**3. Client and WebTarget**

Client and WebTarget

|-Best Practices in Building client

Till now we have created the client but we didn't call the Resource on the client to access the Resource that is running on the server side.

The Resource that is running on the server side also called as target or WebTarget which represents the URI bcz we targetting the Resource to access and the Resource will have URI hence target represents the URI of the Resource which is running on the server side.

**Client and WebTarget:**

We can access the Resource when we have client connection so once we have client obj we can call the target on that client obj that means the purpose of Client, is to create WebTarget or target instances once client have target it can call the Resource so it is the responsibility of the client to create WebTarget or target obj:

Like ClientBuilder, the Client interface implements Configurable. This allows you to change configuration and register components for the Client on the fly as our application executes. In addition to this client we have bunch of methods to create target or WebTarget.

package javax.ws.rs.client.Client;

public interface Client extends Configurable<Client> {

public void close();

public WebTarget target(String uri);

public WebTarget target(URI uri);

public WebTarget target(UriBuilder uriBuilder);

public WebTarget target(Link link);

...

}

The WebTarget interface represents a specific URI we want to invoke on. Through the Client interface, we can create a WebTarget using one of the target() methods:

package javax.ws.rs.client.Client;

public interface WebTarget extends Configurable<WebTarget> {

public URI getUri();

public UriBuilder getUriBuilder();

public WebTarget path(String path);

public WebTarget resolveTemplate(String name, Object value);

public WebTarget resolveTemplate(String name, Object value,

boolean encodeSlashInPath);

public WebTarget resolveTemplateFromEncoded(String name, Object value);

public WebTarget resolveTemplates(Map<String, Object> templateValues);

public WebTarget resolveTemplates(Map<String, Object> templateValues,

boolean encodeSlashInPath);

public WebTarget resolveTemplatesFromEncoded(Map<String,

Object> templateValues);

public WebTarget matrixParam(String name, Object... values);

public WebTarget queryParam(String name, Object... values);

...

}

WebTarget has additional methods to extend the URI we originally constructed it with. We can add path segments or query parameters by invoking path() and queryParam(). If the WebTarget represents a URI template, the resolveTemplate() methods can fill in those variables:

ClientBuilder clientBuilder = null;

Client client = null;

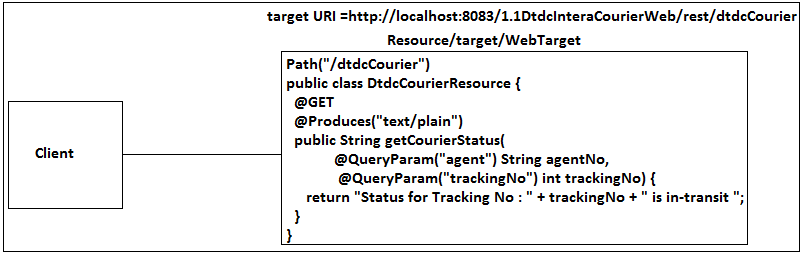
clientBuilder = ClientBuilder.newBuilder();

clientBuilder.property("connection.timeout", 1000);

client = clientBuilder.build();

// If we want we can override timeout at the client level as well

client.property("connection.timeout", 2000);



WebTarget dtdcCourierTarget =

client.target("http://localhost:8083/1.1DtdcInteraCourierWeb/rest/dtdcCourier");

// If we have PathParams, matrixParams and QueryParams add all these

// to the WebTarget that means URI related data will be added to the WebTarget.

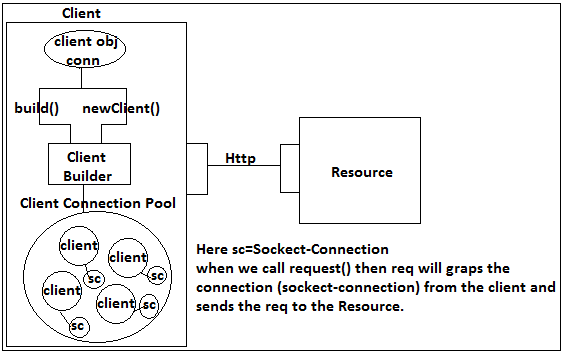
// Actually we need to get agent, trackingNo the data from the customer

// through the web Appl to give the details back to the customers

dtdcCourierTarget = dtdcCourierTarget.queryParam("agent", "ac122");

dtdcCourierTarget = dtdcCourierTarget.queryParam("trackingNo", 2928);

// Still here we have only client obj from the pool which holds target URI containing URI level data in the target but we don't have socket-connection representing the client.



Response response = dtdcCourierTarget.request().get();

/\*\*

\* When we call request() on the target then request() will grab

\* connection (network socket-connection) from the client bcz

\* socket-connections are maintained by client and client obj's will

\* maintained by the builder.

\*/

/\*\*

\* If we wanted add headers, cookies and body then we can add to the

\* target bcz target represents URI so for URI we cannot add headers,

\* cookies and body to the URI hence we need request obj for which we

\* can add headers, cookies and body so we need to create req obj If we

\* want we can we add headers, cookies, body to the request() as

\* request().headers().cookies() etc then send the Http method as get()

\* or post() etc which gives Response.

