Asimio Tech

Helping Java developers to better use Spring Framework, Spring Boot and Spring Cloud projects

Home About Archives Tags Subscribe to Atom Feed Disclaimer

Search Asimio Tech posts

Caching using RestTemplate, Ehcache and ETags

Orlando L Otero | Jul 14, 2017 | api, caching, ehcache, java, restful, resttemplate, spring boot | 12 min read | 4 Comments

This post has been featured on http://www.baeldung.com/java-weekly-186.

Table of Contents

1. INTRODUCTION

Often times I have seen `API` implementations not taking advantage of client side caching. Consider this example, a `REST` service needs to get data from a handful of other services and for every request, even though the upstream response might have not changed for the same input, it's being calculated repeatedly and sent back to the client.

Depending on how expensive this calculation might be, wouldn't be a better approach if the `HTTP` request includes data about what it previously has stored from a prior server response in an attempt for the server to find out if this calculation would be needed at all? This will improve the application performance while saving on server

resources.
And what about if this expensive calculation is not needed, wouldn't be a good practice for the server to let the client know that ***nothing has changed*** on the server side for that request? This will also save on bandwidth, assuming the client service is able to reconstruct the response payload.

This post focuses on the client side of this improvement, configuring `Spring`'s `RestTemplate` to use `HttpClient` and `Ehcache` to cache upstream HTTP responses using `ETags`.

2. REQUIREMENTS

- `Java` 7+.
- `Maven` 3.2+.
- Familiarity with `Spring` Framework.

3. THE DEMO SERVICE 2

This service includes a simple `API` returning a `String`. As part of the `HTTP` response, the `ETag` header value will be set to the `md5` hash of the entity representation (the response body in this demo) via the `Spring`'s ShallowEtagHeaderFilter.
Basically this means the `ETag` header value will change for different `String` responses.

Let's discuss the relevant parts of the *Demo Service 2*:

- *pom.xml* :

```
...
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
...
```

`spring-boot-starter-web` dependency will be used to implement a `RESTful API` using `Spring`.

- *Demo2CachingRestTemplateApplication.java* :

```
package com.asimio.api.demo.main;
...
@SpringBootApplication(scanBasePackages = { "com.asimio.api.demo" })
public class Demo2CachingRestTemplateApplication {
    public static void main(String[] args) {
      SpringApplication.run(Demo2CachingRestTemplateApplication.class, args);
    }

    @Bean
    public Filter shallowEtagHeaderFilter() {
      return new ShallowEtagHeaderFilter();
    }

    @Bean
    public FilterRegistrationBean shallowEtagHeaderFilterRegistration() {
      FilterRegistrationBean result = new FilterRegistrationBean();
      result.setFilter(this.shallowEtagHeaderFilter());
      result.addUrlPatterns("/api/*");
      result.setName("shallowEtagHeaderFilter");
      result.setOrder(1);
      return result;
    }
  }
```

This is the `Spring Boot` app's start class as defined in * `pom.xml*` 's `start-class` property. It's also taking care if registering the ShallowEtagHeaderFilter filter mentioned earlier. It's worth mentioning that:

Important: ShallowEtagHeaderFilter implementation saves on bandwidth because if the `If-None-Match` header passed in the request matches the `ETag` header value to be included in the response, the body won't be included but the `HTTP` status `304 - Not Modified`.
It could be noticed that since the `ETag`'s `md5` hash value is calculated for every request, it doesn't save on server performance, which it's OK to keep this how-to simple.

Note: `ETags` are used for caching and conditional requests.

Tip: A more realistic example to demonstrate saving on server performance would be using `ETags` and conditional requests with `JPA` entities, `@Version` field and `Second-Level` cache.

Stay tuned and sign up to the newsletter to receive updates when content like this is published.

