义带相关参数的调用有关的方面，是生成新对象，例如学生信息。为应对这一挑战，JepREST 使用 Faker 和 PETIT 库自动生成数据。程序员应在 API 规范中定义如何生成不同的数据，并使用不同类型的注解；图中展示了数据如何指定的示例。如果存在特殊注解 @Values，代码生成器会指定使用 Faker 工具生成该数据，并调用注解中描述的方法。如果使用了 @Min、@Max、@Size 或 @Pattern 注解，而没有使用 @Values 注解，代码生成器则会指定使用 PETIT 工具生成该数据，并将注解中的值作为参数传递给即将调用的 PETIT 方法。

因此，只要˙可能，就会始终使用 Faker 工具，因为它能够生成真实的数据，这些数据可能由真实用户在使用˛应用˛时提供。

第二个挑战在于定义测试过程中要提交给应用程序的工作负载。在 JepREST 中，工作负载在一个受 Artillery 工具启发的 YAML 脚本 中定义，其中指定了不同的测试场景。每个场景由一系列操作组成，这些操作在 ***flow*** 标签中定义（使用上一步生成的 Clojure 方法名称），并由多个客户端并发执行，每个场景都有特定的权重。权重有助于确定哪个场景应执行更多次。例如，如果在一个文件中

[1] ˙值得˛注意的是，与 Artillery 测试 *script* 不同，后者会详细说明 REST 操作的调用方式（包括 HTTP 方法、参数及其编码等），而 JepREST 使用的 *script* 只会指出操作名称。

YAML 文件中，场景 1 的权重为 90，场景 2 的权重为 10，那么第一个场景中定义的操作序列将比第二个场景中定义的操作序列频繁执行 9 倍。

4.3 测试执行

在 JepREST 向正在研究的应用提交测试之前，程序员需要定义现有 REST 服务器运行所在容器的位置，即 IP 地址和端口，以便客户端知道可以向何处提交请求。

假设每次执行测试时，应用都会初始化；JepREST 不支持验证已在运行且已处理过 REST 请求的应用。因此，执行测试之前，服务会在 Docker 环境中初始化并启动——待测应用运行在一个或多个 Docker 容器中。JepREST 支持验证这些容器在本地或远程运行的情况，只需提供它们的位置即可。JepREST 执行测试时，负责生成并向 Jepsen 发送已定义的操作序列。随后，Jepsen 的控制容器会创建多个客户端，并告知它们应向应用提交哪些操作，以及请求所需的全部信息，包括容器的位置，即要向其发送请求的服务器所在的位置。

该组件中仍在实现的一些功能包括定义要模拟的故障集合，以及支持同一应用程序使用多个不同的 REST 服务。

4.4 结果验证

测试结束后，JepREST 会分析收到的响应，以验证这些响应是否符合正确的运行结果。Jepsen 在执行各种操作期间，会将执行历史信息自动存储在生成的文件中。

JepREST 使用 Jepsen 提供的 $checker$Knossos 分析已执行操作的历史记录。该 $checker$ 负责根据表示状态及其在操作发生时应如何变化的模型，验证操作历史是否具有线性一致性。

为了使用 Knossos，JepREST 定义并实现了应用程序状态模型，其中表达了在特定 REST 操作的作用下该状态如何发生变化。

模型的实现分为两个阶段：第一阶段选择一种数据类型来表示所研究的 REST 应用程序的状态；第二阶段确定执行特定 REST 操作后应用程序状态应如何变化。在第一阶段，定义用映射来表示 REST 应用程序的状态，通过

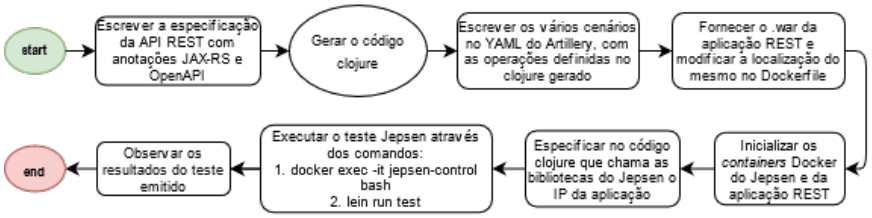

图4. 用户执行 JepREST 的操作。

这种映射的键是资源名称，值则是其他映射；这些映射的键是已创建资源的标识符，值是表示现有资源的 $json$ 对象。

第二阶段 ′定义应用程序的新状态，这是应用›多种已定义 REST 操作的结果。例如，如果对象的标识符已存在，POST 操作会返回错误；如果不存在，则使用接收到的对象更新应用程序状态。其余操作的定义方式类似，同时考虑错误情况及相应的更新。

