RestTemplate 实践

By liuxing

2015-05-21 更新日期: 2015-07-30

文章目录

什么是 RestTemplate?

RestTemplate 是 Spring 提供的用于访问 Rest 服务的客户端，RestTemplate 提供了多种便捷访问远程 Http 服务的方法，能够大大提高客户端的编写效率。

调用 RestTemplate 的默认构造函数，RestTemplate 对象在底层通过使用 java.net 包下的实现创建 HTTP 请求，可以通过使用 ClientHttpRequestFactory 指定不同的 HTTP 请求方式。

ClientHttpRequestFactory 接口主要提供了两种实现方式

- 一种是 SimpleClientHttpRequestFactory，使用 J2SE 提供的方式（既 java.net 包提供的方式）创建底层的 Http 请求连接。
- 一种方式是使用 HttpComponentsClientHttpRequestFactory 方式，底层使用 HttpClient 访问远程的 Http 服务，使用 HttpClient 可以配置连接池和证书等信息。

最新实例代码 更新于 2015-07-30

xml 配置的方式

请查看 RestTemplate 源码了解细节，知其然知其所以然！

RestTemplate 默认是使用 SimpleClientHttpRequestFactory，内部是调用 jdk 的 HttpConnection，默认超时为-1

```
@Autowired
RestTemplate simpleRestTemplate
```

```
@Autowired
RestTemplate restTemplate
```

基于 jdk 的 spring 的 RestTemplate

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd"
default-autowire="byName" default-lazy-init="true">

<!--方式一、使用 jdk 的实现-->
<bean id="ky.requestFactory"
class="org.springframework.http.client.SimpleClientHttpRequestFactory">
<property name="readTimeout" value="10000"/>
<property name="connectTimeout" value="5000"/>
</bean>

<bean id="simpleRestTemplate"
class="org.springframework.web.client.RestTemplate">
<constructor-arg ref="ky.requestFactory"/>
<property name="messageConverters">
<list>
<bean class="org.springframework.http.converter.FormHttpMessageConverter"/>
<bean
class="org.springframework.http.converter.xml.MappingJackson2XmlHttpMessageC
onverter"/>
<bean
class="org.springframework.http.converter.json.MappingJackson2HttpMessageCon
verter"/>
<bean class="org.springframework.http.converter.StringHttpMessageConverter">
<property name="supportedMediaTypes">
<list>
<value>text/plain;charset=UTF-8</value>
</list>
</property>
</bean>
</list>
</property>
</bean>

</beans>
```

使用 Httpclient 连接池的方式

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd"
default-autowire="byName" default-lazy-init="true">

<!--方式二、使用 httpclient 的实现，带连接池-->
<bean id="ky.pollingConnectionManager"
class="org.apache.http.impl.conn.PoolingHttpClientConnectionManager">
<!--整个连接池的并发-->
<property name="maxTotal" value="1000" />
<!--每个主机的并发-->
<property name="defaultMaxPerRoute" value="1000" />
</bean>

<bean id="ky.httpClientBuilder"
class="org.apache.http.impl.client.HttpClientBuilder" factory-
method="create">
<property name="connectionManager" ref="ky.pollingConnectionManager" />
<!--开启重试-->
<property name="retryHandler">
<bean class="org.apache.http.impl.client.DefaultHttpRequestRetryHandler">
<constructor-arg value="2"/>
<constructor-arg value="true"/>
</bean>
</property>
<property name="defaultHeaders">
<list>
<bean class="org.apache.http.message.BasicHeader">
<constructor-arg value="User-Agent"/>
<constructor-arg value="Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/31.0.1650.16 Safari/537.36"/>
</bean>
<bean class="org.apache.http.message.BasicHeader">
<constructor-arg value="Accept-Encoding"/>
<constructor-arg value="gzip,deflate"/>
</bean>
<bean class="org.apache.http.message.BasicHeader">
<constructor-arg value="Accept-Language"/>
<constructor-arg value="zh-CN"/>
</bean>
</list>
</property>
```

```
</bean>

<bean id="ky.httpClient" factory-bean="ky.httpClientBuilder" factory-
method="build" />

<bean id="ky.clientHttpRequestFactory"
class="org.springframework.http.client.HttpComponentsClientHttpRequestFactor
y">
<constructor-arg ref="ky.httpClient"/>
<!--连接超时时间，毫秒-->
<property name="connectTimeout" value="5000"/>
<!--读写超时时间，毫秒-->
<property name="readTimeout" value="10000"/>
</bean>

<bean id="restTemplate" class="org.springframework.web.client.RestTemplate">
<constructor-arg ref="ky.clientHttpRequestFactory"/>
<property name="errorHandler">
<bean class="org.springframework.web.client.DefaultResponseErrorHandler"/>
</property>
<property name="messageConverters">
<list>
<bean class="org.springframework.http.converter.FormHttpMessageConverter"/>
<bean
class="org.springframework.http.converter.xml.MappingJackson2XmlHttpMessageC
onverter"/>
<bean
class="org.springframework.http.converter.json.MappingJackson2HttpMessageCon
verter"/>
<bean class="org.springframework.http.converter.StringHttpMessageConverter">
<property name="supportedMediaTypes">
<list>
<value>text/plain;charset=UTF-8</value>
</list>
</property>
</bean>
</list>
</property>
</bean>

</beans>
```