- `HelloResource.java` :

```
package com.asimio.api.demo.rest;
...
@RestController
@RequestMapping(value = "/api/hello")
public class HelloResource {

  // Shallow implementation, saves bandwidth but doesn't save server resources.
  @RequestMapping(value = "/{name}", method = RequestMethod.GET)
  public String getHello(@PathVariable("name") String name) {
    return String.format("%s %s", "Hello", name);
  }
}
```

A simple implementation of an endpoint to be used by Demo Service 1.

4. THE DEMO SERVICE 1

This service implements a simple `API` that uses `RestTemplate` to delegate requests to Demo Service 2 demonstrating how to configure it using `HttpClient` and `Ehcache` to cache responses using `ETags`. This approach saves us from explicitly caching, updating and evicting objects, managing TTLs, etc. with the associated overhead related to thread safety.

The relevant parts of the *Demo Service 1* are:

- `pom.xml` :

```
...
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-cache</artifactId>
</dependency>
<dependency>
```

```
    <groupId>net.sf.ehcache</groupId>
    <artifactId>ehcache</artifactId>
  </dependency>
  <dependency>
    <groupId>org.apache.httpcomponents</groupId>
    <artifactId>httpclient</artifactId>
  </dependency>
  <dependency>
    <groupId>org.apache.httpcomponents</groupId>
    <artifactId>httpclient-cache</artifactId>
    <version>${httpclient.version}</version>
  </dependency>
  ...
```

Similarly to Demo Service 2, `spring-boot-starter-web` dependency is included to implement an `API` using `Spring MVC` RESTful.
`spring-boot-starter-cache` is a `Spring Boot` starter responsible for creating `Caching`-related beans depending on classes found in the `classpath`, for instance `ehcache`, the cache provider in this tutorial.
`httpclient` library is used as the underlying library used by `RestTemplate` to send outbound requests and `httpclient-cache` is used to provide support for `httpclient` to cache responses.

- *Demo1CachingRestTemplateApplication.java* :

```java
package com.asimio.api.demo.main;
...
@SpringBootApplication(scanBasePackages = { "com.asimio.api.demo.main", "
@EnableCaching // for cacheManager and related beans to get auto-configur
public class Demo1CachingRestTemplateApplication {

  @Value("#{cacheManager.getCache('httpClient')}")
  private Cache httpClientCache;

  public static void main(String[] args) {
    SpringApplication.run(Demo1CachingRestTemplateApplication.class, args
  }

  @Bean
```

```
    public PoolingHttpClientConnectionManager poolingHttpClientConnectionMa
      PoolingHttpClientConnectionManager result = new PoolingHttpClientConn
      result.setMaxTotal(20);
      return result;
    }

    @Bean
    public CacheConfig cacheConfig() {
      CacheConfig result = CacheConfig
        .custom()
        .setMaxCacheEntries(DEFAULT_MAX_CACHE_ENTRIES)
        .build();
      return result;
    }

    @Bean
    public HttpCacheStorage httpCacheStorage() {
      Ehcache ehcache = (Ehcache) this.httpClientCache.getNativeCache();
      HttpCacheStorage result = new EhcacheHttpCacheStorage(ehcache);
      return result;
    }

    @Bean
    public HttpClient httpClient(PoolingHttpClientConnectionManager pooling
      CacheConfig cacheConfig, HttpCacheStorage httpCacheStorage) {

      HttpClient result = CachingHttpClientBuilder
        .create()
        .setCacheConfig(cacheConfig)
        .setHttpCacheStorage(httpCacheStorage)
        .disableRedirectHandling()
        .setConnectionManager(poolingHttpClientConnectionManager)
        .build();
      return result;
    }

    @Bean
    public RestTemplate restTemplate(HttpClient httpClient) {
      HttpComponentsClientHttpRequestFactory requestFactory = new HttpCompo
      requestFactory.setHttpClient(httpClient);
      return new RestTemplate(requestFactory);
    }
  ...
```

First the `@EnableCaching` allows the cacheManager to be auto-configured.