5 实验

图4展示了用户使用 JepREST 工具测试具有 REST 接口的应用程序时需要遵循的步骤。对于希望使用 JepREST 的程序员来说，最大的工作量在于编写所研究 API 的规范、指出 REST 服务器的位置、提供 API 的 .war 文件，并在 API 的 Dockerfile 中指出其位置。其余步骤几乎都是自动完成的，只需在终端中执行命令，或者直接运行代码，例如运行生成器代码，或运行用于向应用程序提交测试的 Clojure 代码。

为了验证 JepREST 系统能否正确评估 REST 应用在发出并发请求时的执行情况，我们开发了一个以 Java 编写的应用，其中使用了<样式 id='9'>嵌入式 H2 数据库，并且仅创建了一个资源和 6 个不同的<样式 id='11'>端点：创建学生（POST）、根据给定标识符返回一名学生（GET）、返回所有学生（GET）、根据给定标识符修改一名学生（PUT）、根据给定标识符部分修改一名学生（PATCH），以及根据给定标识符删除一名学生（DELETE）。在实现这些<样式 id='13'>端点 时，我们选择在需要修改数据库时不使用事务。之所以作出这一决定，是因为并发环境中可能会出现一致性问题；我们希望确认 JepREST 能否在 API 的多次执行中检测到这些问题，其中会发出各种类型的 REST 请求。

在实现 API、编写其规范并生成 Clojure 代码后，编写了要运行的测试，并将其保存在 YAML 文件中，其中定义了一个场景：每 1 毫秒执行多次，持续 30 秒，如图 5 所示。该场景包含一系列操作

```
场景:
    权重:      100
    流程:
      — 创建学生数据
      — 更新学生数据
      — 获取所有学生数据
      — 获取学生数据
      — 删除学生数据
```

图 5. JepR 系统执行的场景 EST.

```
11:59:59 :4 : 调用, : put, {: 名字 " Brycen ", : 姓氏 "Cummerata ",
            : 电子邮件 "adam@prince .com", : 年龄 129, : 电话 "119-364-8408"},
            : 路径 "498C98D9E8CB"
11:59:59 :0 : 调用, : GET
11:59:59 :2 : 调用, : GET, : 路径 "498C98D9E8CB"
11:59:59 :3 : 调用 , : 删除 , : 路径 "498C98D9E8CB"
11:59:59 :4 : 确定, : PUT, : 输出 {: 标识符 "498C98D9E8CB",
            : 名字 "Brycen ", : 姓氏 "Cummerata ",
            : 电子邮件 "adam@prince .com", : 年龄 129, : 电话 "119-364-8408"}
11:59:59 :3 : ok, : 删除 , : 输出 "498C98D9E8CB"
11:59:59 :0 : 失败 , : GET
11:59:59 :2 : 确定, : GET, : 路径 "498C98D9E8CB", : 输出 {: 标识符 "498C98D9E8CB",
            : 名字 "Brycen ", : 姓氏 "Cummerata",
            : 电子邮件"adam@prince .com", : 年龄 129, : 电话 "119-364-8408"}
```

图6。部分 d e 在执行期间由 JepRest 生成的 *log* ˛c˜所定义的测试。

这些操作将由多个客户端并发提交给应用程序；即使序列中途发生故障，JepREST 的多个客户端仍会继续发出这些操作。

JepREST 多次将此测试提交给创建的应用程序 `，并在几乎每次执行中都检测到了不同的一致性问题。接下来将展示所定义测试的多次执行结果，并简要描述检测到的一致性问题。

