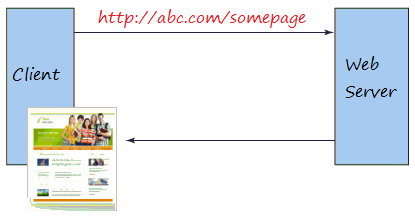
1- Web vs Web Service

Avant de répondre à la question de RESTful, je souhaite que vous reconnaissiez la différence entre le  Web et le service Web.

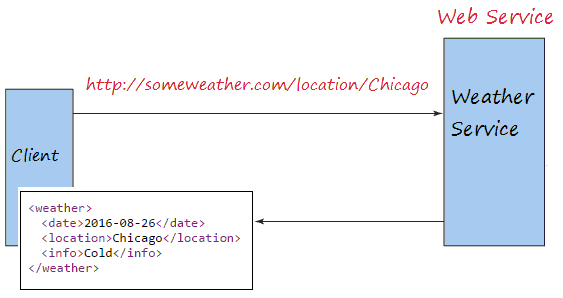
Lorsque vous saisissez une URL dans le navigateur et vous obtenez un site web. Ceci est un contenu normal que vous pouvez lire et que son contenu est destiné aux utilisateurs finaux.



Alors que le Service Web est un concept plus large que celui du web ordinaire. Il fournit l'information brute et source de confusion pour la plupart des utilisateurs donc, il est utilisée par des applications. Ces applications vont analyser des données brutes avant de les renvoyer aux utilisateurs finaux.

Par exemple, lorsque vous entrez sur un certain site Web ABC pour voir les informations météorologiques et les informations sur les titres. Le site Web affichera l'information que vous voulez.  
  
Afin d'obtenir les données météorologiques, l'application ABC doit obtenir les informations d'une certaine source. La source peut être un service Web qui fournit les données météorologiques à chaque région respectivement.  
  
De même, afin d'obtenir les données sur les titres, l'application ABC doit contacter les services de fourniture des données. Les données seront traitées avant de vous retourner un site Web complet.

Les Services Web fournissent souvent les données brutes qui sont difficiles à comprendre pour la plupart des utilisateurs finaux courants et les Web Services sont souvent retournés en format XML ou JSON.



2- Qu'est - ce que RESTful Service ?

RESTful Web Service est un Web Service qui utilise la structure REST. REST a été largement utilisé, remplaçant les Web Services  basés sur SOAP et WSDL. Les  RESTful Web Services  sont légers (lightweigh), faciles à étendre et à entretenir.

Le premier concept sur REST (Representational State Transfer) a été lancé en 2000 dans la thèse de doctorat de*Roy Thomas Fielding* (co-fondateur du protocole HTTP). Ses recherches détaillent les contraintes, les règles et la façon de fonctionner dans le système afin d'obtenir un système REST.

REST définit un ensemble de principes architecturaux par lesquels vous pouvez concevoir des services Web qui se concentrent sur les ressources d'un système, y compris la façon dont les états des ressources sont adressés et transférés via HTTP par un large éventail de clients écrits dans différentes langues. Mesuré par le nombre de services Web qui l'utilisent, REST s'est imposé au cours des dernières années seulement comme un modèle prédominant de conception de services Web. En fait, REST a eu un tel impact sur le Web qu'il a surtout déplacé la conception d'interfaces basées sur SOAP et WSDL parce que c'est un style considérablement plus simple à utiliser.

REST est un ensemble de règles qui vise à créer une application de Web Service selon les quatre règles de base ci-dessous :

1. Utiliser les méthodes HTTP explicitement.
2. Être apatride.
3. Exposer les URI de type structure de répertoires.
4. Transférer XML, JavaScript Object Notation (JSON), ou les deux.