\*/

if (response.getStatus() == 200) {

String body = response.readEntity(String.class);

System.out.println(body);

}

In this example, we initialized a WebTarget with a URI template string. The resolveTemplate() method fills in the id expression, and we add another query parameter.

**Best Practices in Building cient:**

1. If we have multiple Resource methods in a Resource class then we need to create Two-WebTargets representing each Resource method URI as follows.

Webtarget searchTarget =

client.target(http://localhost:8082/JAXRSInjection/rest/bigBasket/search/{brand}

/{productName});

Webtarget searchTarget =

client.target(http://localhost:8082/JAXRSInjection/rest/bigBasket/care);

If any changes happen in the portNo or ApplContextRoot or Runtime url pattern from /rest to /resources or Resource class URI changes then we need to change in 2-WebTargets which difficult to maintain and to modify that is the reason it is not recommended to create WebTargets with hardcoded Root/Base Resource URI in 2-WebTargets rather we need to separate base URI and declare it as an constant and refer that constant in all the WebTargets so that if any changes occurs in the Root/Base Resource URI we can modify in only one so that it will reflects in all WebTargets.

2. Never create multiple client obj's rather there should be only one client obj that has to be created should be created per Resource bcz from Resource to Resource the meta-data is going to vary bcz some Resources will have Xml/JSON and some other Resources will have Xml only or some other hence it is recommended to create one client per Resource bcz client is a pool of connection.

That means for all the Sub-Resource methods that are there in a Resource can be accessed by using one-client obj similarly if we have multiple Resources containing multiple sub-Resources then we need to create one client for each Resource bcz client is client-pool which holds the network connection.

3. It is not recommended to hard code the path params, Matrix Params or Query Params must we need to resolve then dynamically.

4. For every Sub-Resource that is there in the Resource class we need to create Webtarget by writing every time including Root URI instead we can create one target which can be used for all the Resources which contains Root URI only so that in other resource methods we no need to write once again so that we proceed easily.

abstract public class Invoker {

protected Client client;

public void open() {

if (client == null) {

client = ClientBuilder.newClient();

}

}

public void close() {

if (client != null) {

client.close();

}

}

}

We made Invoker as abstract bcz we don't wanted allow to create obj.

public class BigBasketInvoker extends Invoker {

private static BigBasketInvoker instance;

private final String BIG\_BASKET\_ROOT\_URI =

"http://localhost:8083/2.1JAXRSInjection/rest/bigBasket";

private WebTarget bigBasketTarget;

private BigBasketInvoker() {

// no-op

}

public synchronized static BigBasketInvoker getInstance() {

if (instance == null) {

instance = new BigBasketInvoker();

}

return instance;

}

// We have 1st Resource search() in BigbasketResource

public String doSearch(String brand, int brandQuantity, String productName,

int productQuantity, String category) {

String searchResult = null;

WebTarget searchTarget = null;

/\*\*

\* BigBasketInvoker is child class of Invoker hence protected 'client'

\* obj will inherited bcz both are there in the same package hence we

\* can refer client obj directly

\*/

target = client.target(BIG\_BASKET\_ROOT\_URI);

// Here we are using bigBasketTarget to create searchTarget

searchTarget = bigBasketTarget.path("search").path("{brand}")

.resolveTemplate("brand", brand)

.matrixParam("quantity", brandQuantity).path("{productName}")

.resolveTemplate("productName", productName)

.matrixParam("quantity", productQuantity)

.queryParam("category", category);

System.out.println(searchTarget.getUri().toString());

Response response = searchTarget.request().get();

if (response.getStatus() == 200) {

searchResult = response.readEntity(String.class);

}

return searchResult;

}

// We have 2nd Resource getCustomerCareContact() in BigbasketResource

public String getCustomerCareContact(String city) {

WebTarget getCustomerCareContactTarget = null;

String customerCareDetailsResult = null;

// Here we are re-using bigBasketTarget to create getCustomerCareContactTarget

getCustomerCareContactTarget = bigBasketTarget.path("care").

queryParam("city",city);

Response response = getCustomerCareContactTarget.request().get();

if (response.getStatus() == 200) {

customerCareDetailsResult = response.readEntity(String.class);

}

return customerCareDetailsResult;

}

}

**Note:**

Using WebTarget we can build complete URI.

If we take a look at the UriBuilder class, we’ll see that WebTarget pretty much mirrors it. Instead of building URIs, though, WebTarget is building instances of WebTargets that we can use to invoke HTTP requests.

public interface WebTarget extends Configurable<WebTarget> {

public abstract URI getUri();

public abstract UriBuilder getUriBuilder();

// So when we call getUriBuilder() on WebTarget obj we will get UriBuilder so we can access all the methods that are there in the UriBuilder to build complete uri.

public URI getUri();

public UriBuilder getUriBuilder();

public WebTarget path(String path);

public WebTarget resolveTemplate(String name, Object value);

public WebTarget resolveTemplate(String name, Object value, boolean encodeSlashInPath);

public WebTarget resolveTemplateFromEncoded(String name, Object value);

.....

}

When we call getUriBuilder() on WebTarget obj we will get UriBuilder so we can access all the methods that are there in the UriBuilder to build complete uri.

That means we can build complete URI using UriBuilder or by using simple methods that are there in WebTarget.