`PoolingHttpClientConnectionManager`, `HttpClient` and `RestTemplate` beans look similar to the ones included in Troubleshooting Spring's RestTemplate Requests Timeout except that in this post the `HttpClient` object is instantiated using `CachingHttpClientBuilder` while in the other post the `HttpClient` bean was instantiated using `HttpClientBuilder`. \

Basically this three beans are used to configure the `RestTemplate` bean to use `Apache HttpClient` instead of the default implementation which is based on the `JDK` plus some basic configuration such as the number of connections in the pool.

It's also worth mentioning `httpClient` reference in line 10 refers to the cache name as found in *`ehcache.xml`* .

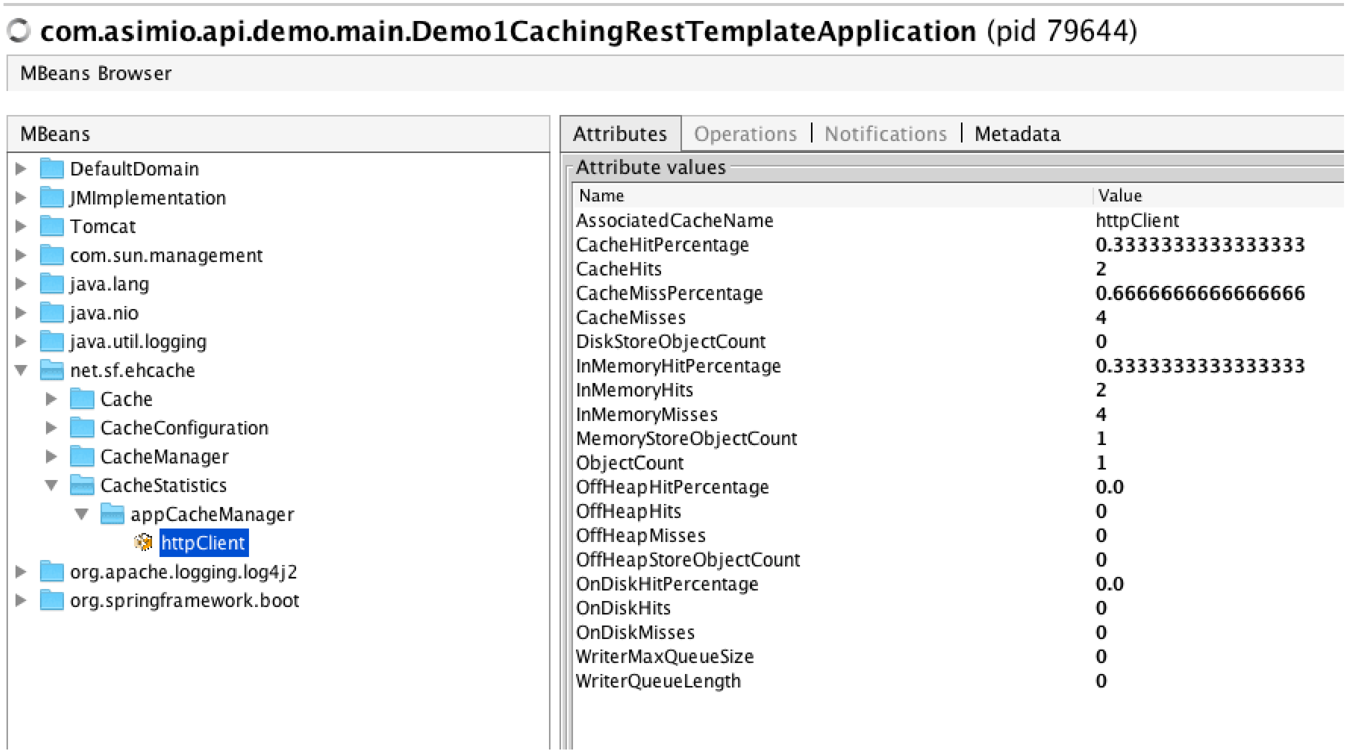
Tip: Be aware of configuring the HttpClient timeouts appropriately and a few HttpClient connection pool manager properties whose default values might affect performance.