*Dans le mot RESTful, ful est suffixe (suffix) en anglais, comme help signifie l'aide , helpful est très utile.*

3- Utiliser la méthode HTTP explicitement

REST donne une règle obligeant les programmeurs à spécifier leur but via la méthode HTTP. Les finalités comprennent normalement l'obtention, l'insertion, la mise à jour ou l'effacement des données. Dans le cas où vous souhaitez réaliser l'un des objectifs, vous devez prendre note des règles ci-dessous :

* Pour créer une ressource sur le serveur, vous devez utiliser la méthode POST.
* Pour accéder à une ressource, utilisez GET.
* Pour modifier l'état d'une ressource ou pour la mettre à jour, utilisez PUT.
* Pour annuler ou supprimer une ressource, utilisez DELETE.

Veuillez à noter que les règles ci-dessus ne sont pas obligatoires, en fait vous pouvez utiliser la méthode de GET pour demander les données, insérer, éditer ou supprimer les données sur le Serveur. Cependant, REST donne les règles mentionnées ci-dessus sur l'objectif que tout devient clair et compréhensible.  
  
L'exemple ci-dessous est la façon dont vous utilisez GET pour ajouter plus de données sur le serveur (notez que cela est contraire aux règles de REST).

Utilisez GET pour demander d'ajouter un utilisateur nommé​​​​​​​ *Robert*.

GET /adduser?name=Robert HTTP/1.1

Utilisez GET pour demander le serveur à renommer le nom des utilisateurs de Robert à Smith.

GET /updateuser?name=Robert&amp;newname=Smith HTTP/1.1

Et maintenant, selon les règles de REST tout devient plus clair.

Envoyez la commande POST HTTP pour ajouter des données :

POST /users/Robert HTTP/1.1

Host: myserver

Content-Type: application/xml

<?xml version="1.0"?>

<user>

<name>Robert</name>

</user>

Envoyez une commande GET si vous souhaitez obtenir les données sur le système.

GET /users/Robert HTTP/1.1

Host: myserver

Accept: application/xml

Envoyez une commande PUT si vous souhaitez mettre à jour les données.

PUT /users/Robert HTTP/1.1

Host: myserver

Content-Type: application/xml

<?xml version="1.0"?>

<user>

<name>Smith</name>

</user>

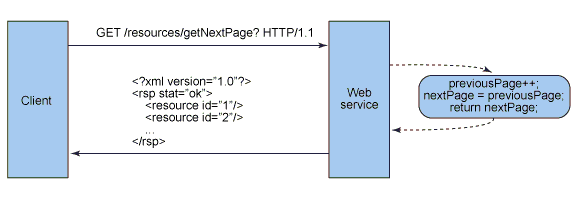
Envoyez la commande DELETE si vous voulez supprimer les données :

DELETE /users/Robert HTTP/1.1

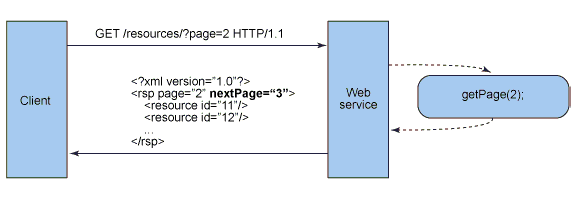
4- Être apatride (Stateless)

Une caractéristique de REST est apatride, ce qui signifie qu'elle ne stocke pas les informations du client. Par exemple, vous venez d'envoyer une demande pour voir la page2 d'un document et vous souhaitez maintenant voir la page suivante (Page 3). REST ne stockera pas l'information qui vous a servi la page 2 précédemment. Cela signifie que REST ne gère pas Session.

L'image ci-dessous illustre une application de stockage d'état qui sait quel numéro de page les utilisateurs visualisent. Et les utilisateurs ont besoin uniquement de demander la *"Next page"* pour obtenir la page désirée.



Pour les conceptions sans état, le client doit envoyer une exigence claire, y compris le numéro de page



Les composants côté serveur sans état, par contre, sont moins compliqués à concevoir, à écrire et à distribuer sur des serveurs à répartition de charge équilibrée. Un service sans état non seulement fonctionne mieux, mais il transfère la plus grande partie de la responsabilité du maintien de l'état à l'application client. Dans un service Web RESTful, le serveur est responsable de générer des réponses et de fournir une interface qui permet au client de maintenir seul l'état de l'application.

