

# ReadyAPI 从入门到放弃

ReadyAPI 的入门到放弃之路，是一家创业公司的测试团队从0到1的发展之路。在怎样的快速迭代和团队资源下，选择了ReadyAPI 进行 RESTful API 测试？ReadyAPI 如何支持多套测试环境，脚本参数化，检查点？如何与Jenkins集成？在两年的实践过程中，ReadyAPI 达到了怎样的效果，是如何配合团队成长的？最终又为何选择放弃？本文将向你——道来。

## 背景

2015 年1月我加入一家提供 SaaS 服务的创业公司。当时每周有一次常规版本迭代，不定期的 hotfix 上线。众所周知，每一次的版本更新都有可能引入风险。在这样频繁的版本迭代之下，团队对测试的期望是，将对外公开的 RESTful API 服务作为测试环节的重中之重，确保覆盖每一个接口的每一个字段，将版本更新的风险降到最低。

一般来说，一项 code change 完整的测试过程是：

1. 开发人员提交代码到分支，测试人员在测试环境中针对该分支进行测试。
2. 分支测试通过后，开发人员合并代码到 master，测试人员在测试环境中针对 master 进行验证。
3. 测试通过之后即可上线。版本上线之后，测试人员在生产环境进行验证。

上述过程还未包括回归测试过程，测试人员针对 bugs 的复测，以及预发布环境的环节，接口的压力测试等等，可见实际的测试工作量之大。而当时，测试团队只有我一个人，而我并不知道何时会有下一个小伙伴加入。不论是从提高工作效率的角度，还是团队沉淀的需要，API 的测试自动化都是势在必行。

## 工具选型

在我加入之前，API 测试主要是开发人员通过 Chrome 插件 DHC（现又称“Restlet Client”），Postman 之类的 REST API 客户端工具来验证可用性。尽管现在这两款工具都越来越强大，支持多环境 URL 切换，Assertion 设置等功能，但在两年前，它们的功能比较单薄，可编程性不强，不能算是专业的测试工具。

对于任何工具的选型，我认为都不应该脱离团队与业务当下的实际情况与发展趋势。当时对 API 测试工具的需求归纳为以下几点。

- 软需求：

- 上手快，学习成本低。虽然当时测试人员只有我一人，但招聘一直在进行中。我不能选择一款非常依赖某种特定编程语言的工具，这无异于对招聘提高了难度，更何况大量依赖编程语言写的脚本，在 testcase failed 的情况下，要花费的 debug 时间可能会长一些。最好是图形界面友好，测试人员只需要关注测试接口本身，不需要花很多时间去学习“如何实现”测试操作。
- 社区氛围好，技术资源充足。创业团队的时间宝贵，还是使用业界的“知名”工具吧，即便有坑也大多被前人踩过了。

- 硬需求：

- 强大的配置功能。可配置多个环境的 Root Url，以及该环境特有的测试数据。可以做到一套脚本，适用于多个环境，方便切换。
- 强大的 Assertion 功能。无论是检查响应状态码，还是 Header，又或者检查某个字段是否存在，检查该字段的类型，字段的值，甚至是检查该字段是否符合某一项规则，都可以方便添加检查点。
- 支持数据驱动，以及测试场景之间的逻辑调用。对于单个接口而言，往往要针对一个字段测试多个值。对于多个接口相关联的场景而言，经常会遇到接口 B 的执行，要依赖接口 A 的响应结果；又或者要等到异步处理，需要设置 Think time。
- 可编程性：可以使用编程语言生成接口签名，或实现一些复杂的字符串处理。
- 脚本的执行支持命令行，可灵活执行 Test Cases，Test Suites，方便集成在 Jenkins 之类的 CI 工具中。
- 生成各种类型的报告，比如接口测试覆盖度报告，测试结果报告。

简而言之，我的目标是“尽可能快而全面地提供质量反馈”。

最终，我选择了 ReadyAPI：<https://smartbear.com/product/ready-api/overview/>

## ReadyAPI 实践过程

ReadyAPI 实践过程中的细节很多，篇幅有限，下文将从“功能概览”，“使用 SoapUI”，“集成 Jenkins”和“建立项目规范”四个方面来介绍 ReadyAPI 是如何融入到日常测试工作中的。