The other interesting beans are `CacheConfig` and `HttpCacheStorage`. `HttpCacheStorage` is in fact not required to provide client side caching. If it were removed (along with the `@EnableCaching` annotation), a `BasicHttpCacheStorage` or `ManagedHttpCacheStorage` implementation would be used instead as explained in the next table with configuration set in the `CacheConfig` bean.

HttpCacheStorage implementations	Description
BasicHttpCacheStorage	Default implementation if *cacheDir* is set when instantiating an `HttpClient` instance via *`CachingHttpClientBuilder`* .
EhcacheHttpCacheStorage	Discussed in this post, uses `Ehcache` as the backend.
ManagedHttpCacheStorage	Default implementation if *cacheDir* is not set when instantiating an `HttpClient` instance via *`CachingHttpClientBuilder`* .

MemcacheHttpCacheStorage	Uses Memcache as the backend.

But using `EhcacheHttpCacheStorage` allows for more configuration settings, the application might already be using `Ehcache` and its statistics, data and operations could be accessed via `JMX MBean`.



Name	Value
AssociatedCacheName	httpClient
CacheHitPercentage	0.3333333333333333
CacheHits	2
CacheMissPercentage	0.6666666666666666
CacheMisses	4
DiskStoreObjectCount	0
InMemoryHitPercentage	0.3333333333333333
InMemoryHits	2
InMemoryMisses	4
MemoryStoreObjectCount	1
ObjectCount	1
OffHeapHitPercentage	0.0
OffHeapHits	0
OffHeapMisses	0
OffHeapStoreObjectCount	0
OnDiskHitPercentage	0.0
OnDiskHits	0
OnDiskMisses	0
WriterMaxQueueSize	0
WriterQueueLength	0

JMX - MBeans - Ehcache Stats

- *`ehcache.xml`* :

```
...
<cache
  name="httpClient"
  maxElementsInMemory="10"
  timeToLiveSeconds="86400"
  eternal="false"
  overflowToDisk="false" />
...
```

The cache configuration used to store the `HTTP` responses.

- *`HelloResource.java`* :

```
package com.asimio.api.demo.rest;
...
@RestController
```

```
@RequestMapping(value = "/api/hello")
public class HelloResource {

  @Autowired
  private RestTemplate restTemplate;

  @RequestMapping(value = "/{name}", method = RequestMethod.GET)
  public String getHello(@PathVariable(value = "name") String name) {
    ResponseEntity<String> response = this.restTemplate.getForEntity("http://loca
    return response.getBody();
  }
}
```

This is a sample `API` that sends requests to another web service where caching details are completely transparent to the application. This code doesn't need to update the cache or evict items, etc.. In fact, it doesn't know values might have been retrieved from a cache.

5. RUNNING THE SERVICES

Let's send an HTTP request to Demo Service 1, which in turn sends a request to Demo Service 2:

```
curl -v "http://localhost:8090/api/hello/orlando"
*   Trying ::1...
* TCP_NODELAY set
* Connected to localhost (::1) port 8090 (#0)
> GET /api/hello/orlando HTTP/1.1
> Host: localhost:8090
> User-Agent: curl/7.51.0
> Accept: */*
>
< HTTP/1.1 200
< X-Application-Context: application:8090
< Content-Type: text/plain;charset=UTF-8
< Content-Length: 13
< Date: Tue, 11 Jul 2017 11:24:56 GMT
<
* Curl_http_done: called premature == 0
* Connection #0 to host localhost left intact
Hello orlando
```