bean 初始化+静态工具

线程安全的单例（懒汉模式）

基于 jdk 的 spring 的 RestTemplate

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.context.annotation.Lazy;
import org.springframework.http.client.SimpleClientHttpRequestFactory;
import org.springframework.http.converter.FormHttpMessageConverter;
import org.springframework.http.converter.HttpMessageConverter;
import org.springframework.http.converter.StringHttpMessageConverter;
import
org.springframework.http.converter.json.MappingJackson2HttpMessageConverter;
import
org.springframework.http.converter.xml.MappingJackson2XmlHttpMessageConverte
r;
import org.springframework.stereotype.Component;
import org.springframework.web.client.DefaultResponseErrorHandler;
import org.springframework.web.client.RestTemplate;

import javax.annotation.PostConstruct;
import java.nio.charset.Charset;
import java.util.ArrayList;
import java.util.List;

/**
* @title: 基于 jdk 的 spring 的 RestTemplate
* @author: liuxing
* @date: 2015-05-18 09:35
*/
@Component
@Lazy(false)
public class SimpleRestClient {

private static final Logger LOGGER =
LoggerFactory.getLogger(SimpleRestClient.class);

private static RestTemplate restTemplate;

static {
SimpleClientHttpRequestFactory requestFactory = new
SimpleClientHttpRequestFactory();
requestFactory.setReadTimeout(5000);
requestFactory.setConnectTimeout(5000);

// 添加转换器
```

```
List<HttpMessageConverter<?>> messageConverters = new ArrayList<>();
messageConverters.add(new StringHttpMessageConverter(Charset.forName("UTF-
8")));
messageConverters.add(new FormHttpMessageConverter());
messageConverters.add(new MappingJackson2XmlHttpMessageConverter());
messageConverters.add(new MappingJackson2HttpMessageConverter());

restTemplate = new RestTemplate(messageConverters);
restTemplate.setRequestFactory(requestFactory);
restTemplate.setErrorHandler(new DefaultResponseErrorHandler());

LOGGER.info("SimpleRestClient 初始化完成");
}

private SimpleRestClient() {

}

@PostConstruct
public static RestTemplate getClient() {
return restTemplate;
}

}
```

使用 Httpclient 连接池的方式

```
import org.apache.http.Header;
import org.apache.http.client.HttpClient;
import org.apache.http.impl.client.DefaultConnectionKeepAliveStrategy;
import org.apache.http.impl.client.DefaultHttpRequestRetryHandler;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.impl.conn.PoolingHttpClientConnectionManager;
import org.apache.http.message.BasicHeader;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.context.annotation.Lazy;
import
org.springframework.http.client.HttpComponentsClientHttpRequestFactory;
import org.springframework.http.converter.FormHttpMessageConverter;
import org.springframework.http.converter.HttpMessageConverter;
import org.springframework.http.converter.StringHttpMessageConverter;
```

```
import
org.springframework.http.converter.json.MappingJackson2HttpMessageConverter;
import
org.springframework.http.converter.xml.MappingJackson2XmlHttpMessageConverte
r;
import org.springframework.stereotype.Component;
import org.springframework.web.client.DefaultResponseErrorHandler;
import org.springframework.web.client.RestTemplate;

import javax.annotation.PostConstruct;
import java.nio.charset.Charset;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.TimeUnit;