### 功能概览

Case, Test Step。一个 Project 就是一个 xml 格式的文件。

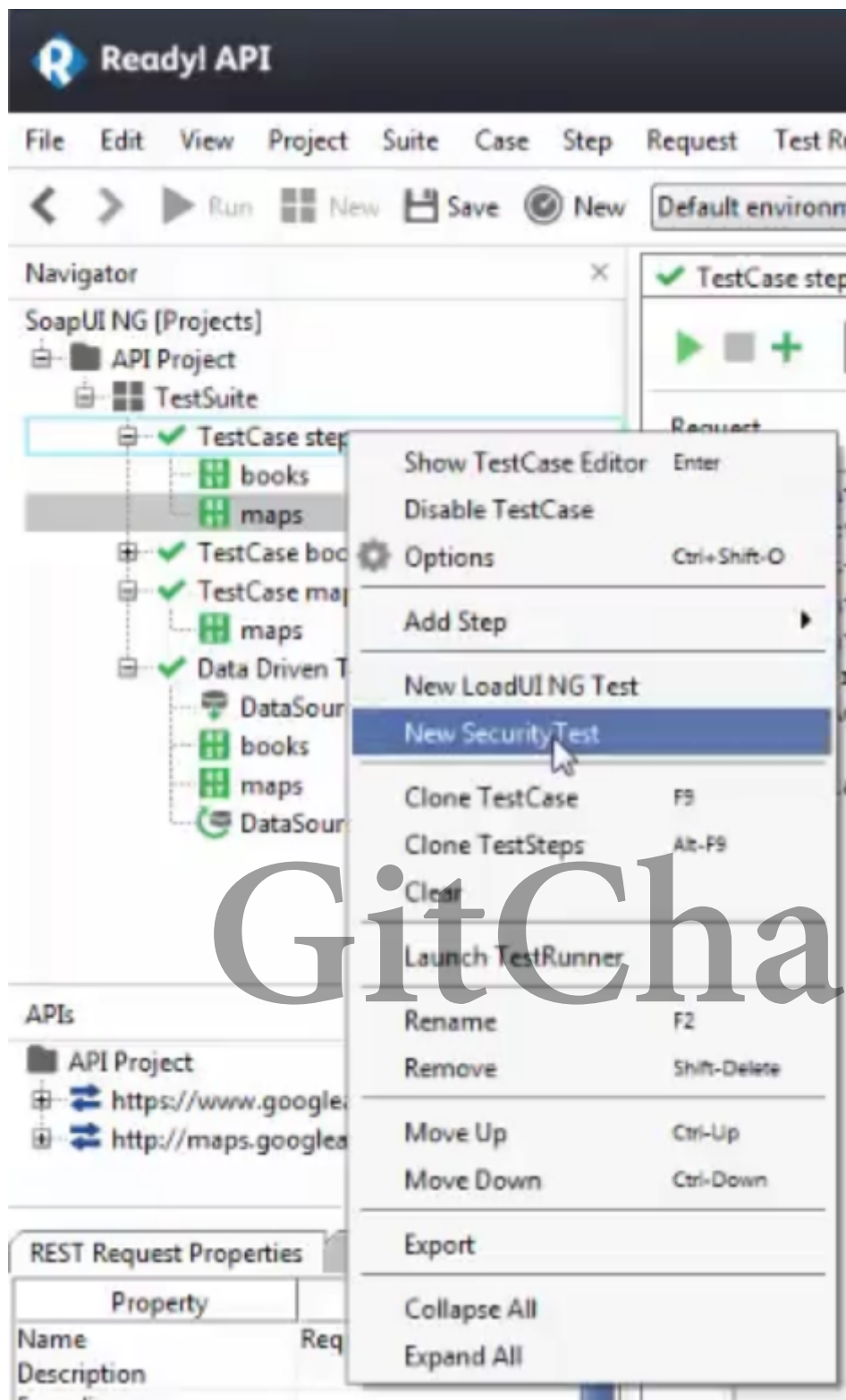


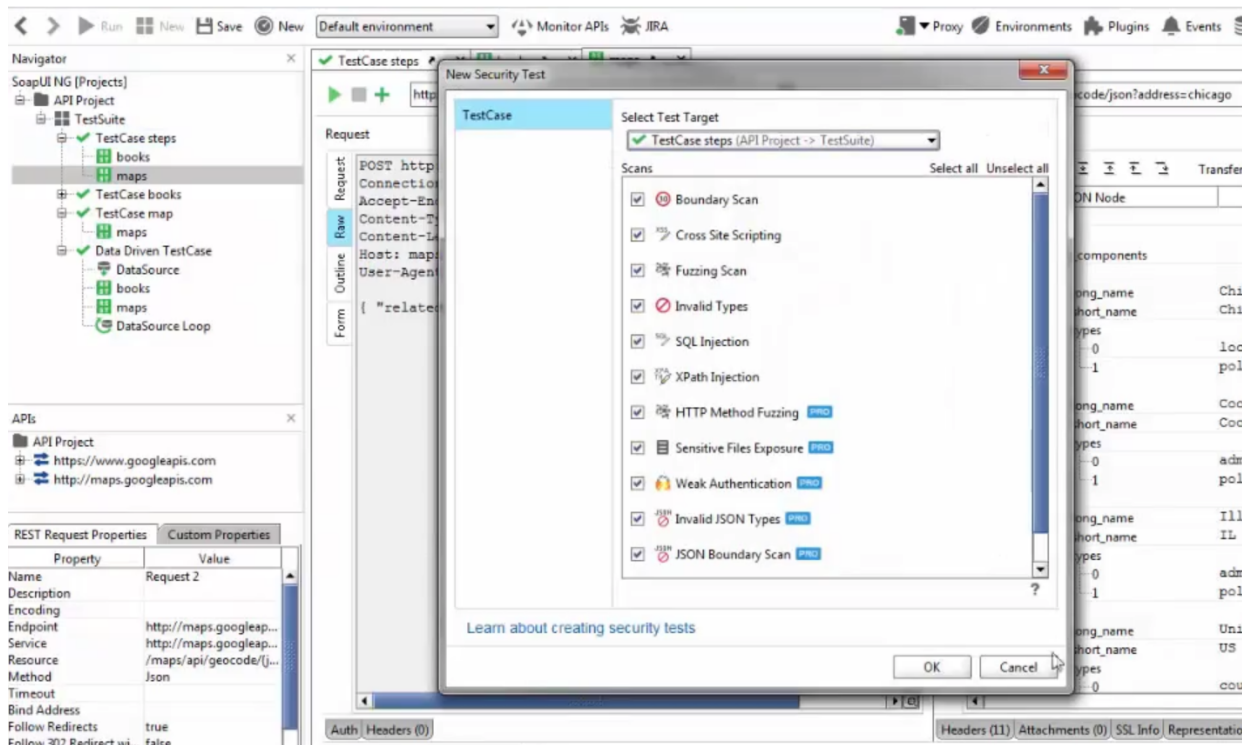
Functional Testing features	SoapUI Open Source	SoapUI NG Pro
WSDL Coverage		✓
Request/Response Coverage		✓
Message Assertion	✓	✓
Test Refactoring		✓
Test History: Baseline and Comparison		✓
OAuth2 Support in HTTP requests		✓
REST Discovery		✓
Dynamic Data Generation		✓
Drag and Drop Test Creation	✓	✓
Message Pretty Printing	✓	✓
Coding Free Test Assertion		✓
Running of Multiple Tests	✓	✓

