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。

---

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.FormHttpMessageConvert
er"/>
                <bean
class="org.springframework.http.converter.xml.MappingJackson2Xml
HttpMessageConverter"/>
                <bean
class="org.springframework.http.converter.json.MappingJackson2Ht
tpMessageConverter"/>
                <bean
class="org.springframework.http.converter.StringHttpMessageConve
rter">
                    <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.PoolingHttpClientConnectionMana
ger">
        <!--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.DefaultHttpRequestRetryHandle
r">
                <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.HttpComponentsClientHttpR
equestFactory">
        <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.DefaultResponseErrorHandle
r"/>
        </property>
        <property name="messageConverters">
            <list>
                <bean
class="org.springframework.http.converter.FormHttpMessageConvert
er"/>
                <bean
class="org.springframework.http.converter.xml.MappingJackson2Xml
HttpMessageConverter"/>
                <bean
class="org.springframework.http.converter.json.MappingJackson2Ht
tpMessageConverter"/>
                <bean
class="org.springframework.http.converter.StringHttpMessageConve
rter">
                    <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.MappingJackson2HttpMessa
geConverter;
import
org.springframework.http.converter.xml.MappingJackson2XmlHttpMes
sageConverter;
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.HttpComponentsClientHttpRequestF
actory;
import
org.springframework.http.converter.FormHttpMessageConverter;
import org.springframework.http.converter.HttpMessageConverter;
import
org.springframework.http.converter.StringHttpMessageConverter;
import
org.springframework.http.converter.json.MappingJackson2HttpMessa
geConverter;
import
org.springframework.http.converter.xml.MappingJackson2XmlHttpMes
sageConverter;
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
 */
@Component
@Lazy(false)
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(1000);
        // 同路由的并发数
        pollingConnectionManager.setDefaultMaxPerRoute(1000);

        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(new
DefaultConnectionKeepAliveStrategy());

//        RequestConfig.Builder builder =
RequestConfig.custom();
//        builder.setConnectionRequestTimeout(200);
//        builder.setConnectTimeout(5000);
//        builder.setSocketTimeout(5000);
//
//        RequestConfig requestConfig = builder.build();
//
httpClientBuilder.setDefaultRequestConfig(requestConfig);

        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"));
        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());

        restTemplate = new RestTemplate(messageConverters);

restTemplate.setRequestFactory(clientHttpRequestFactory);
        restTemplate.setErrorHandler(new
DefaultResponseErrorHandler());

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

    private RestClient() {

    }
```

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

}
```

HttpClientUtils

```
import com.dooioo.commons.Strings;
import com.dooioo.framework.SpringContextHolder;
import com.dooioo.ky.cache.HttpClientResultCache;
import org.apache.commons.collections.MapUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.http.HttpEntity;
import org.springframework.util.LinkedMultiValueMap;
import org.springframework.util.MultiValueMap;

import java.util.Map;

/**
 *
 * 类功能说明：httpclient工具类,基于httpclient 4.x
 * Title: HttpClientUtils.java
 * @author 刘兴
 * @date 2014-3-7 下午7:48:58
 * @version V1.0
 */
public class HttpClientUtils {

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

    /**
     * post请求
     * @param url
     * @param formParams
     * @return
     */
    public static String doPost(String url, Map<String, String>
formParams) {
        if (MapUtils.isEmpty(formParams)) {
            return doPost(url);
```

```
        }

        try {
            MultiValueMap<String, String> requestEntity = new
LinkedMultiValueMap<>();
            formParams.keySet().stream().forEach(key ->
requestEntity.add(key, MapUtils.getString(formParams, key,
"")));
            return RestClient.getClient().postForObject(url,
requestEntity, String.class);
        } catch (Exception e) {
            LOGGER.error("POST请求出错：{}", url, e);
        }

        return Strings.EMPTY;
    }

    /**
     * post请求
     * @param url
     * @return
     */
    public static String doPost(String url) {
        try {
            return RestClient.getClient().postForObject(url,
HttpEntity.EMPTY, String.class);
        } catch (Exception e) {
            LOGGER.error("POST请求出错：{}", url, e);
        }

        return Strings.EMPTY;
    }

    /**
     * get请求
     * @param url
     * @return
     */
    public static String doGet(String url) {
        try {
            return RestClient.getClient().getForObject(url,
String.class);
        } catch (Exception e) {
            LOGGER.error("GET请求出错：{}", url, e);
        }
```

```
        return Strings.EMPTY;
    }

}
```

ErrorHolder

自定义的一个异常结果包装类

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

/**
 * @title: ErrorHolder
 * @author: liuxing
 * @date: 2015-06-16 11:01
 */
public class ErrorHolder {

    private HttpStatus statusCode;

    private String statusText;

    private String responseBody;

    private HttpHeaders responseHeaders;

    public ErrorHolder(HttpStatus statusCode, String statusText,
String responseBody) {
        this.statusCode = statusCode;
        this.statusText = statusText;
        this.responseBody = responseBody;
    }

    public ErrorHolder(String statusText) {
        this.statusText = statusText;
    }

    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;
    }

    public static ErrorHolder build(Exception exception) {
        if (exception instanceof HttpServerErrorException) {
            HttpServerErrorException e =
(HttpServerErrorException) exception;
            return new ErrorHolder(e.getStatusCode(),
e.getStatusText(), e.getResponseBodyAsString());
        }

        if (exception instanceof HttpClientErrorException) {
            HttpClientErrorException e =
(HttpClientErrorException) exception;
            return new ErrorHolder(e.getStatusCode(),
e.getStatusText(), e.getResponseBodyAsString());
        }

        return new ErrorHolder(exception.getMessage());
```

```
    }
}
```

使用样例

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

```
ResponseEntity<List<KyArea>> result =
RestClient.getClient().exchange(DIVIDE_PLATE_API,
HttpMethod.GET, HttpEntity.EMPTY, new
ParameterizedTypeReference<List<KyArea>>() {}, map("empNo",
empNo));
List<KyArea> list = result.getBody();

ResponseEntity<KyArea> result =
RestClient.getClient().exchange(DIVIDE_PLATE_API,
HttpMethod.GET, HttpEntity.EMPTY, KyArea.class, map("empNo",
empNo));
KyArea kyArea = result.getBody();
```

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.HttpComponentsClientHttpRequestF
actory
 */
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);
    }
}
```