A successful *200 OK* response, but let’s look at the logs generated when Demo Service 1 sends the request to Demo Service 2:

```
2017-07-11 07:24:56 DEBUG RestTemplate:87 - Created GET request for "http://loca
2017-07-11 07:24:56 DEBUG RestTemplate:779 - Setting request Accept header to [t
2017-07-11 07:24:56 DEBUG RequestAddCookies:123 - CookieSpec selected: default
2017-07-11 07:24:56 DEBUG RequestAuthCache:77 - Auth cache not set in the contex
2017-07-11 07:24:56 DEBUG CachingExec:275 - Cache miss
2017-07-11 07:24:56 DEBUG PoolingHttpClientConnectionManager:255 - Connection re
2017-07-11 07:24:56 DEBUG PoolingHttpClientConnectionManager:288 - Connection le
2017-07-11 07:24:56 DEBUG MainClientExec:235 - Opening connection {}->http://loc
2017-07-11 07:24:56 DEBUG DefaultHttpClientConnectionOperator:139 - Connecting t
2017-07-11 07:24:56 DEBUG DefaultHttpClientConnectionOperator:146 - Connection e
2017-07-11 07:24:56 DEBUG MainClientExec:256 - Executing request GET /api/hello/
2017-07-11 07:24:56 DEBUG MainClientExec:261 - Target auth state: UNCHALLENGED
2017-07-11 07:24:56 DEBUG MainClientExec:267 - Proxy auth state: UNCHALLENGED
2017-07-11 07:24:56 DEBUG headers:133 - http-outgoing-0 >> GET /api/hello/orland
2017-07-11 07:24:56 DEBUG headers:136 - http-outgoing-0 >> Accept: text/plain, a
2017-07-11 07:24:56 DEBUG headers:136 - http-outgoing-0 >> Host: localhost:8080
2017-07-11 07:24:56 DEBUG headers:136 - http-outgoing-0 >> Connection: Keep-Aliv
2017-07-11 07:24:56 DEBUG headers:136 - http-outgoing-0 >> User-Agent: Apache-Ht
2017-07-11 07:24:56 DEBUG headers:136 - http-outgoing-0 >> Accept-Encoding: gzip
2017-07-11 07:24:56 DEBUG headers:136 - http-outgoing-0 >> Via: 1.1 localhost (A
2017-07-11 07:24:56 DEBUG wire:73 - http-outgoing-0 >> "GET /api/hello/orlando H
2017-07-11 07:24:56 DEBUG wire:73 - http-outgoing-0 >> "Accept: text/plain, appl
2017-07-11 07:24:56 DEBUG wire:73 - http-outgoing-0 >> "Host: localhost:8080[\r]
2017-07-11 07:24:56 DEBUG wire:73 - http-outgoing-0 >> "Connection: Keep-Alive[\
2017-07-11 07:24:56 DEBUG wire:73 - http-outgoing-0 >> "User-Agent: Apache-HttpC
2017-07-11 07:24:56 DEBUG wire:73 - http-outgoing-0 >> "Accept-Encoding: gzip,de
2017-07-11 07:24:56 DEBUG wire:73 - http-outgoing-0 >> "Via: 1.1 localhost (Apac
2017-07-11 07:24:56 DEBUG wire:73 - http-outgoing-0 >> "[\r][\n]"
2017-07-11 07:24:56 DEBUG wire:73 - http-outgoing-0 << "HTTP/1.1 200 [\r][\n]"
2017-07-11 07:24:56 DEBUG wire:73 - http-outgoing-0 << "X-Application-Context: a
2017-07-11 07:24:56 DEBUG wire:73 - http-outgoing-0 << "ETag: "023e8caa26fd74114
2017-07-11 07:24:56 DEBUG wire:73 - http-outgoing-0 << "Content-Type: text/plain
2017-07-11 07:24:56 DEBUG wire:73 - http-outgoing-0 << "Content-Length: 13[\r][\
2017-07-11 07:24:56 DEBUG wire:73 - http-outgoing-0 << "Date: Tue, 11 Jul 2017 1
2017-07-11 07:24:56 DEBUG wire:73 - http-outgoing-0 << "[\r][\n]"
2017-07-11 07:24:56 DEBUG wire:87 - http-outgoing-0 << "Hello orlando"
```

```
2017-07-11 07:24:56 DEBUG headers:122 - http-outgoing-0 << HTTP/1.1 200
2017-07-11 07:24:56 DEBUG headers:125 - http-outgoing-0 << X-Application-Context
2017-07-11 07:24:56 DEBUG headers:125 - http-outgoing-0 << ETag: "023e8caa26fd741
2017-07-11 07:24:56 DEBUG headers:125 - http-outgoing-0 << Content-Type: text/pla
2017-07-11 07:24:56 DEBUG headers:125 - http-outgoing-0 << Content-Length: 13
2017-07-11 07:24:56 DEBUG headers:125 - http-outgoing-0 << Date: Tue, 11 Jul 2017
2017-07-11 07:24:56 DEBUG MainClientExec:285 - Connection can be kept alive indef
2017-07-11 07:24:56 DEBUG PoolingHttpClientConnectionManager:320 - Connection [id
2017-07-11 07:24:56 DEBUG PoolingHttpClientConnectionManager:326 - Connection rel
2017-07-11 07:24:56 DEBUG RestTemplate:691 - GET request for "http://localhost:80
2017-07-11 07:24:56 DEBUG RestTemplate:102 - Reading [java.lang.String] as "text/
```