/**
* @title：使用 spring 的 restTemplate 替代 httpclient 工具
* @author：liuxing
* @date：2015-05-18 08:48
*/
public class RestClient {

private static final Logger LOGGER =
LoggerFactory.getLogger(SimpleRestClient.class);

private static RestTemplate restTemplate;

static {
// 长连接保持 30 秒
PoolingHttpClientConnectionManager pollingConnectionManager = new
PoolingHttpClientConnectionManager(30, TimeUnit.SECONDS);
// 总连接数
pollingConnectionManager.setMaxTotal(500);
// 同路由的并发数
pollingConnectionManager.setDefaultMaxPerRoute(500);

HttpClientBuilder httpClientBuilder = HttpClients.custom();
httpClientBuilder.setConnectionManager(pollingConnectionManager);
// 重试次数，默认是 3 次，没有开启
httpClientBuilder.setRetryHandler(new DefaultHttpRequestRetryHandler(2,
true));
// 保持长连接配置，需要在头添加 Keep-Alive
httpClientBuilder.setKeepAliveStrategy(DefaultConnectionKeepAliveStrategy.IN
STANCE);
```

```
List<Header> headers = new ArrayList<>();
headers.add(new BasicHeader("User-Agent", "Mozilla/5.0 (Windows NT 6.1)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/31.0.1650.16 Safari/537.36"));
headers.add(new BasicHeader("Accept-Encoding", "gzip,deflate"));
headers.add(new BasicHeader("Accept-Language", "zh-CN,zh;q=0.8,en;q=0.6"));
headers.add(new BasicHeader("Connection", "keep-alive"));

httpClientBuilder.setDefaultHeaders(headers);

HttpClient httpClient = httpClientBuilder.build();

// httpClient 连接配置，底层是配置 RequestConfig
HttpComponentsClientHttpRequestFactory clientHttpRequestFactory = new
HttpComponentsClientHttpRequestFactory(httpClient);
// 连接超时
clientHttpRequestFactory.setConnectTimeout(5000);
// 数据读取超时时间，即 SocketTimeout
clientHttpRequestFactory.setReadTimeout(5000);
// 连接不够用的等待时间，不宜过长，必须设置，比如连接不够用时，时间过长将是灾难性的
clientHttpRequestFactory.setConnectionRequestTimeout(200);
// 缓冲请求数据，默认值是 true。通过 POST 或者 PUT 大量发送数据时，建议将此属性更改为
false，以免耗尽内存。
// clientHttpRequestFactory.setBufferRequestBody(false);

// 添加内容转换器
List<HttpMessageConverter<?>> messageConverters = new ArrayList<>();
messageConverters.add(new StringHttpMessageConverter(Charset.forName("UTF-
8")));
messageConverters.add(new FormHttpMessageConverter());
messageConverters.add(new MappingJackson2XmlHttpMessageConverter());
messageConverters.add(new MappingJackson2HttpMessageConverter());
messageConverters.add(new ByteArrayHttpMessageConverter());

restTemplate = new RestTemplate(messageConverters);
restTemplate.setRequestFactory(clientHttpRequestFactory);
restTemplate.setErrorHandler(new DefaultResponseErrorHandler());

LOGGER.info("RestClient 初始化完成");
}

private RestClient() {

}
```

```
public static RestTemplate getClient() {
return restTemplate;
}

}
```

使用样例

注意点

api 里面可以做自动的参数匹配：
如：http://you domainn name/test?empNo={empNo}，则下面方法的最后一个参数为数据匹配参数，会自动根据 key 进行查找，然后替换
API 没有声明异常，注意进行异常处理
更多使用语法请查看 API 文档

完整的实例代码

定义一个异常

```
import org.springframework.core.NestedRuntimeException;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.web.client.HttpClientErrorException;
import org.springframework.web.client.HttpServerErrorException;

/**
* 包装一个 RestClient 请求时抛出的异常
*
* @author : liuxing
* @since : 2015-07-15 21:33
*/
public class RestClientException extends NestedRuntimeException {