5- Exposer la structure de répertoire comme URI

REST donne une structure permettant aux utilisateurs d'accéder à leurs ressources via les URL. Les ressources ici sont toutes les choses que vous pouvez nommer (vidéo, image, bulletin météorologique, ...)

Vous devez créer les  REST services qui renvoient les ressources aux utilisateurs, respectivement.

Les adresses REST service vent être intuitives au point d'être faciles à deviner. Pensez à une URI comme une sorte d'interface d'auto-documentation qui nécessite peu ou pas d'explications ou de références pour qu'un développeur comprenne ce qu'elle indique et puisse en tirer des ressources connexes. cette fin, la structure d'une URI doit être simple, prévisible et facile à comprendre.

Voir l'URL ci-dessous, il fournit des informations météorologiques d'une région correspondant à une date donnée et il est facile à comprendre pour les utilisateurs.

http://myservice.com/weather/chicago/2016-09-27

http://myservice.com/weather/hanoi/2016-11-11

Quelques lignes directrices supplémentaires à prendre en compte lors de la réflexion sur la structure URI d'un service Web RESTful sont les suivantes :

* Masquez les extensions de fichiers de la technologie de script côté serveur (.jsp,.php,.asp), s'il y en a, pour que vous puissiez porter sur autre chose sans modifier les URIs.
* Gardez tout en minuscule.
* Remplacez les espaces par des traits d'union ou des tirets de soulignement (l'un ou l'autre).
* Évitez autant que possible les chaînes de requête.
* Au lieu d'utiliser le code 404 Not Found si la requête URI est pour un chemin partiel, fournissez toujours une page ou une ressource par défaut comme réponse.

6- Transférer XML, JSON ou tous les deux

Lorsque le Client envoie une requête au service web, elle est souvent transmise en XML ou JSON et reçoit souvent des formulaires similaires.

Parfois, le Client peut également spécifier les types de données de retour qu'il souhaite (JSON ou XML, ...), ces désignations sont appelées le type de MINE, il est envoyé dans la section HEADER de la demande.  
Voici les types communs de MINE utilisent normalement avec le REST service .

| **MIME-Type** | **Content-Type** |
| --- | --- |
| **JSON** | application/json |
| **XML** | application/xml |
| **XHTML** | application/xhtml+xml |

Par exemple, le client envoie une demande pour obtenir des informations météorologiques et demande les données de retour au format XML.

GET /weather/chicago/2016-08-27 HTTP/1.1

Host: myservice.com

Accept: application/xml;q=0.5

Et les données reçues :

<**weather**>

<**date**>2016-08-27</**date**>

<**location**>Chicago</**location**>

<**info**>Hot</**info**>"//

</**weather**>

Dans le cas où le client exige des données de retour au format JSON:

{

"date": "2016-08-27",

"location": "Chicago",

"info": "Hot"

}

Introduction

Le document est écrit sur la base de :

* Eclipse 4.6 (NEON).
* Jersey

Si vous êtes débutant avec RESTful Web Service, vous devez prendre 10 minutes pour savoir ce qui est RESTful Web service avant de commencer avec une application Java RESTful serivce.  
  
Vous pouvez voir "Qu'est-ce que le service web RESTful" ici :

* [Qu'est-ce que RESTful Web Service?](https://devstory.net/10773/qu-est-ce-que-restful-web-service)

2- Qu'est-ce que Jersey?

Jersey qui est une plate-forme Java de source ouverte vous permet de développer des applications RESTful Web service et les applications Client connexes. Jersey exécute les spécifications JSR 311.

Le Jersey fournit des bibliothèques de mettre en œuvre RESTful web service dans le conteneur de servlet (Servlet Container).  
  