实验 1。 图6展示了 JepREST 执行所定义的测试后生成的部分日志 ´，其中呈现了已提交的操作序列，以及应用程序执行这些操作后的响应。日志 中的一行按从左到右的顺序包含以下信息：表示操作被调用/执行时刻的时间戳 、负责提交/接收该操作的 Jepsen 客户端标识符、表示该操作是调用（invoke）还是响应（ok/fail）的标记、REST 操作类型（post/put/get/- patch/delete），最后是有关操作的额外信息，例如输入 和输出。

此日志 显示在第 1、4、5 和 6 行，向应用程序并发提交了四个不同的 REST 操作 `。观察应用程序对这些请求的响应可知，第 10 行的 DELETE ´操作成功执行，也就是说，标识符为“498C98D9E8CB”´的学生已被删除；然而，第 12 行中，应用程序响应称 GET ´操作成功执行，并返回了标识符为“498C98D9E8CB” ´的学生。这是因为 GET 操作在检查该学生是否存在之后，给定标识符对应的学生才被删除˛。不过，由于

```
1  12:30:17  :2  : 调用,  : put,  {: 名字  " Sasha ",  : 姓氏  "Hyatt ",
2            : 电子邮件 " c l a u d i n e . prosacco@hotmail . com",  : 年龄  93 ,
3            : 电话  "(165)479 - 5262  x15024"} , : 路径  "71D1083D76BD"
4  12:30:17  :3  : 调用 ,  : get
5  12:30:17  :1  : 调用 ,  : GET,  : 路径  "71D1083D76BD"
6  12:30:17  :0  : 调用  ,  : 删除  ,  : 路径  "71D1083D76BD"
7  12:30:17  :1  : 确定,  : 获取,  : 路径  "71D1083D76BD" ,  : 输出  {: 标识符  "71D1083D76BD" ,
8            : 名字  "Grayce " ,  : 姓氏  " Brekke " ,
9            : 电子邮件"marques . rodriguez@yahoo .com" ,  : 年龄  40 ,
10           : 电话"1 - 162 - 508 - 6862 x13530"}
11 12: 30: 17  :3  : 确定,  : GET,  : 输出  {: 标识符  "71D1083D76BD",
12           : 名字  "Grayce " ,  : 姓氏  " Brekke " ,
13           : 电子邮件 "marques . rodriguez@yahoo .com" ,  : 年龄  40 ,
14           : 电话 "1 - 162 - 508 - 6862 x13530" })
15 12:30:17  :0  : ok ,  : 删除  ,  : 输出  "71D1083D76BD"
16 12:30:17  :2  : ok ,  : PUT,  : 路径  "71D1083D76BD",  : 输出  nil
```

图7. JepRest 在执行已定义的测试期间生成的 *log* 的一部分。

DELETE 操作在 GET 操作之前成功执行（第 10 行），因此 JepREST 报告了一致性错误，指出 GET 操作不可能成功执行，因为给定标识符对应的学生已经被删除。

实验 2. 本实验展示了对同一测试的另一次执行，该测试应用于已实现的 REST 应用程序，所得结果与第一次实验不同。图7展示了 JepRestest 生成的日志 的一部分，其中呈现的信息与前一次实验的 日志 相同。

与实验 1 的 日志 一样，可以从第 1、4、5 和 6 行看到来自不同客户端的多个并发请求 `访问 REST 应用。在本次实验中，JepREST 在第 16 行检测到一致性错误，因为 ´可以观察到，更新标识符为“71D1083D76BD”的学生的操作´已成功执行；然而，删除该标识符对应学生的操作在此之前不久已于第 15 行成功执行。所创建的系统 JepREST 检测到这一不一致，并报告 PUT 操作 ´的结果不正确（第 15 行），因为该操作试图更新的学生标识符已被 DELETE 操作删除（第 16 行）。

6 结论与未来工作

JepREST 系统能够简化采用 REST 接口的分布式应用测试，并验证其正确性。当前版本的系统包含测试采用 REST 标准 模型的应用所需的全部组件。实验结果表明，JepREST 能够检测出仅因多个客户端之间的并发而出现的应用正确性问题。

