Restful测试工具，简单讲就是根据测试需求，向所测Web Server发起各种各样的Request请求，完整的获取Web Server返回Response响应。并能够便捷地对Response响应的各个内容进行验证。

那么web服务器测试工具Gulu——Restful测试工具是如何做到的呢？

本篇博文将结合代码实现及UML图示从请求设置，响应保存，响应验证，3个部分展开介绍。

【作者】 公远/陈震（gongyuan.cz$taobao.com/ 100continue$sina.com）

【源码下载】

github地址：https://github.com/100Continue/Gulu--web-server-test-tools

【功能介绍】

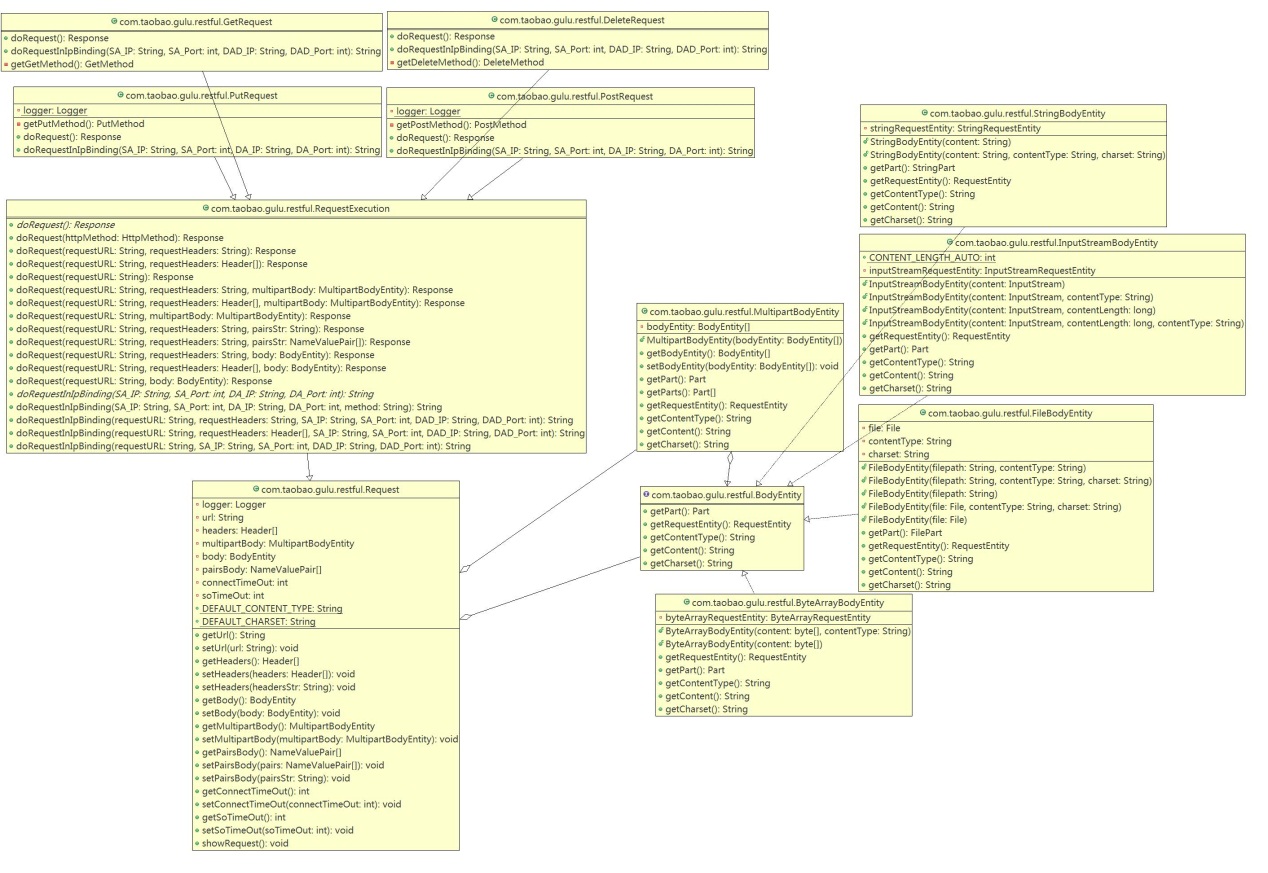
1. 请求设置功能：

设计要求：

1. 由于测试需求千差万别，因此需要支持各种请求构建需求，如自由定制Method、Header；请求Body允许各种类型——form表单类型的Key-Value对；MultiPart分段上传的body类型；ByteArray类型、String类型、InputStream类型、上传数据来自本地文件等；支持绑定虚拟IP的请求等等需求。
2. 支持代码编写便捷的要求；

具体实现：

结合上述设计要求及保障可扩展性，可维护性的要求。采用策略模式——将数据与算法分开的思路实现。详情参见下幅UML图：



Request作为数据保存类，复杂保存Request请求所涉及到的所有数据信息；

注：从代码编写便捷角度出发，Header与Form表单body设置支持String类型传入，如Header模板：a:b$c:d$cookie:tt=xx;cc=pp；表示Header为：a:b c：d cookie:tt=xx;cc=pp; Form表单Body模块：a=b$c=d$e=f； 表示Form表单Body为： a=b c=d e=f；

RequestExecution作为各种请求算法实现类，组合Request的各种数据向Web Server发起请求，支持本地IP地址及绑定虚拟IP地址的请求发起。

注：从代码编写便捷角度出发，定义多种doRequest、doRequestInIpBinding接口，从而实现一行测试代码发起测试的目的；

GetRequest、DeleteRequest、PutRequest、PostRequest作为具体请求类型对象，通过各种的get\*Method接口实现不同请求类型的区分；

注：PutRequest与PostRequest需要设置Body，如果未设置Body将抛出Exception报错；