Functional Testing features	SoapUI Open Source	SoapUI NG Pro
WSDL Coverage		✓
Request/Response Coverage		✓
Message Assertion	✓	✓
Test Refactoring		✓
Test History: Baseline and Comparison		✓
OAuth2 Support in HTTP requests		✓
REST Discovery		✓
Dynamic Data Generation		✓
Drag and Drop Test Creation	✓	✓
Message Pretty Printing	✓	✓
Coding Free Test Assertion		✓
Running of Multiple Tests	✓	✓

Running on multiple tests	✓	✓
Test Logs	✓	✓
Test Configuration	✓	✓
Easy Content Transfer	✓	✓
Data Source Driven Tests		✓
Data Collection		✓
MockResponse	✓	✓
Maven Integration	✓	✓
Standalone Server Runners	✓	✓
Scripting Support (Groovy, JavaScript)	✓	✓
Scripting Libraries		✓
Requirements Management		✓
Unit Reporting	✓	✓
Advanced Reporting		✓
Form Based input for easy manual testing		✓
Tree Based input for easy manual testing		✓
Create Test from Web Service Recordings	✓	✓
OAuth 2 Support	✓	✓
OAuth 2 Profiles	✓	✓
OAuth 2 Flow Automation	✓	✓
WS-Security Support	✓	✓
WS-I Integration	✓	✓
Web Service Recording	✓	✓
WS-Addressing Support	✓	✓
WS-Reliable Messaging	✓	✓
Manual TestStep	✓	✓
Assertion TestStep		✓
Assertion entire Message		✓
Test Debugging		✓
SAML 1&2 Support	✓	✓
NTLM 1&2 Support	✓	✓
Kerberos Support	✓	✓

而就 API 的安全测试来说，除了 Secure Pro 之外，我们还使用过其他一些接口安全扫描工具。单从工具扫描结果来看，没发现 Secure Pro 有啥特别的亮点。只是结合 SoapUI 来扫描非常方便，如下图，直接右键选择安全测试即可。笔者在安全方面的造诣太浅，就不再妄论了。





再谈到 API 的压力测试，由于当时 Web 界面的压测工具比较适合我们团队，可以将测试人员从压测的执行过程中解脱出来，因此我们最终没有选择 LoadUI 作为日常压测工具。

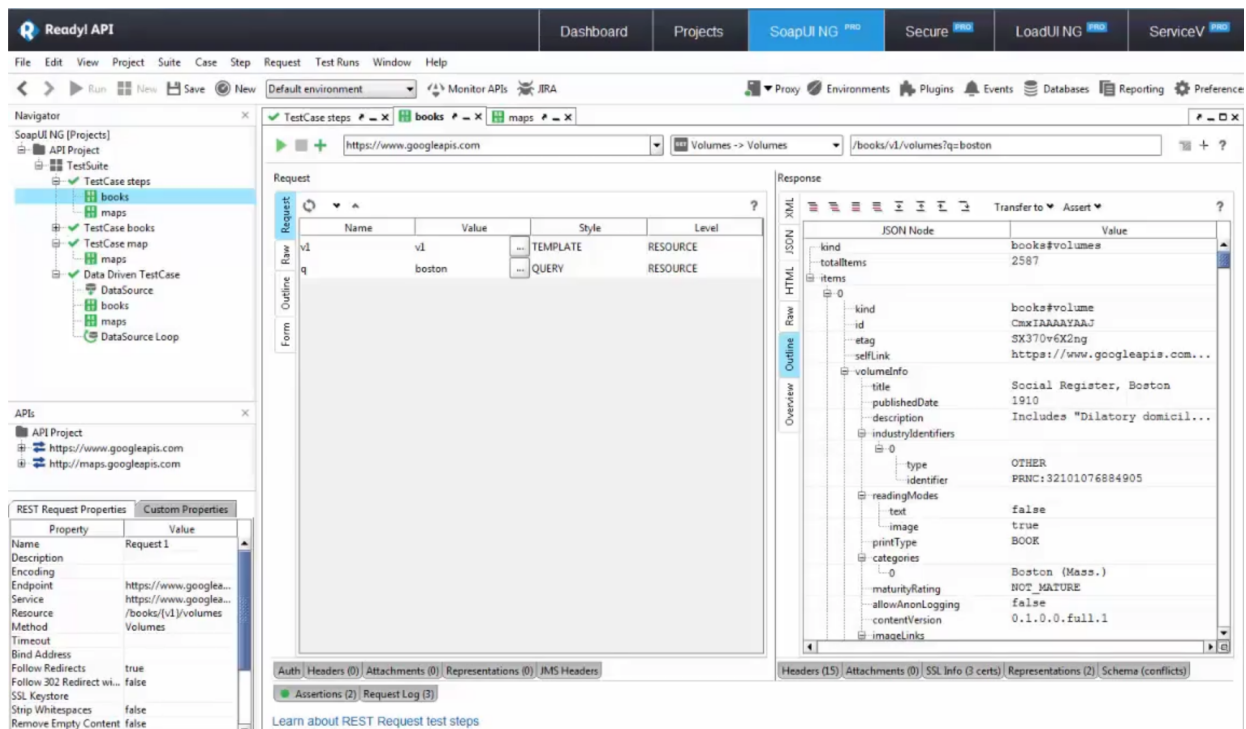
综上所述，这两年来虽说我们一直在做 ReadyAPI 实践，本质上其实是 SoapUI NG Pro 实践。