A côté serveur, Jersey fournit une implémentation de servlet qui analyse des classes prédéfinies pour identifier les ressources RESTful. Dans votre fichier de configuration web.xml, enregistrez cette servlet pour votre application web.  
  
L'implémentation Jersey fournit également une bibliothèque client pour communiquer avec un service web RESTful.

Voici la structure de liaison des ressources proposées par Jersey.



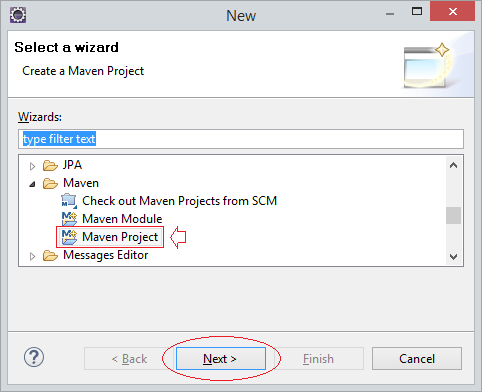
Dans le quel com.sun.jersey.spi.container.servlet.ServletContainer est un servlet fourni par JERSEY, et doit être déclaré dans *web.xml*.

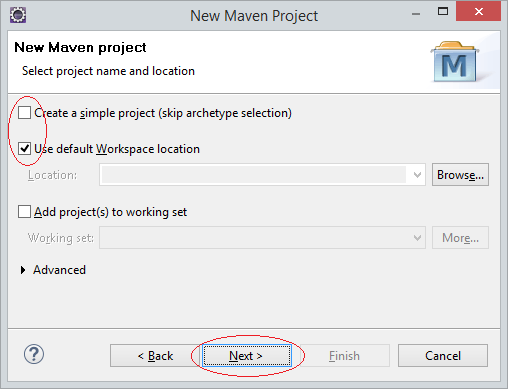


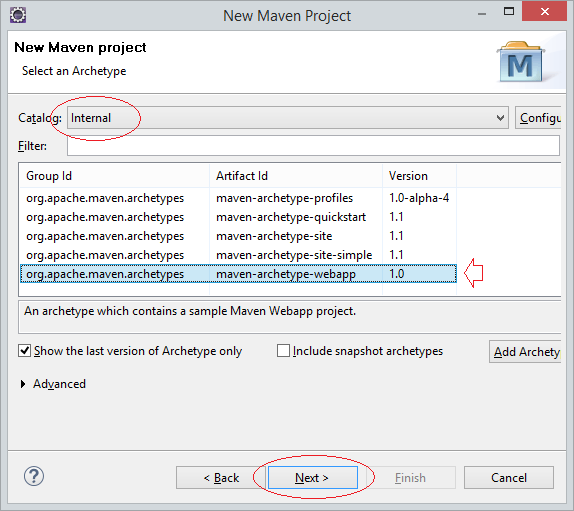
3- Créerr une application Maven

Sur Eclipse sélectionnez :

* File/New/Other..

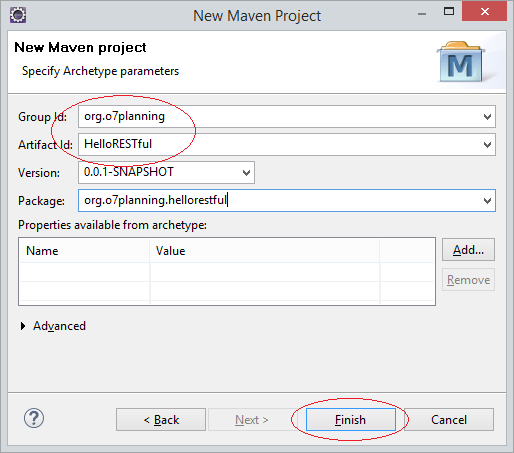


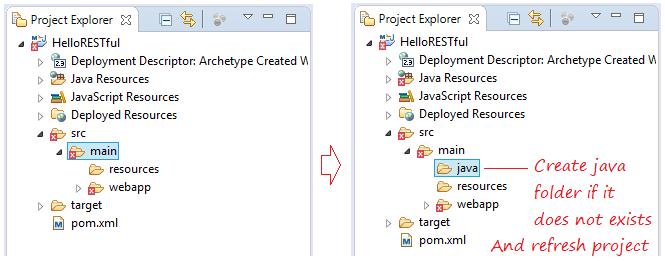




Saisissez :