/**
* 状态码
*/
private HttpStatus statusCode;
/**
* 状态码文本
*/
private String statusText;
/**
* 异常时返回的内容
*/
private String responseBody;
```

```
/**
* 返回的头
*/
private HttpHeaders responseHeaders;

public RestClientException(Exception exception) {
super(exception.getMessage(), exception);

if (exception instanceof HttpServerErrorException) {
HttpServerErrorException e = (HttpServerErrorException) exception;

this.statusCode = e.getStatusCode();
this.statusText = e.getStatusText();
this.responseBody = e.getResponseBodyAsString();
this.responseHeaders = e.getResponseHeaders();
} else if (exception instanceof HttpClientErrorException) {
HttpClientErrorException e = (HttpClientErrorException) exception;

this.statusCode = e.getStatusCode();
this.statusText = e.getStatusText();
this.responseBody = e.getResponseBodyAsString();
this.responseHeaders = e.getResponseHeaders();
} else {
this.statusText = exception.getMessage();
}
}

public HttpStatus getStatusCode() {
return statusCode;
}

public void setStatusCode(HttpStatus statusCode) {
this.statusCode = statusCode;
}

public String getStatusText() {
return statusText;
}

public void setStatusText(String statusText) {
this.statusText = statusText;
}

public String getResponseBody() {
```

```
return responseBody;
}

public void setResponseBody(String responseBody) {
this.responseBody = responseBody;
}

public HttpHeaders getResponseHeaders() {
return responseHeaders;
}

public void setResponseHeaders(HttpHeaders responseHeaders) {
this.responseHeaders = responseHeaders;
}
}
```

工具集

```
import com.dooioo.se.commons.Lang;
import com.dooioo.se.utils.RestClientBuilder;
import org.apache.commons.beanutils.BeanUtils;
import org.apache.commons.collections.MapUtils;
import org.springframework.core.ParameterizedTypeReference;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpMethod;
import org.springframework.http.MediaType;
import org.springframework.util.LinkedMultiValueMap;
import org.springframework.util.MultiValueMap;
import org.springframework.web.client.RestTemplate;

import java.lang.reflect.Field;
import java.util.Collection;
import java.util.HashMap;
import java.util.Map;

/**
* httpclient 工具类,基于 httpclient 4.x
* 不需要设置 header 的情况:
* 1.普通的非校验型请求
* 2.普通的表单请求
* <p/>
* 需要设置 header 的情况:
* 1.头部带 token 校验的请求
* 2.提交 json 数据的请求
*
```

```
* @author 刘兴
* @version V1.0
* @since 2014-3-7 下午 7:48:58
*/
public class RestClient {

/**
* 执行请求
*
* @param url 请求地址
* @param method 请求方式
* @param responseType 返回的数据类型
* @param uriVariables url 自动匹配替换的参数，如 url 为 api/{a}/{b},参数为
["1","2"],则解析的 url 为 api/1/2，使用 Map 参数时，遵循按 key 匹配
* @return 结果对象
* @throws RestClientException RestClient 异常，包含状态码和非 200 的返回内容
*/
public static <T> T exchange(String url, HttpMethod method, Class<T>
responseType, Object... uriVariables) throws RestClientException {
return exchange(url, method, null, null, responseType, uriVariables);
}

/**
* 执行请求
*
* @param url 请求地址
* @param method 请求方式
* @param headers 设置的头信息
* @param responseType 返回的数据类型
* @param uriVariables url 自动匹配替换的参数，如 url 为 api/{a}/{b},参数为
["1","2"],则解析的 url 为 api/1/2，使用 Map 参数时，遵循按 key 匹配
* @return 结果对象
* @throws RestClientException RestClient 异常，包含状态码和非 200 的返回内容
*/
public static <T> T exchange(String url, HttpMethod method, HttpHeaders
headers, Class<T> responseType, Object... uriVariables) throws
RestClientException {
return exchange(url, method, headers, null, responseType, uriVariables);
}

/**
* 执行请求
*
* @param url 请求地址
```

```
* @param method 请求方式
* @param body 要提交的数据
* @param responseType 返回数据类型
* 返回 bean 时指定 Class
* @param uriVariables url 自动匹配替换的参数，如 url 为 api/{a}/{b},参数为
["1","2"],则解析的 url 为 api/1/2，使用 Map 参数时，遵循按 key 匹配
* @return 结果对象
* @throws RestClientException RestClient 异常，包含状态码和非 200 的返回内容
*/
public static <T> T exchange(String url, HttpMethod method, Object body,
Class<T> responseType, Object... uriVariables) throws RestClientException {
return exchange(url, method, null, body, responseType, uriVariables);
}