# GitChat

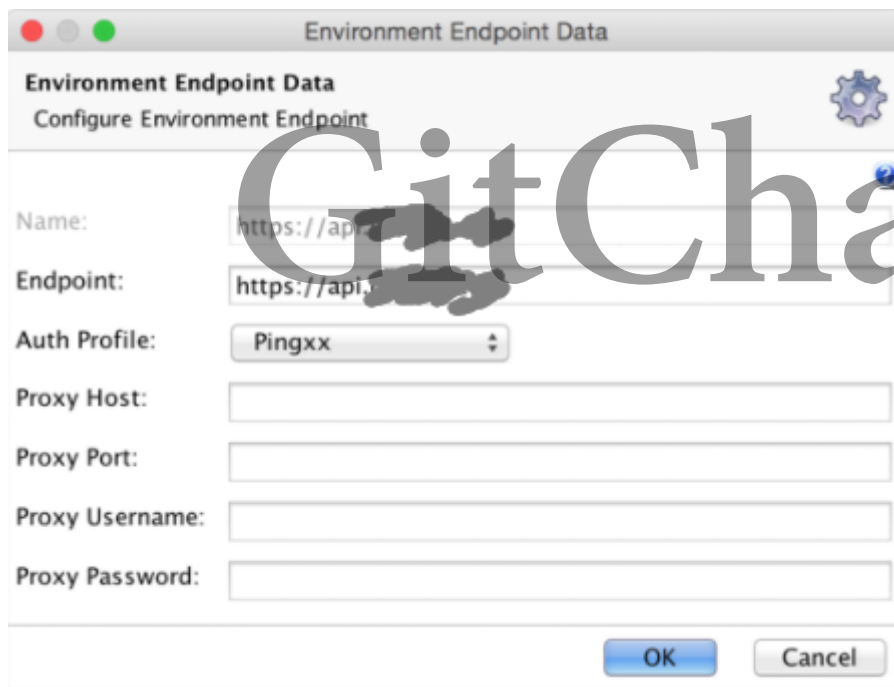
## 使用 SoapUI

接下来，我们简单聊聊 SoapUI 的功能。操作细节在 ReadyAPI 官网上有非常详细的教程，篇幅有限，咱们只是来侃侃常用的功能。

下面是 SoapUI NG 的截图。从图中可以看出，整个界面布局非常合理。不需要看教程就可以渡过破冰阶段。左侧的项目结构图中，绿色表明该步骤上一次执行成功，所有检查点都通过了。