BodyEntity接口定义了各种Body类型所需要支持的公共接口，Request类的Body数据直接采用该接口类型进行定义，从而不必再区分各种不同的Body类型。

ByteArrayBodyEntity、FileBodyEntity、InputStreamBodyEntity、StringBodyEntity、MultipartBodyEntity作为具体Body类型的实现类。

代码应用：

// 发起自定义Get请求

new GetRequest().doRequest(url, headers);

// 发起自定义Post请求上传Form表单数据并保存成Response对象

Response response = new PostRequest().doRequest(url, headers, formBody);

// 发起自定义Put请求上传本地文件并保存成Response对象

FileBodyEntity fileBody = new FileBodyEntity(filePath, null);

Response response = new PutRequest().doRequest(url, headers, fileBody)

// 发起自定义Post请求上传Multipart数据（包含本地文件、bytearray、inputstream类型，string类型）

StringBodyEntity stringBody = **new** StringBodyEntity(

"string body content", **null**, **null**);

FileBodyEntity fileBody = **new** FileBodyEntity(filePath, **null**);

ByteArrayBodyEntity byteArrayBody = **new** ByteArrayBodyEntity(

"byte array body content".getBytes(), **null**);

InputStream inputStream = **new** FileInputStream(**new** File(filePath));

InputStreamBodyEntity inputStreamBody = **new** InputStreamBodyEntity(inputStream, -2, **null**);

MultipartBodyEntity multipartBody = **new** MultipartBodyEntity(stringBody, fileBody);

new PostRequest().doRequest(url, multipartBody);