/**
* 执行请求
*
* @param url 请求地址
* @param method 请求方式
* @param httpHeaders 请求头
* @param body 要提交的数据
* @param responseType 返回数据类型
* 返回 bean 时指定 Class
* @param uriVariables url 自动匹配替换的参数，如 url 为 api/{a}/{b},参数为
["1","2"],则解析的 url 为 api/1/2，使用 Map 参数时，遵循按 key 匹配
* @return 结果对象
* @throws RestClientException RestClient 异常，包含状态码和非 200 的返回内容
*/
public static <T> T exchange(String url, HttpMethod method, HttpHeaders
httpHeaders, Object body, Class<T> responseType, Object... uriVariables)
throws RestClientException {
try {
HttpEntity<?> requestEntity = new HttpEntity(body, httpHeaders);
requestEntity = convert(requestEntity);

if (uriVariables.length == 1 && uriVariables[0] instanceof Map) {
Map<String, ?> _uriVariables = (Map<String, ?>) uriVariables[0];
return getClient().exchange(url, method, requestEntity, responseType,
_uriVariables).getBody();
}

return getClient().exchange(url, method, requestEntity, responseType,
uriVariables).getBody();
} catch (Exception e) {
```

```
throw new RestClientException(e);
}
}

/**
* 执行请求
*
* @param url 请求地址
* @param method 请求方式
* @param responseType 返回的数据类型，例：new
ParameterizedTypeReference<List<Bean>>(){}
* @param uriVariables url 自动匹配替换的参数，如 url 为 api/{a}/{b},参数为
["1","2"],则解析的 url 为 api/1/2，使用 Map 参数时，遵循按 key 匹配
* @return 结果对象
* @throws RestClientException RestClient 异常，包含状态码和非 200 的返回内容
*/
public static <T> T exchange(String url, HttpMethod method,
ParameterizedTypeReference<T> responseType, Object... uriVariables) throws
RestClientException {
return exchange(url, method, null, null, responseType, uriVariables);
}

/**
* 执行请求
*
* @param url 请求地址
* @param method 请求方式
* @param headers 设置的头信息
* @param responseType 返回的数据类型，例：new
ParameterizedTypeReference<List<Bean>>(){}
* @param uriVariables url 自动匹配替换的参数，如 url 为 api/{a}/{b},参数为
["1","2"],则解析的 url 为 api/1/2，使用 Map 参数时，遵循按 key 匹配
* @return 结果对象
* @throws RestClientException RestClient 异常，包含状态码和非 200 的返回内容
*/
public static <T> T exchange(String url, HttpMethod method, HttpHeaders
headers, ParameterizedTypeReference<T> responseType, Object... uriVariables)
throws RestClientException {
return exchange(url, method, headers, null, responseType, uriVariables);
}

/**
* 执行请求
*
```

```
* @param url 请求地址
* @param method 请求方式
* @param body 要提交的数据
* @param responseType 返回数据类型，例：new
ParameterizedTypeReference<List<Bean>>(){}
* 返回 bean 时指定 Class
* @param uriVariables url 自动匹配替换的参数，如 url 为 api/{a}/{b},参数为
["1","2"],则解析的 url 为 api/1/2，使用 Map 参数时，遵循按 key 匹配
* @return 结果对象
* @throws RestClientException RestClient 异常，包含状态码和非 200 的返回内容
*/
public static <T> T exchange(String url, HttpMethod method, Object body,
ParameterizedTypeReference<T> responseType, Object... uriVariables) throws
RestClientException {
return exchange(url, method, null, body, responseType, uriVariables);
}