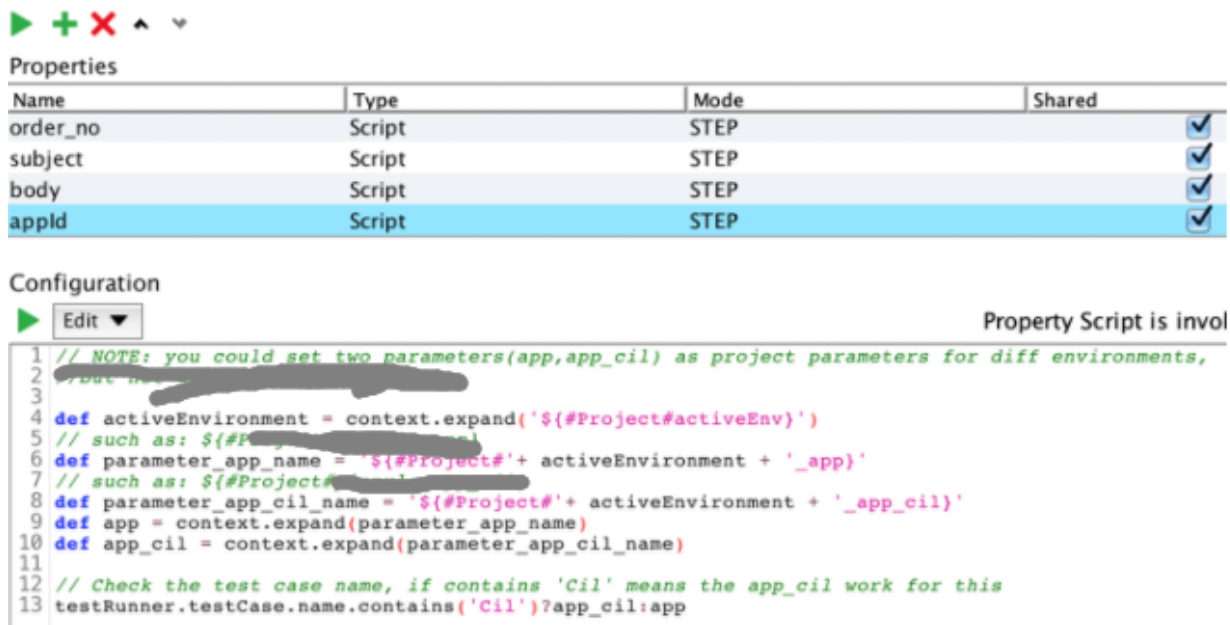


工具栏有“Environments”选项供你管理环境信息，有下拉列表方便切换。



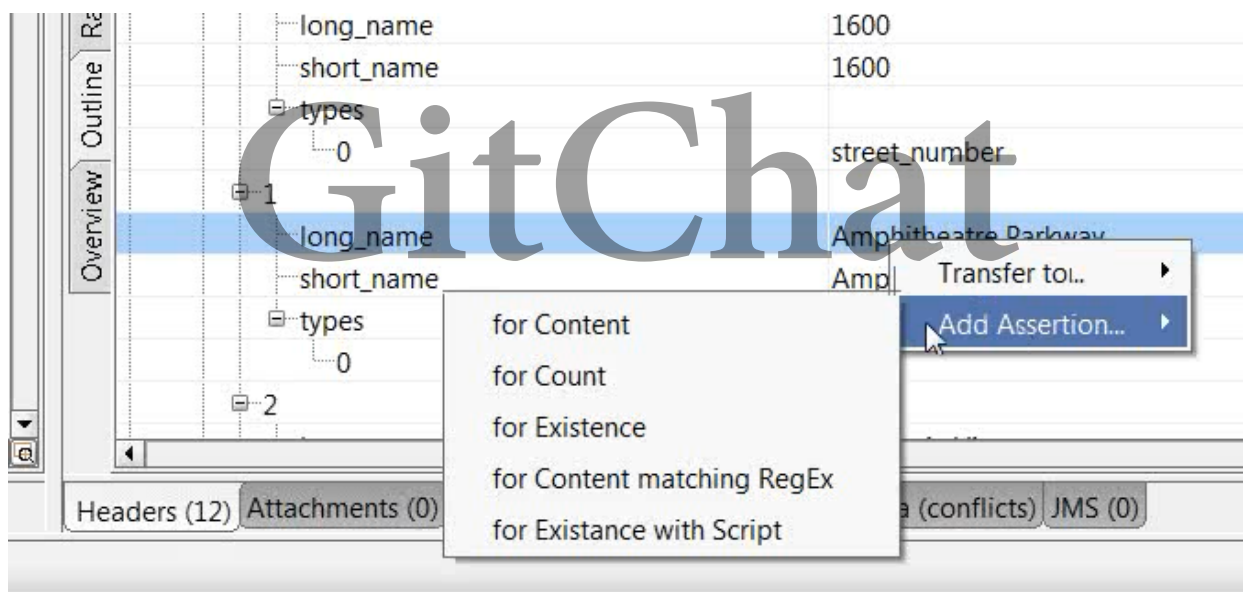
可添加 Groovy 或 JavaScript 作为测试步骤。下图是 Groovy 脚本的示例，根据当前所选择的环境，来判断获取哪一个值。





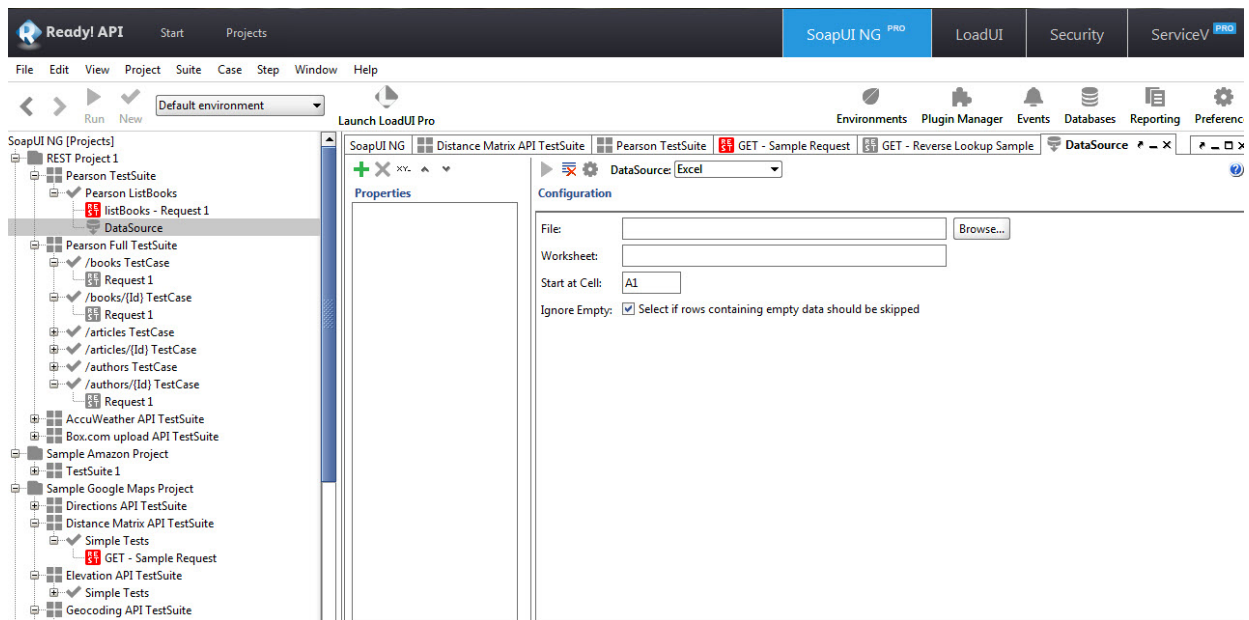
## 单个接口测试：Assertion和数据驱动

添加检查点非常方便。如下图，右键选中 Response body 中的字段，就可以直接添加 Assertion 了，支持正则表达式。



为了满足数据驱动的需要，SoapUI 提供了 DataSource 类型 TestStep。如下图，它支持 excel，xml，JDBC 连接（需要将驱动 Jar 包置于 ReadyAPI 安装目录下的 bin/ext 目录中）等。