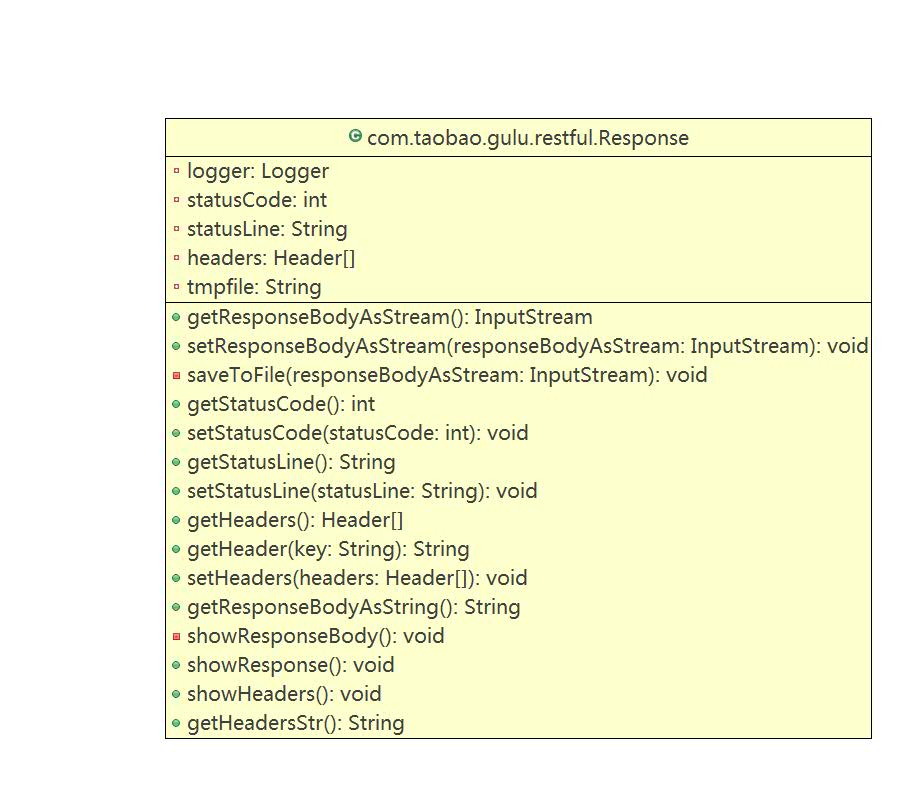
1. 响应保存功能：

设计要求：

1. 能够完整的保存Web Server返回的所有数据；
2. 能够自由便捷的取得所需要Response各个部分的数据；

具体实现：

基于上述设计要求，将Web Server返回的所有数据保存到Response类中，并对外提供查询接口，具体实现见下图：



Response作为Web Server返回的数据保存类，将状态码、状态提示、Header数据等保存于对象中，考虑到响应头中可能出现未包含Content-length 或Body长度超过最大Integer这两种情况，因此将Body保存成本地文件，对外提供getResponseBodyAsStream接口进行访问。

注：除了响应头未包含Content-length 或Body长度超过最大Integer这两种情况外，响应Body均可以通过getResponseBodyAsString接口进行访问。

代码应用：

// 请求后保存成Response对象并获取StatusCode

Response response = new PostRequest().doRequest(url, headers, formBody);

int statusCode = response.getStatusCode();

// 请求后直接获取Header数组

Header[] headers = new GetRequest().doRequest(url).getHeaders();

// 请求后直接获取响应Body

String body = new GetRequest().doRequest(url).getResponseBodyAsString();

1. 响应验证功能：

设计要求：

1. 支持所有Response组成部分的匹配验证，并且支持全匹配及包含匹配两种方式；
2. Response Body验证尽可能多的支持各种类型，如String类型、Json对象、本地文件对比验证等；
3. 代码编写便捷，类似Junit的Assert的使用习惯及验证失败后抛出Exception及失败提示信息；