/**
* 执行请求
*
* @param url 请求地址
* @param method 请求方式
* @param httpHeaders 请求头
* @param body 要提交的数据
* @param responseType 返回数据类型，例：new
ParameterizedTypeReference<List<Bean>>(){}
* 返回 bean 时指定 Class
* @param uriVariables url 自动匹配替换的参数，如 url 为 api/{a}/{b},参数为
["1","2"],则解析的 url 为 api/1/2，使用 Map 参数时，遵循按 key 匹配
* @return 结果对象
* @throws RestClientException RestClient 异常，包含状态码和非 200 的返回内容
*/
public static <T> T exchange(String url, HttpMethod method, HttpHeaders
httpHeaders, Object body, ParameterizedTypeReference<T> responseType,
Object... uriVariables) throws RestClientException {
try {
HttpEntity<?> requestEntity = new HttpEntity(body, httpHeaders);
requestEntity = convert(requestEntity);

if (uriVariables.length == 1 && uriVariables[0] instanceof Map) {
Map<String, ?> _uriVariables = (Map<String, ?>) uriVariables[0];
return getClient().exchange(url, method, requestEntity, responseType,
_uriVariables).getBody();
}
```

```
return getClient().exchange(url, method, requestEntity, responseType,
uriVariables).getBody();
} catch (Exception e) {
throw new RestClientException(e);
}
}

/**
* 获得一个RestTemplate客户端
*
* @return
*/
public static RestTemplate getClient() {
return RestClientBuilder.build();
}

/**
* 获取一个application/x-www-form-urlencoded头
*
* @return
*/
public static HttpHeaders buildBasicFORMHeaders() {
HttpHeaders headers = new HttpHeaders();
headers.setContentType(MediaType.APPLICATION_FORM_URLENCODED);
return headers;
}

/**
* 获取一个application/json头
*
* @return
*/
public static HttpHeaders buildBasicJSONHeaders() {
HttpHeaders headers = new HttpHeaders();
headers.setContentType(MediaType.APPLICATION_JSON);
return headers;
}

/**
* 获取一个text/html头
*
* @return
*/
```

```
public static HttpHeaders buildBasicHTMLHeaders() {
HttpHeaders headers = new HttpHeaders();
headers.setContentType(MediaType.TEXT_HTML);
return headers;
}

/**
* 构建一个json头
*
* @param arrays
* @return
*/
public static HttpHeaders buildJSONHeaders(Object... arrays) {
if (arrays.length % 2 != 0) {
throw new RuntimeException("arrays 长度 必须为偶数");
}

HttpHeaders headers = buildBasicJSONHeaders();

for (int i = 0; i < arrays.length; i++) {
headers.add(Lang.defaultEmptyStr(arrays[i]),
Lang.defaultEmptyStr(arrays[++i]));
}

return headers;
}

/**
* 对bean对象转表单模型做处理
*
* @param requestEntity
* @return
*/
private static HttpEntity<?> convert(HttpEntity<?> requestEntity) {
Object body = requestEntity.getBody();
HttpHeaders headers = requestEntity.getHeaders();

if (body == null) {
return requestEntity;
}

if (body instanceof Map) {
MultiValueMap<String, String> multiValueMap = new LinkedMultiValueMap<>();
Map<String, ?> _body = (Map<String, ?>) body;
```

```
for (String key : _body.keySet()) {
multiValueMap.add(key, MapUtils.getString(_body, key));
}

requestEntity = new HttpEntity<>(multiValueMap, headers);
}

if (headers == null
|| !MediaType.APPLICATION_FORM_URLENCODED.equals(headers.getContentType()))
{
return requestEntity;
}

if (body instanceof String) {
return requestEntity;
}

if (body instanceof Collection) {
return requestEntity;
}

if (body instanceof Map) {
return requestEntity;
}

MultiValueMap<String, Object> formEntity = new LinkedMultiValueMap<>();

Field[] fields = body.getClass().getDeclaredFields();
for (int i = 0; i < fields.length; i++) {
String name = fields[i].getName();
String value = null;

try {
value = BeanUtils.getProperty(body, name);
} catch (Exception e) {
e.printStackTrace();
}

formEntity.add(name, value);
}

return new HttpEntity<>(formEntity, headers);
}
```

```
public final static Object[] EMPTY_URI_VARIABLES = new Object[]{};

public final static HttpHeaders EMPTY_HEADERS = new HttpHeaders();

public final static Map<String, ?> EMPTY_BODY = new HashMap<>(1);

public final static HttpEntity EMPTY_ENTITY = new HttpEntity(EMPTY_HEADERS);