JepREST 系统仍在开发中，尤其是以下功能。首先，应用程序的执行支持 ` 正在扩展，以支持在执行过程中注入故障。要引入的故障将由程序员在脚本文件中指定。其次，计划支持实现不同语义的操作，

包括定义事务的功能。为此，需要使用 Elle [20] 来验证应用程序的正确性。

致谢：感谢 Kyle Kingsbury 对 Je-pREST 系统提出的意见。本研究部分由 FCT/MCTES 的项目 UIDB/04516/2020、PTDC/CCI-INF/32662/2017 和 PTDC/CCI-INF/32081/2017 资助。

参考文献

1. 2021 年 15 款顶级负载测试工具（开源指南）。https://testguild.com/load-testing-tools，访问于 2021年7月 2. 一款开源负载测试工具。https://locust.io/，访问于 2021年7月 3. Apache JMeter。https://jmeter.apache.org/，访问于 2021年7月 4. Artillery? https://artillery.io/，访问于 2021年7月 5. 代号 Taurus。https://gettaurus.org/，访问于 2021年7月 6. 一致性模型。https://jepsen.io/consistency，访问于 2021年2月 7. Faker 文档! https://faker.readthedocs.io/，访问于 2021年7月 8. Jepsen。https://jepsen.io/，访问于 2021年6月 9. Knossos。https://github.com/jepsen-io/knossos，访问于 2021年6月 10. OpenAPI 规范。https://swagger.io/specification/，访问于 2021年7月 11 2021 12 1998年 2006年。cURL。https://curl.se/（），访问于 2021年6月。SoapUI。https://www.soapui.org/（），访问于 2021年6月 2021 13 2018年。ReqBin：在线 REST 与 SOAP API 测试工具（），访问于 2021年6月 14 2020年。Dredd。https://dredd.org/en/latest/index.html（），访问于 2021年6月 15 2020年。Postman。https://learning.getpostman.com/（），访问于 2021年6月 16. Alvaro, P., Tymon, S.：将天才从故障测试中抽离：普通用户需要能够自动选择定制故障并进行注入的工具。《Queue》**15**(5)，29–53（2017年10月）。17. Freitas, F., Leitao, J., Pregui.ca, N., Rodrigues, R.：刻画在线服务的一致性（实践经验报告）。载于 DSN 2016。第 638–645 页（2016）。18. Herlihy, M.P., Wing, J.M.：线性一致性：并发对象的一种正确性条件。《ACM 程序设计语言与系统汇刊》**12**(3)，463–492（1990年7月）。19. Jiang, Z.M., Hassan, A.E., Hamann, G., Flora, P.：自动识别负载测试问题。载于：ICSME 2008。第 307–316 页。IEEE（2008年9月）20. Kingsbury, K., Alvaro, P.：Elle：从实验观察中推断隔离异常。《VLDB 会议录》**14**(3)，268–280（2020年11月）21. Madeira, H., Costa, D., Vieira, M.：关于通过软件故障注入来模拟软件故障。载于 DSN 2000。第 417–426 页。（2000年6月）22. Matos, M., Pardal, M., Amaral, M.A.P., Mercier, H.：Faultsee：分布式系统中的可复现故障注入。载于 EDCC 2020（2020年9月）23. Natella, R., Cotroneo, D., Madeira, H.S.：通过软件故障注入评估可信性：综述。《ACM 计算调查》**48**(3)（2016年2月）。24. Nidhra, S., Dondeti, J.：黑盒与白盒测试技术——文献综述。《国际嵌入式系统与应用杂志》**2**(2)，29–50。25. Ribeiro, A.：《不变量驱动的自动化测试》。硕士论文，里斯本新大学，里斯本（2021）26. Stott, D., Floering, B., Burke, D., Kalbarczpk, Z., Iyer, R.：NFTAPE：一种利用轻量级故障注入器评估分布式系统可信性的框架。载于 IPDS 2000。第 91–100 页（2000年3月）。

27. Tan, C., Zhao, C., Mu, S., Walfish, M.：Cobra：让事务型键值存储实现可验证的可串行化。载于：OSDI 2020。第63–80页。（2020年11月）。28. Tseitlin, A.：反脆弱组织。《ACM通讯》**56**(8)，40–44（2013年8月）。