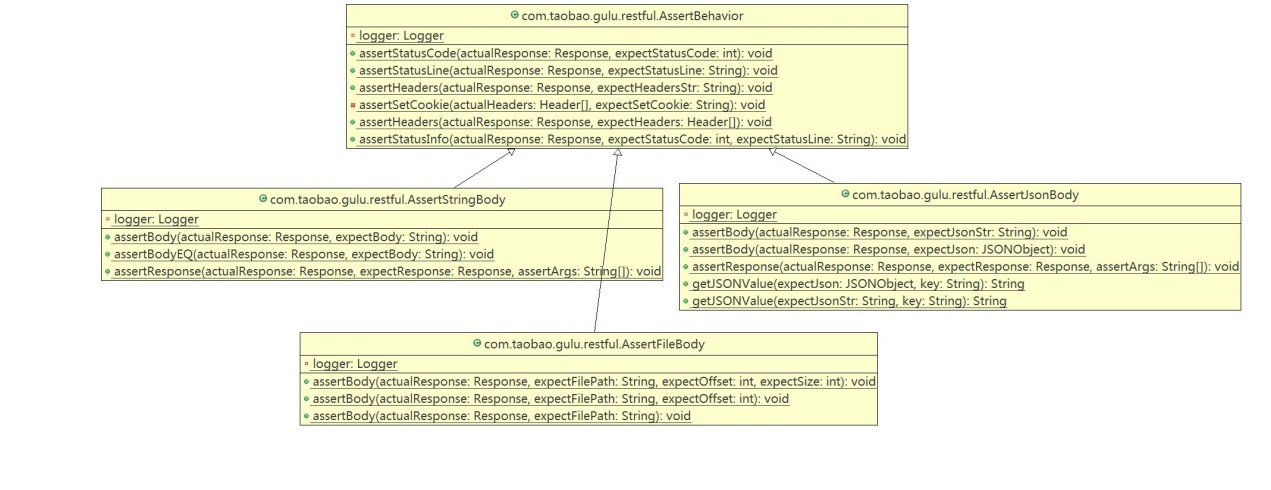
具体实现：

基于上述设计需求，响应验证功能设计有以下3个特点：

1. 为了具有类似Junit的Assert的使用习惯、验证类均为Static静态类，用户可直接使用类型调用验证接口进行测试；并且在Tools工具类中实现ComparisonFailureHandle 和 FailedHandle，使得验证结果及错误信息的展现与Junit的Assert相似，方便Bug定位和调试；
2. 将除Body外的所有Response组成部分的匹配验证单独封装在AssertBehavior中便于统一维护和扩展；
3. 继承于AssertBehavior类分别实现了3种不同类型的Response Body验证类，如AssertStringBody专门应用于Response Body是String类型的验证；AssertJsonBody专门应用于Response Body是Json对象的验证；AssertFileBody专门应用于Response Body需要与本地文件进行对比验证的测试；

注： Headers验证仅验证用户预期的Key Value，对Response包含的其他Header不做验证处理。并且相应的Header Value验证为包含匹配验证；

详情参见下图：



代码应用：

// 请求后保存成Response对象并验证StatusCode

Response response = new PostRequest().doRequest(url, headers, formBody);

AssertStringBody.assertStatusCode(response, 200);

AssertJsonBody.assertStatusCode(response, 201); // 验证失败则直接抛出Exception及Junit Assert类型的提示信息；

// 请求后保存成Response对象并验证Headers信息

Response response = new PostRequest().doRequest(url, headers, formBody);

AssertStringBody. assertHeaders (response, expectHeadersStr); // headers 支持上述Request类中设计的Header String 模板；

AssertJsonBody. assertHeaders (response, expectHeaders); // headers支持Header[]；

// 请求后保存成Response对象并验证String类型的Body信息

Response response = new PostRequest().doRequest(url, headers, formBody);

AssertStringBody. assertBodyEQ (response, expectResponseBody); //Body验证方式为全匹配方式；

AssertStringBody. assertBody (response, expectResponseBody); // Body验证方式为包含匹配方式；

// 请求后保存成Response对象并验证Json类型的Body信息

Response response = new PostRequest().doRequest(url, headers, formBody);

AssertJsonBody. assertBody (response, expectResponseJsonBodyStr); //Body为String类型的Json数据；

AssertJsonBody. assertBody (response, expectResponseJsonBodyStr); // Body为Json对象；

// 请求后保存成Response对象并将Response Body与本地文件进行对比验证类型

Response response = new PostRequest().doRequest(url, headers, formBody);

AssertFileBody. assertBody (response, localFilePath, offset, size); // 支持与本地文件的部分内容进行对比验证

AssertFileBody. assertBody (response, localFilePath); // 只需上传本地文件路径即可；

// 请求后保存成Response对象并直接对比两个Response对象

Response response1 = new PostRequest().doRequest(url, headers, formBody);

Response response2 = new PostRequest().doRequest(url, headers, formBody);

AssertStringBody. assertResponse (response1, response2);

AssertJsonBody. assertResponse (response1, response2);