}
```

更多

RestTemplate API 说明和使用参考

http://docs.spring.io/spring/docs/4.1.x/javadoc-api/org/springframework/web/client/RestTemplate.html

http://docs.spring.io/spring/docs/4.1.x/javadoc-api/org/springframework/http/client/SimpleClientHttpRequestFactory.html

http://docs.spring.io/spring/docs/4.1.x/javadoc-api/org/springframework/http/client/HttpComponentsClientHttpRequestFactory.html

HttpClient 官方示例和参数配置说明

http://hc.apache.org/httpcomponents-client-4.4.x/examples.html

http://hc.apache.org/httpcomponents-client-4.4.x/tutorial/html/index.html

依赖

spring 3.x 以上

```
<dependency>
<groupId>com.fasterxml.jackson.dataformat</groupId>
<artifactId>jackson-dataformat-xml</artifactId>
<version>2.5.3</version>
</dependency>

<dependency>
<groupId>org.codehaus.jackson</groupId>
<artifactId>jackson-mapper-asl</artifactId>
<version>1.9.13</version>
</dependency>
```

注意点

1.关于 httpclient 配置的 `defaultMaxPerRoute` 和 `maxTotal`

`defaultMaxPerRoute`：最大路由并发数，以主机为单位

`maxTotal`：整个连接池的并发数

例如：

`defaultMaxPerRoute` 为 10，`maxTotal` 为 100

那么能同时并发到客源的只能是 10，房源也是 10，整个连接永远不会到 100

2.部分方法注意查看源码，默认构造里面会新增常用的数据转换器，spring 对 jackson 比较情有独钟，在解析 xml 和 json 时，优先使用 jackson

```
/**
* Create a new instance of the {@link RestTemplate} using default settings.
* Default {@link HttpMessageConverter}s are initialized.
*/
public RestTemplate() {
this.messageConverters.add(new ByteArrayHttpMessageConverter());
this.messageConverters.add(new StringHttpMessageConverter());
this.messageConverters.add(new ResourceHttpMessageConverter());
this.messageConverters.add(new SourceHttpMessageConverter<Source>());
this.messageConverters.add(new AllEncompassingFormHttpMessageConverter());

if (romePresent) {
this.messageConverters.add(new AtomFeedHttpMessageConverter());
this.messageConverters.add(new RssChannelHttpMessageConverter());
}
if (jackson2XmlPresent) {
messageConverters.add(new MappingJackson2XmlHttpMessageConverter());
}
else if (jaxb2Present) {
this.messageConverters.add(new Jaxb2RootElementHttpMessageConverter());
}
if (jackson2Present) {
this.messageConverters.add(new MappingJackson2HttpMessageConverter());
}
else if (gsonPresent) {
this.messageConverters.add(new GsonHttpMessageConverter());
}
}

/**
* Create a new instance of the {@link RestTemplate} based on the given
{@link ClientHttpRequestFactory}.
* @param requestFactory HTTP request factory to use
* @see org.springframework.http.client.SimpleClientHttpRequestFactory
* @see
org.springframework.http.client.HttpComponentsClientHttpRequestFactory
*/
public RestTemplate(ClientHttpRequestFactory requestFactory) {
this();
setRequestFactory(requestFactory);
}
```

再看添加转换器的方法外部添加转换器时，this.messageConverters.clear();会先清除已有的，需要注意

```
/**
* Create a new instance of the {@link RestTemplate} using the given list of
* {@link HttpMessageConverter} to use
* @param messageConverters the list of {@link HttpMessageConverter} to use
* @since 3.2.7
*/
public RestTemplate(List<HttpMessageConverter<?>> messageConverters) {
Assert.notEmpty(messageConverters, "'messageConverters' must not be empty");
this.messageConverters.addAll(messageConverters);
}


/**
* Set the message body converters to use.
* <p>These converters are used to convert from and to HTTP requests and
responses.
*/
public void setMessageConverters(List<HttpMessageConverter<?>>
messageConverters) {
Assert.notEmpty(messageConverters, "'messageConverters' must not be empty");
// Take getMessageConverters() List as-is when passed in here
if (this.messageConverters != messageConverters) {
this.messageConverters.clear();
this.messageConverters.addAll(messageConverters);
}
}
```