* Group Id: org.o7planning
* Artifact Id: HelloRESTful





4- Déclarer maven et web.xml

Vous devez déclarer la bibliothèque JERSEY et Servlet dans le pom.xml :

pom.xml

<**project** xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/maven-v4\_0\_0.xsd">

<**modelVersion**>4.0.0</**modelVersion**>

<**groupId**>org.o7planning</**groupId**>

<**artifactId**>HelloRESTful</**artifactId**>

<**packaging**>war</**packaging**>

<**version**>0.0.1-SNAPSHOT</**version**>

<**name**>HelloRESTful Maven Webapp</**name**>

<**url**>http://maven.apache.org</**url**>

<**dependencies**>

<**dependency**>

<**groupId**>junit</**groupId**>

<**artifactId**>junit</**artifactId**>

<**version**>3.8.1</**version**>

<**scope**>test</**scope**>

</**dependency**>

<!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->

<**dependency**>

<**groupId**>javax.servlet</**groupId**>

<**artifactId**>javax.servlet-api</**artifactId**>

<**version**>3.1.0</**version**>

<**scope**>provided</**scope**>

</**dependency**>

<!-- https://mvnrepository.com/artifact/asm/asm -->

<**dependency**>

<**groupId**>asm</**groupId**>

<**artifactId**>asm</**artifactId**>

<**version**>3.3.1</**version**>

</**dependency**>

<!-- https://mvnrepository.com/artifact/com.sun.jersey/jersey-bundle -->

<**dependency**>

<**groupId**>com.sun.jersey</**groupId**>

<**artifactId**>jersey-bundle</**artifactId**>

<**version**>1.19.2</**version**>

</**dependency**>

<!-- https://mvnrepository.com/artifact/org.json/json -->

<**dependency**>

<**groupId**>org.json</**groupId**>

<**artifactId**>json</**artifactId**>

<**version**>20160810</**version**>

</**dependency**>

<!-- https://mvnrepository.com/artifact/com.sun.jersey/jersey-server -->

<**dependency**>

<**groupId**>com.sun.jersey</**groupId**>

<**artifactId**>jersey-server</**artifactId**>

<**version**>1.19.2</**version**>

</**dependency**>

<!-- https://mvnrepository.com/artifact/com.sun.jersey/jersey-core -->

<**dependency**>

<**groupId**>com.sun.jersey</**groupId**>

<**artifactId**>jersey-core</**artifactId**>

<**version**>1.19.2</**version**>

</**dependency**>

</**dependencies**>

<**build**>

<**finalName**>HelloRESTful</**finalName**>

<**plugins**>

<**plugin**>

<**groupId**>org.apache.tomcat.maven</**groupId**>

<**artifactId**>tomcat7-maven-plugin</**artifactId**>

<**version**>2.2</**version**>

<**configuration**>

<**path**>/HelloRESTful</**path**>

<**port**>8080</**port**>

</**configuration**>

</**plugin**>

</**plugins**>

</**build**>

</**project**>

Déclarer RESTful Servlet - *com.sun.jersey.spi.container.servlet.ServletContainer* - Un Servlet est fourni par Jersey REST API.  
  
Vous devez déclarer le package qui contient votre classe RESTful avec ce servlet via les paramètre *com.sun.jersey.config.property.packages*.  
  