The interesting logs here are the *200 OK* response from Demo Service 2 which also includes the header *ETag: "023e8caa26fd7411445527af3d9aed055"* and *Hello orlando* in the body.
023e8caa26fd7411445527af3d9aed055 being the `md5` digest for *Hello orlando*.

Let's now repeat the same request:

```
curl http://localhost:8090/api/hello/orlando
Hello orlando
```

Looking again at the logs generated when Demo Service 1 sends the request to Demo Service 2:

```
2017-07-11 07:26:50 DEBUG RestTemplate:87 - Created GET request for "http://local
2017-07-11 07:26:50 DEBUG RestTemplate:779 - Setting request Accept header to [te
2017-07-11 07:26:50 DEBUG RequestAddCookies:123 - CookieSpec selected: default
2017-07-11 07:26:50 DEBUG RequestAuthCache:77 - Auth cache not set in the context
2017-07-11 07:26:50 DEBUG CachingExec:300 - Revalidating cache entry
2017-07-11 07:26:50 DEBUG PoolingHttpClientConnectionManager:255 - Connection req
2017-07-11 07:26:50 DEBUG wire:87 - http-outgoing-0 << "end of stream"
2017-07-11 07:26:50 DEBUG DefaultManagedHttpClientConnection:79 - http-outgoing-0
2017-07-11 07:26:50 DEBUG PoolingHttpClientConnectionManager:288 - Connection lea
2017-07-11 07:26:50 DEBUG MainClientExec:235 - Opening connection {}->http://loca
2017-07-11 07:26:50 DEBUG DefaultHttpClientConnectionOperator:139 - Connecting to
2017-07-11 07:26:50 DEBUG DefaultHttpClientConnectionOperator:146 - Connection es
2017-07-11 07:26:50 DEBUG MainClientExec:256 - Executing request GET /api/hello/o
2017-07-11 07:26:50 DEBUG MainClientExec:261 - Target auth state: UNCHALLENGED
2017-07-11 07:26:50 DEBUG MainClientExec:267 - Proxy auth state: UNCHALLENGED
```

```
2017-07-11 07:26:50 DEBUG headers:133 - http-outgoing-1 >> GET /api/hello/orland
2017-07-11 07:26:50 DEBUG headers:136 - http-outgoing-1 >> Accept: text/plain, a
2017-07-11 07:26:50 DEBUG headers:136 - http-outgoing-1 >> Host: localhost:8080
2017-07-11 07:26:50 DEBUG headers:136 - http-outgoing-1 >> Connection: Keep-Aliv
2017-07-11 07:26:50 DEBUG headers:136 - http-outgoing-1 >> User-Agent: Apache-Ht
2017-07-11 07:26:50 DEBUG headers:136 - http-outgoing-1 >> Accept-Encoding: gzip
2017-07-11 07:26:50 DEBUG headers:136 - http-outgoing-1 >> Via: 1.1 localhost (A
2017-07-11 07:26:50 DEBUG headers:136 - http-outgoing-1 >> If-None-Match: "023e8
2017-07-11 07:26:50 DEBUG wire:73 - http-outgoing-1 >> "GET /api/hello/orlando H
2017-07-11 07:26:50 DEBUG wire:73 - http-outgoing-1 >> "Accept: text/plain, appl
2017-07-11 07:26:50 DEBUG wire:73 - http-outgoing-1 >> "Host: localhost:8080[\r]
2017-07-11 07:26:50 DEBUG wire:73 - http-outgoing-1 >> "Connection: Keep-Alive[\