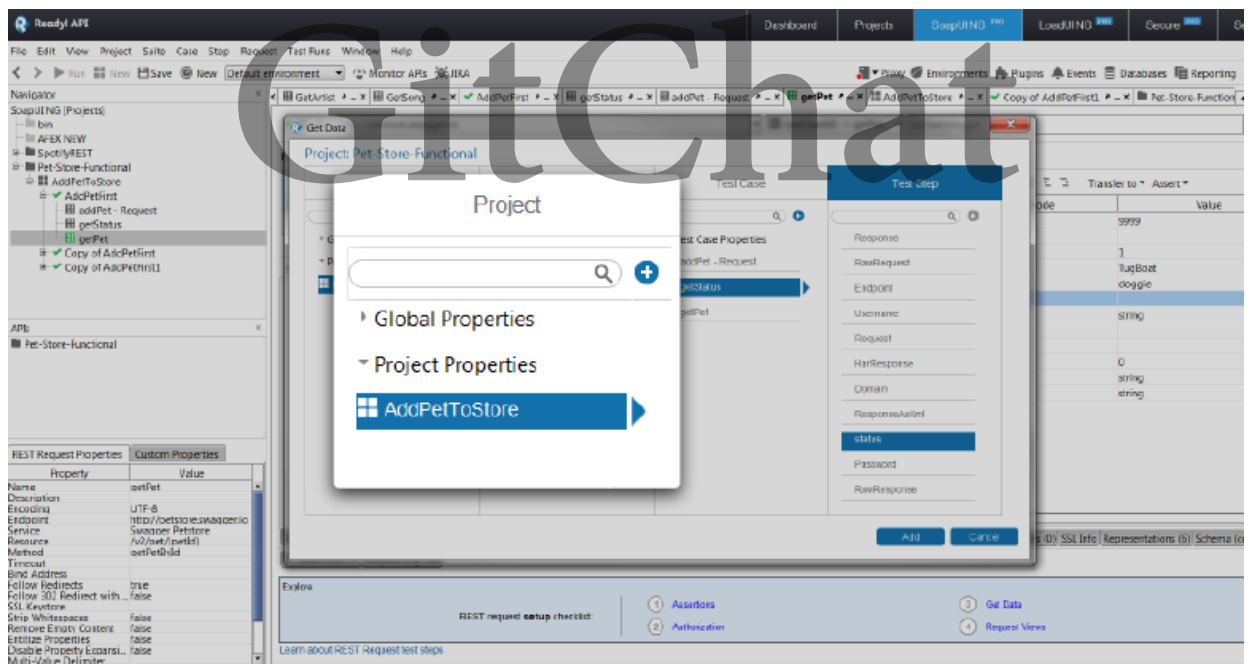




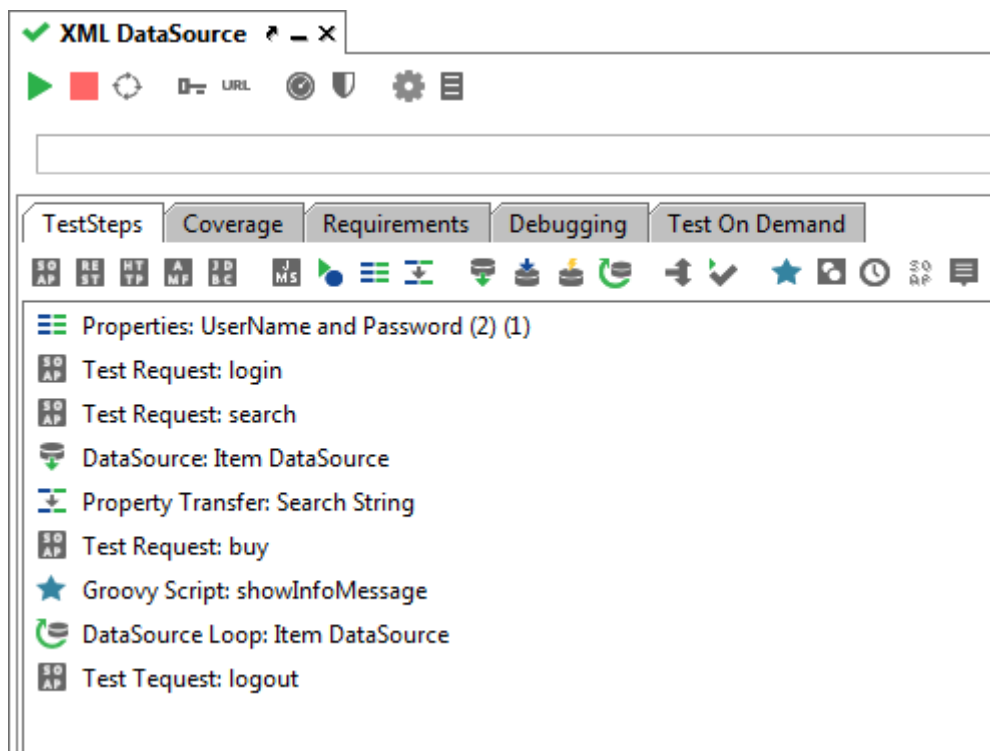
JDBC 驱动下载：<https://www.soapui.org/jdbc/reference/jdbc-drivers.html>