(Notez l'utilisation du *servlet* > = 3.0)

web.xml

<?xml version="1.0" encoding="UTF-8"?>

<**web-app** xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns="http://java.sun.com/xml/ns/javaee"

xmlns:web="http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee

http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"

version="3.0">

<**display-name**>Hello RESTful Service</**display-name**>

<**servlet**>

<**servlet-name**>jerseyServlet</**servlet-name**>

<**servlet-class**>com.sun.jersey.spi.container.servlet.ServletContainer</**servlet-class**>

<**init-param**>

<**param-name**>com.sun.jersey.config.property.packages</**param-name**>

<**param-value**>org.o7planning.hellorestful</**param-value**>

</**init-param**>

<**load-on-startup**>1</**load-on-startup**>

</**servlet**>

<**servlet-mapping**>

<**servlet-name**>jerseyServlet</**servlet-name**>

<**url-pattern**>/rest/\*</**url-pattern**>

</**servlet-mapping**>

</**web-app**>

5- RESTful Service class

Dans l'étape suivante, vous devez créer votre classe de service RESTful. Par exemple, la classe WeatherRESTfulService ci-dessous est un service Web fournissant des informations météorologiques.

WeatherRESTfulService.java

**package** org.o7planning.hellorestful;

**import** java.text.DateFormat;

**import** java.text.ParseException;

**import** java.text.SimpleDateFormat;

**import** java.util.Date;

**import** java.util.Random;

**import** javax.ws.rs.GET;

**import** javax.ws.rs.Path;

**import** javax.ws.rs.PathParam;

**import** javax.ws.rs.Produces;

@Path("/weather")

**public** **class** **WeatherRESTfulService** {

**private** **static** **final** DateFormat df = **new** **SimpleDateFormat**("yyyy-MM-dd");

//

// http://localhost:8080/contextPath/rest/weather/{location}/{date}

// Example:

// http://localhost:8080/contextPath/rest/weather/chicago/2016-09-27

// http://localhost:8080/contextPath/rest/weather/hanoi/2016-09-27

//

@Path("{location}/{date}")

@GET

@Produces("application/xml")

**public** String **getWeather\_XML**(@PathParam("location") String location,//

@PathParam("date") String dateStr) {

Date date = null;

**if** (dateStr == null || dateStr.length() == 0) {

date = **new** **Date**();

} **else** {

**try** {

date = df.parse(dateStr);

} **catch** (ParseException e) {

date = **new** **Date**();

}

}

dateStr = df.format(date);

String[] weathers = **new** **String**[] { "Hot", "Rain", "Cold" };

int i = **new** **Random**().nextInt(3);

String weather = weathers[i];

**return** "<weather>"//

+ "<date>" + dateStr + "</date>"//

+ "<location>" + location + "</location>"//

+ "<info>" + weather + "</info>"//

+ "</weather>";

}

//

// http://localhost:8080/contextPath/rest/weather/{location}

// Example:

// http://localhost:8080/contextPath/rest/weather/chicago

// http://localhost:8080/contextPath/rest/weather/hanoi

//

@Path("{location}")

@GET

@Produces("application/xml")

**public** String **getWeather\_XML**(@PathParam("location") String location) {

**return** getWeather\_XML(location, null);

}

//

// http://localhost:8080/contextPath/rest/weather/{location}/{date}

// Example:

// http://localhost:8080/contextPath/rest/weather/chicago/2016-09-27

// http://localhost:8080/contextPath/rest/weather/hanoi/2016-09-27

//

@Path("{location}/{date}")

@GET

@Produces("application/json")

**public** String **getWeather\_JSON**(@PathParam("location") String location,//

@PathParam("date") String dateStr) {

Date date = null;

**if** (dateStr == null || dateStr.length() == 0) {

date = **new** **Date**();

} **else** {

**try** {

date = df.parse(dateStr);

} **catch** (ParseException e) {

date = **new** **Date**();

}

}

dateStr = df.format(date);

String[] weathers = **new** **String**[] { "Hot", "Rain", "Cold" };

int i = **new** **Random**().nextInt(3);

String weather = weathers[i];