2017-07-11 07:26:50 DEBUG wire:73 - http-outgoing-1 >> "User-Agent: Apache-HttpC
2017-07-11 07:26:50 DEBUG wire:73 - http-outgoing-1 >> "Accept-Encoding: gzip,de
2017-07-11 07:26:50 DEBUG wire:73 - http-outgoing-1 >> "Via: 1.1 localhost (Apac
2017-07-11 07:26:50 DEBUG wire:73 - http-outgoing-1 >> "If-None-Match: "023e8caa
2017-07-11 07:26:50 DEBUG wire:73 - http-outgoing-1 >> "[\r][\n]"
2017-07-11 07:26:50 DEBUG wire:73 - http-outgoing-1 << "HTTP/1.1 304 [\r][\n]"
2017-07-11 07:26:50 DEBUG wire:73 - http-outgoing-1 << "X-Application-Context: a
2017-07-11 07:26:50 DEBUG wire:73 - http-outgoing-1 << "ETag: "023e8caa26fd74114
2017-07-11 07:26:50 DEBUG wire:73 - http-outgoing-1 << "Date: Tue, 11 Jul 2017 1
2017-07-11 07:26:50 DEBUG wire:73 - http-outgoing-1 << "[\r][\n]"
2017-07-11 07:26:50 DEBUG headers:122 - http-outgoing-1 << HTTP/1.1 304
2017-07-11 07:26:50 DEBUG headers:125 - http-outgoing-1 << X-Application-Context
2017-07-11 07:26:50 DEBUG headers:125 - http-outgoing-1 << ETag: "023e8caa26fd74
2017-07-11 07:26:50 DEBUG headers:125 - http-outgoing-1 << Date: Tue, 11 Jul 201
2017-07-11 07:26:50 DEBUG MainClientExec:285 - Connection can be kept alive inde
2017-07-11 07:26:50 DEBUG PoolingHttpClientConnectionManager:320 - Connection [i
2017-07-11 07:26:50 DEBUG PoolingHttpClientConnectionManager:326 - Connection re
2017-07-11 07:26:50 DEBUG RestTemplate:691 - GET request for "http://localhost:8
2017-07-11 07:26:50 DEBUG RestTemplate:102 - Reading [java.lang.String] as "text,
```

First notice the request sent from Demo Service 1 now includes the header ***If-None-Match: "023e8caa26fd7411445527af3d9aed055"***. Then look at Demo Service 2's response status, ***304 NOT MODIFIED*** with the same ***ETag*** value and no body. But the `curl` output was ***Hello orlando***, that's because it was retrieved from the cache.

And that's it for this post, thanks for reading and feedback is always appreciated. If you found this post helpful and would like to receive updates when content like this gets published, sign up to the newsletter.

6. SOURCE CODE

Accompanying source code for this blog post can be found at:

- demo-caching-resttemplate-1
- demo-caching-resttemplate-2

7. REFERENCES

- https://tools.ietf.org/html/rfc7232#section-2.3
- http://www.baeldung.com/etags-for-rest-with-spring
- http://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/filter/ShallowEtagHeaderFilter.html

ABOUT THE AUTHOR

Orlando L Otero

Orlando L Otero is a Software Engineer Consultant at FedEx Express, focusing on integration, microservices, API design and implementation and agile delivery.

300
Shares