## 多个接口的场景测试：参数值传递

如下图，某个接口的字段值，可以来自全局或项目属性，或是其他 test step 的 Response。

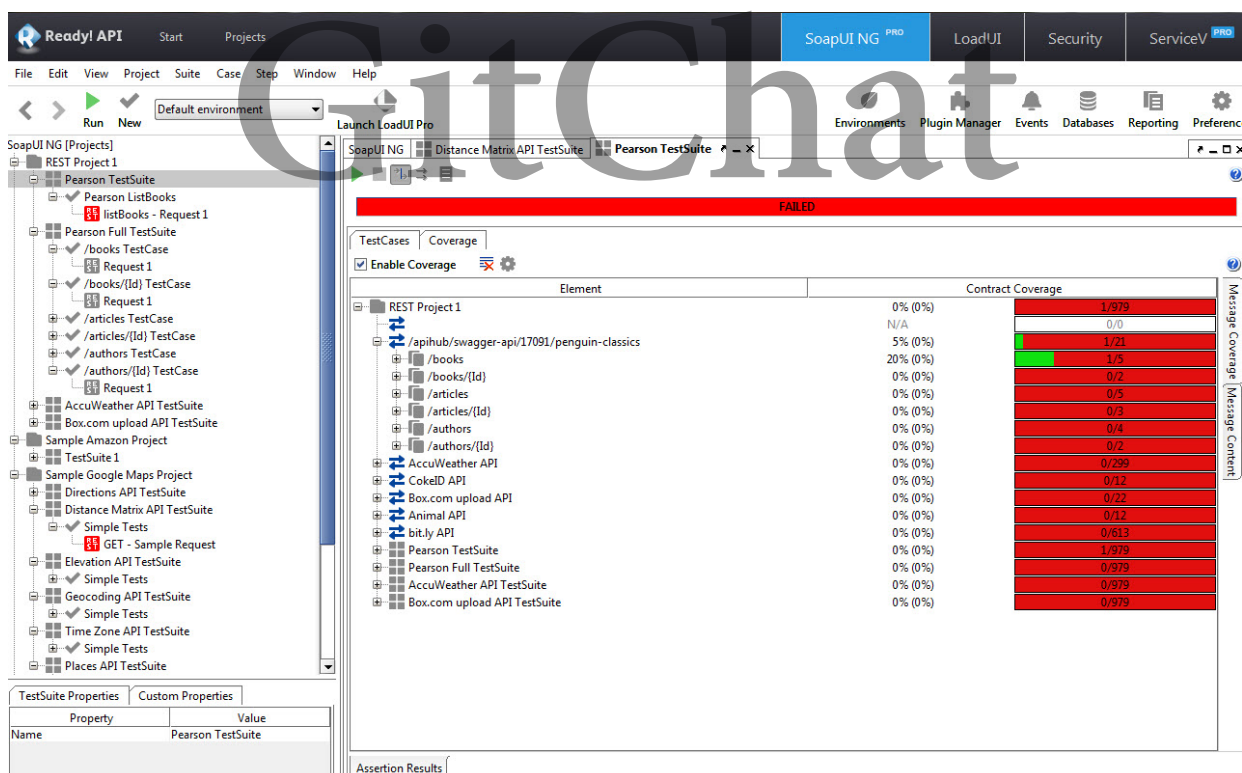


下图是一个购物场景，从登录，搜索，购买完成，最后注销。



## 测试覆盖率

勾选 Enable Coverage 之后执行 Test Suite，即可查看覆盖率情况。



Jenkins 集成：使用 soapui-pro-maven-plugin 插件

实践过程中需要 Maven，Jenkins 的基础知识。在 SoapUI 项目配置到 Jenkins 之前，先按照以下步骤确保 SoapUI 项目在本地可通过 Maven 编译执行。


- 安装 Maven，此处略过。

- 下载插件：  
<http://smartbearsoftware.com/repository/maven2/com/smartbear/soapui/soapui-pro-maven-plugin/>.
- 终端命令安装插件包：

```
mvn install:install-file -Dfile=/home/applewu/soapui-pro-maven-plugin-5.1.2.jar -DgroupId=com.smartbear.soapui -DartifactId=soapui-pro-maven-plugin-5.1.2.jar -Dpackaging=jar -Dversion=5.1.2
```


- 为你的 soapui 项目准备相应的 pom.xml，保证 pom.xml 与 soapui 项目文件在同一目录下。pom.xml 格式参考官网：<http://www.soapui.org/test-automation/maven/maven-2-x.html>
- 终端命令运行 soapui 项目：mvn com.smartbear.soapui:soapui-pro-maven-plugin:5.1.2:test

确保 SoapUI 项目在本地执行，可生成类似 JUnit 结果的 xml 格式的报后，便可在 Jenkins 上创建 Maven 项目，进行配置。以下是测试结果。可在 Jenkins 上选择单个模块运行测试。