**return** "{" //

+ "'date': '" + dateStr + "'," //

+ "'location': '" + location + "'," //

+ "'info': '" + weather + "'" //

+ "}";

}

//

// http://localhost:8080/contextPath/rest/weather/{location}

// Example:

// http://localhost:8080/contextPath/rest/weather/chicago

// http://localhost:8080/contextPath/rest/weather/hanoi

//

@Path("{location}")

@GET

@Produces("application/json")

**public** String **getWeather\_JSON**(@PathParam("location") String location) {

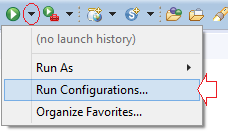
**return** getWeather\_JSON(location, null);

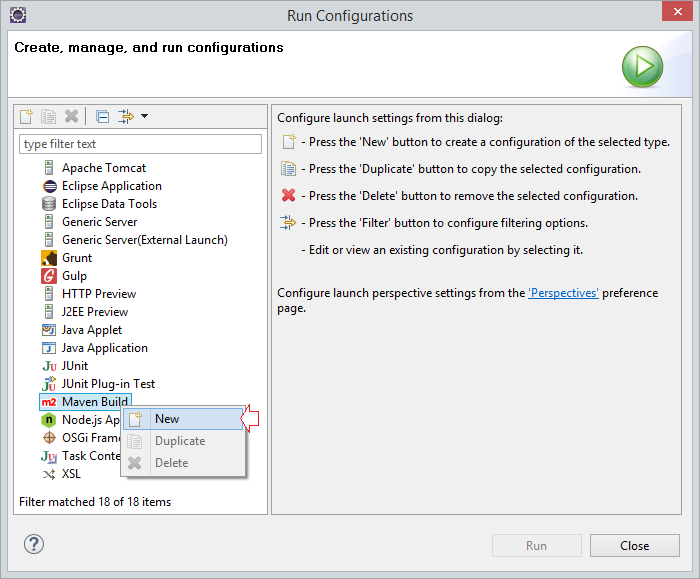
}

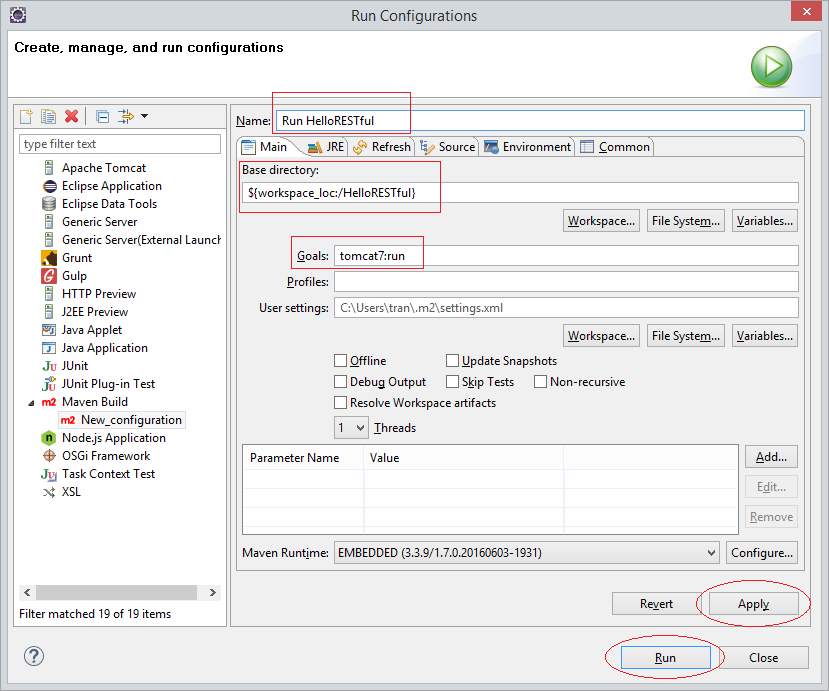
}

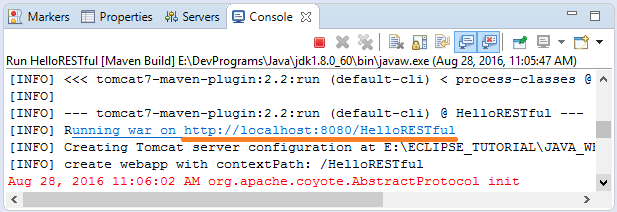
6- Exécuter l'application

Configurez pour exécuter l’application :