Revision: c6a04cea55efae41e2c16a706924d329121df7ca

- refs/remotes/origin/master



Test Result (失败)

- [RedEnvelope\\_TK.GetRedEnvelopes.GetRedEnvelopesByExpand](#)
- [RedEnvelope\\_TK.GetRedEnvelopes.GetRedEnvelopesByChannel\\_Status](#)
- [Event\\_TK.GetEvent\\_Negative.GetEvent-InvalidType](#)
- [Event\\_TK.GetEvent\\_Negative.GetEvent-InvalidEventId](#)
- [Event\\_TK.GetEvent\\_Positive.GetEventByEventId-summary.daily.available](#)
- [Event\\_TK.GetEvent\\_Positive.GetEventByEventId-summary.weekly.available](#)
- [Event\\_TK.GetEvent\\_Positive.GetEventByEventId-summary.monthly.available](#)
- [Event\\_TK.GetEvent\\_Positive.GetEventByEventId-charge.succeeded](#)
- [Event\\_TK.GetEvent\\_Positive.GetEventByEventId-refund.succeeded](#)
- [Event\\_TK.GetEvent\\_Positive.GetEventByEventId-transfer.succeeded](#)

[显示全部失败单元测试 >>>](#)

## Module Builds

● ChargeTests-livemode (didn't run)	
● EventTests-livemode (didn't run)	
● RedEnvelopeTests-livemode (didn't run)	
● RefundTests-livemode (didn't run)	
● Transferests-livemode (didn't run)	
● SoapUI tests (didn't run)	
● <a href="#">ChargeTests-testmode</a>	2 分 9 秒
● <a href="#">EventTests-testmode</a>	1 秒
● <a href="#">RedEnvelopeTests-testmode</a>	21 秒
● <a href="#">RefundTests-testmode</a>	1 分 31 秒
● <a href="#">TransferTests-testmode</a>	21 秒
● <a href="#">SoapUI tests</a>	1.9 秒

## 建立项目规范

这里的“项目规范”，是测试人员在维护 SoapUI 项目过程中对命名规范，组织结构，以及检查粒度的约定。大概在 SoapUI 使用了半年之后，测试团队也不再是我单兵作战了，项目规范应运而生。我们的规范包括了以下几个部分。由于规范的具体内容与业务相关，这里就不详述了。

- 项目结构规范
- 命名规范
  - Project/Resource/Method/Request命名。
  - TestSuite/TestCase/TestStep命名。
- 基本信息的编写（比如用例场景，用例维护人员）
- 参数化
  - 环境参数化
  - 脚本参数化
- Assertion 设置

## 成果与困扰

在两年来的 ReadyAPI 实践过程中，测试团队由我一人，发展到十人；从测试人员本地执行的“小打小闹”，到融入团队的正式上线流程中，自动化脚本帮助我们发现了不少问题。曾经有开发同学无意中，将金额字段的返回类型由整型改成了字符串类型；曾经我们调用的第三方合作商接口的返回格式变更了，而开发同学不知情，导致原有功能不可用... 其实，充分的自动化测试，可以增强快速迭代的信心，开发同学不用担心代码重构会对测试同学增加很大的工作量。可以说，有了简单易用的 ReadyAPI，API 测试自动化很早就融入到团队的基因中。

然而，在使用 ReadyAPI 过程中，存在一些困扰：

- 第一是 ReadyAPI 很容易在本地 Crash，尤其是在 workspace 里加载了两个以上的 Project 文件的时候。有时候倒不是一加载就卡死，而是你点了保存，正要保存刚刚写好的一段脚本，这时整个界面卡死。之后，就会经历那种再打开，发现什么都没有的绝望。这种事情真的发生不只一次了。
- 第二是 ReadyAPI 的 bugs，以及频繁的版本迭代。从2015年的1.2，到现在的 2.0 版本，差不多三个月更新一个版本。期间我们还遇到过 environment 切换的 bug，在 1.4.1 版本中 fixed 了。有些功能的向下兼容做得也不够好，A同学的脚本在B同学那

里，因为版本的问题可能就没法运行了。想想也挺不爽的，每隔几个月又要下载安装包，在本地装一遍，期间还可能遇到 bugs，真是闹心。

- 第三是不方便 review。我们的 SoapUI 项目文件是存放在 Git 上管理的。通过 diff 实在不方便 review 文件的版本差异。每次 review cases，就是通过 diff 大概了解一下改了哪些地方，然后再打开 ReadyAPI，在界面上查看具体的相关信息。

## 破碎与重建

随着业务增长越来越快，我们的日常工作越来越依赖 API 自动化，ReadyAPI 带来的困扰无形中也越来越严重。测试同学们纷纷“自寻出路”，有些人使用 JMeter，虽然要依赖一些插件才能很好地满足需求，但好歹用得顺手；有些人开始用 Java，Python 或 PHP 写 API 测试的脚本。虽说这些用起来可能比 ReadyAPI 麻烦一些，但好歹是自己控制的，哪里可能有坑，自己也会清楚一些。当然，为了避免重复劳动，有部分 ReadyAPI 已经覆盖的场景，大家仍在继续使用；而对于新的测试任务，大家就采用自己熟悉的方式去完成。至此，ReadyAPI 已经失去了我们团队中接口自动化测试的“第一把交椅”，我们重新开始探索 API 自动化测试的方法，将在一条新的大道上启程。

# GitChat