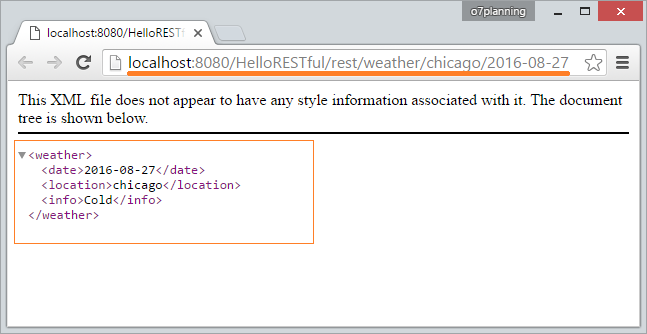




7- Explain RESTful

Copiez l'URL ci-dessous et l'exécutez sur le navigateur :

* <http://localhost:8080/HelloRESTful/rest/weather/chicago/2016-08-27>



Lorsque vous exécutez l'URL ci-dessus, c'est la même chose que lorsque vous envoyez une requête au serveur :

GET /HelloRESTful/rest/weather/chicago/2016-08-27 HTTP/1.1

Host: localhost:8080

Si la requete ne spécifie pas le type de données de retour (type MINE), le type de données par défaut sera renvoyé par le service web.

Les requêtes peuvent spécifier le type de données de retour.

GET /HelloRESTful/rest/weather/chicago/2016-08-27 HTTP/1.1

Host: localhost:8080

Accept: application/xml

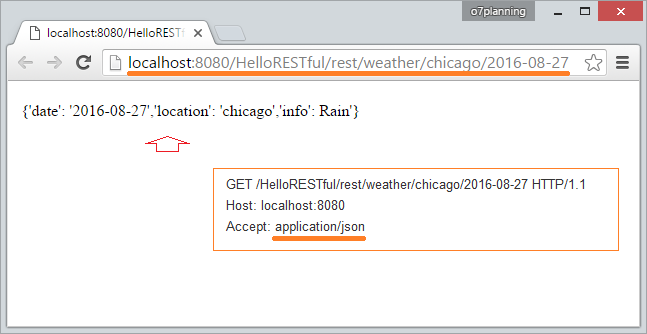
Ou :

GET /HelloRESTful/rest/weather/chicago/2016-08-27 HTTP/1.1

Host: localhost:8080

Accept: application/json

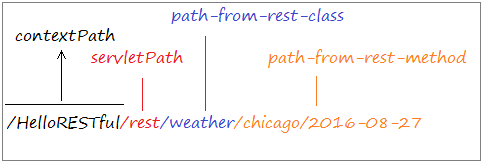
Dans l'exemple ci-dessus, si requête spécifie le type de retour comme application/json, vous obtiendrez un résultat comme ci-dessous :



*Pour spécifier le type de données renvoyées, vous devez rédiger une application Client pour former des Requêtes (Request) personnalisées et les envoyer au service web. Toutefois, vous pouvez également utiliser les AddOns du navigateur pour ajuster les requêtes avant de les envoyer au service RESTful.  
  
RESTClient AddOns est un tel plugin, vous pouvez le configurer sur Firefox ou Chrome. Il vous aide à tester les services web RESTful dans le processus de développement d'applications.*

* [*RESTClient Un débogueur pour RESTful Web Service*](https://devstory.net/11203/restclient-un-debogueur-pour-restful-web-service)

Comment fonctionne Jersey RESTful ?



contextPath:



REST Path:

