Dec 01 Intro to Distributed Apps

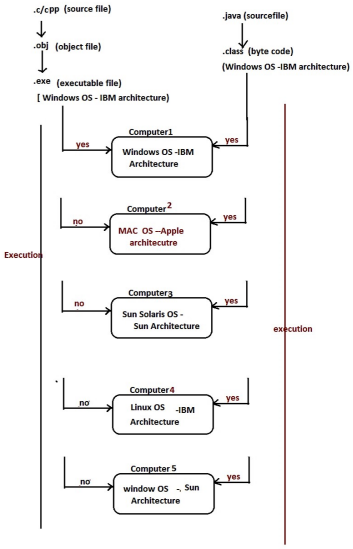
Java is platform indepenndnt and Architecture neutral

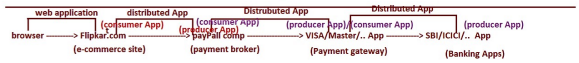
=> The plan of manifacturing vehicle is called vehicle architecture
like the four wheeler architectures are
a) Car architecture
b) Jeep architecture
c) Van architecture
d) Mini Bus Architecture
e) Mini Lorry archicture
and etc..

=> The plan /process of manifacturing computer is called computer archicture and the popular architectures are

=> IBM Architecture (all our regular laptops and desktops fall under this archicture)
=> Apple/MAC architecture
=> Sun Architecture
and etc..

=> c,c++ languages are not only Platform dependent (OS) and they are also archicture dependent where java is platform(OS) and archicture independent



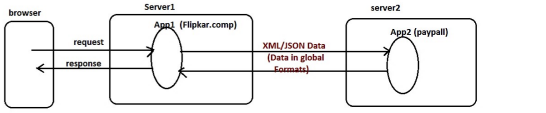


=>VISA/Master/... and etc.. Payment gateway apps providing world wide infrastructure to perform cards based (debit card/credit card) based Transactions

paypal /pay u money / rupey /razor pay and etc.. are payment broker acting as bridge b/w e-commerce Apps and payment gateways

=> The cards issued by payment gateway will come to customer through Banks by linking with bank Accounts

=>Distributed Computing / App developnent makes two Apps of two different servers belonging to same machine or different machines talking each other and also exchanging the data



=>Generally every Application we develop contains 4 layers

a) presesentation layer /UI Layer —> contains UI logics (struts ,jsf, spring mvc, servlet,jsp, angular, react js,html,js,css) (view and controller)

Model layer

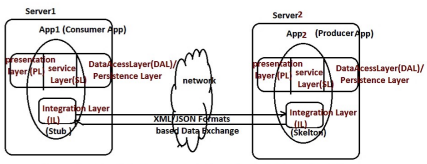
b) Serivce Layer ----------> contains b.logics (calculations, analyzations, filterings, sortings and etc..) [using java class or spring beans(spring core+spring Tx)]

c) Persinstence Layer/DataAccess Layer ------> contains persistence logic like performing CURD Operations [using jdbc, hibernate, spring jdbc, spring data jpa, spring data jdbc, spring data mongodb,spring orm and etc..]

d) Data Layer ----------> contains the data of the Application (The real persistence store) [DB s/w (SQL or No SQL) , Files (Text files, properties files, Xml files,JSON Files,...)]

if want to link one App with another App using Distributed Computing or App development env.. we need to anddtional in both cosumer and producer App that is "Integration Layer" we can that logics of this Integration layer using Distributed TEchnologies or frameworks like RMI,EJB, webservices and etc...

Consumer App —> Client App that cosumes the services of server App/Producre App
Producer App —> Server App that developes the services of server App/Producre App and keeps them ready for consumption



=> The integration layer logic of consumer App is technically called as Stub logics
=> The integration layer logic of Producer App is technically called as Skelton logics

developing
Different ways of integration logics in Java env.. /Different Distributted Technologies of Java

(a) RMI (Remote Method Invocation) | (java based)
(b) EJB (Enterprise Java Beans)
(d) CORBA (can be implemented in multiple languages)
(c) Http Invoker (from spring JEE module) | (java based)
(e) WebServices (can be implemented in multiple languages)
and etc..

other domain distributed Technologies

a) RPC (using c/c++)
b) DCOM (using micro soft technologies)
c) .Net Remoting (using .net technologies)
d) Corba (can be implemeted in muliple languages)
e) webServices (can be implemented in multiple languages)

RPC :: Remote Procedure Calls
CORBA: Common Object Request Broker Archicture
DCOM :: Distributed Computing

RMI

=>Given by Sun Ms
=> part of Jdk s/w (JSE module)
=>It is language dependent i.e both Consumer and Server App must be there in java
=> It is platform independent and Architeture neautral
=> Can not use Internet network as the communication channel b/w consumer and producer Apps\
=> Uses JRMP (Java Remote Method Procotol) as the protocol for communication
=>Outdated becoz of EJB
=> Does not give any built-in middleware services (The ready made secondary logics like security, Transaction mgmt and etc..)
=>Here data will be exchanged in binary format b/w consumer and producer

EJB

=>gives by Sun Ms
=> Enhancement of RMI
=>It is language dependent
=> It is platform and Architecture independent
=>can use both LAN (Local Arean network) and Internet(WAN) as communication channel
=> needs the heavy weight EJB Container to execute EJB Comps [EJB container is part of another heavy weight Applicaiton SErver like weblogic, glassFish and etc..)
=>Gives lots of built-in middlewares (It was popular for banking apps earlier for this reason)
=> Outdated becoz of webServices
=> It is technology of JEE module and EJB Containers of applicaiton server softwares will come as the implementations..
=>We must deploy the devleoped EJB comps in the EJB containers of Application server for execution..

note:: Though Tomcat is called as Applicaiton server from version 7 .. It is still not providing EJB container.

=> Very complex to lean and execute
=>Here data will be exchanged in binary format b/w consumer and producer

CORBA (Common Object Reqquest Broker Archicteture)

=> Platform , Architecture and language indepndent
=> we can develop/produce and consume services in multiple languages like java , c++ , c and etc..
=> CORBA is specification and it will be implemented as IDL (Interface definitation language)
=> CORBA is complex learn and apply
=> CORBA is heavy weight to implement
=> CORBA looks great conceptwise .. But gives problems towards the implementations.

Dec 01.1 Intro to Distributed Apps

WebServices

=>WebSErvices is a mechanism of linking two apps as consumer and producer apps using the protocol http .

=> WebSerevices in platform independent and archicture independent and language independent..

A producer developed in "Java" can be used/consumed in .net , java, php, phyton , Java script and etc.. (Even reverse possible)

=> WebSErvices says Develop/produce any where and consume any where.

=> Webservices makes the consumer and producer Apps exchanging data either in XML or in JSON (Global formats)

=> Java and languages have provided multiple apis/Technologies and frameworks to implement web service logics as skelton logics (Integration Logics Producer) and as stub logics (Integration logics of consumer)

=>WebServices support http request and http response based method invocation i.e as http request the cosumer App invokes method of producer App and the results method execution goes back to consumer App as http response.

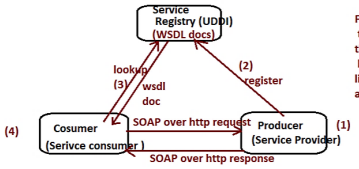
Two ways of implementing web services
- a) SOAP WebServices
- b) RESTfull WebServices

SOAP :: Simple object Access protocol (protocol)
REST :: REpresentational STatefull
web services (It is not a protocol it is kind of architecture/ mechanism)

a) SOAP WebServices

=> This runs based 3 component pricipile a) SErivce Provider(producer /Server)
b) Sevice Consumer (Consumer/Client)
c) Service Registry (where services will be registered to expose)

The Registries in SOAP based webServices env.. is UDDI (Universal Description Discovery and Integration)



For every SOAP web service that is registered in service Registry there will be one WSDL doc (xml document) having multiple details about web service like how to consume, the services names and etc (endpoint details)

WSDL :: WebService Description Language

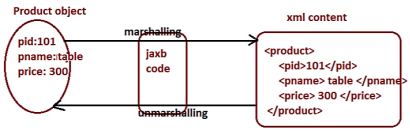
=> The http request carries xml baased SOAP message as request body nothing but inputs

=> The http response carries xml baased SOAP message as response body nothing but outputs

SOAP message strictly typed or complex XML messages which will be conveted java obejcts and reverse using Jaxb api

jaxb :: Java archicture for xml binding (nothing but converting java object data to xml content and vice-versa)

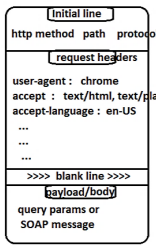
java object data to xml conversion is called marshalling and reverse is called unmarshalling



=>The consumer sends inputs to producer as SOAP message(xml) in http request
=>The Poducer sends outputs to cosumer as SOAP message(xml) in http response

=>The http request and http response contains two parts
- a) HEAD part
  - |---> contains two sections
    - a) Initial Line
    - b) headers
- b) Body part

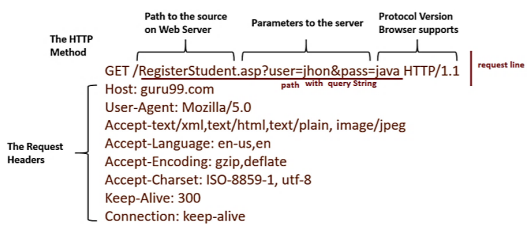
Structure of http request
=========



http methods or modes are (9)
- GET
- POST
- HEAD
- TRACE
- PUT
- DELETE
- OPTIONS
- CONNECT (reserved for future)
- PATCH (for partial Updates)

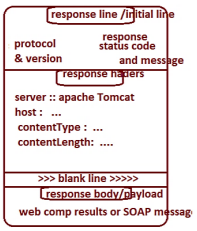
In Normal http requeust , the request body will be query String having request param names and values like sno=101&sname=raja&sadd=hyd
In soap over http request , the request body will be xml based SOAP message (xml tags)

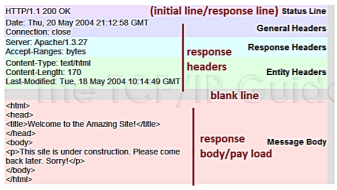
=>POST mode reuqest contain request body representing query String where as GET Mode request does not request body becoz its carries inputs as query String appended to the request url.



Http response status codes
==========
- 1xx (100-199) :: Informational
- 2xx (200-299) :: Success
- 3xx (300-399) :: Redirection
- 4xx (400-499) :: Incomplete (Client side errors)
- 5xx (500-599) :: Server side errors

Http response structure
==========

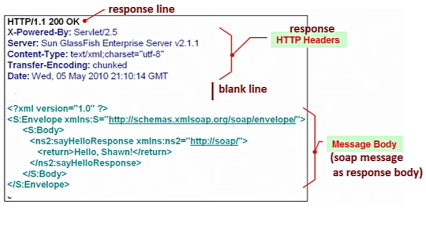




Normal http response contains the web comp genertad ouput (generally html code or plain text) as response body where as soap over http response contains the soap message(xml) as the response body

soap over http request
==========

SOAP over http response



Dec 02 Intro to Distributed Apps

WebServices
==========

=>WebSErvices is a mechanism of linking two apps as consumer and producer apps using the protocol http .

=> WebSerevices in platform independent and archicture independent and language independent..

A producer developed in "Java" can be used/consumed in .net , java, php, phyton , Java script and etc.. (Even reverse possible)

=> WebSErvices says Develop/produce any where and consume any where.

=> Webservices makes the consumer and producer Apps exchanging data either in XML or in JSON (Global formats)

=> Java and languages have provided multiple apis/Technologies and frameworks to implement web service logics as skelton logics (Integration Logics Producer) and as stub logics (Integration logics of consumer)

=>WebServices support http request and http response based method invocation i.e as http request the cosumer App invokes method of producer App and the results method execution goes back to consumer App as http response.

Two ways of implementing web services

a) SOAP WebServices — SOAP :: Simple object Access protocol (protocol)
b) RESTfull WebServices — REST :: REpresentational STatefull

web services (It is not a protocol it is kind of architecture/ mechanism)

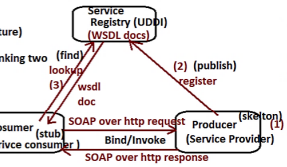
a) SOAP WebServices

=> This runs based 3 component priciple a) SErivce Provider(producer /Server)
b) Sevice Consumer (Consumer/Client)
c) Service Registry (where services will be registered to expose)

The Registries in SOAP based webServices env.. is UDDI (Universal Description Discovery and Integration)

SOAP based web services fall under SOA Archicteture (Service Orienteted Architecture)

=>SOA is the mechanism of linking two Apps contains
=> SOA service producer, Service consumer and Service registry as shown in the diagram



For every SOAP web service that is registered in service Registry there will be one WSDL doc (xml document) having multiple details about web service like how to consume, the services names and etc (endpoint details)

WSDL :: WebService Description Language

=> The http request carries xml baased SOAP message as request body nothing but inputs

=> The http response carries xml baased SOAP message as response body nothing but outputs

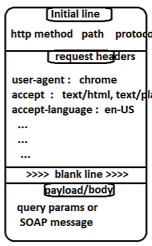
SOAP message strictly typed or complex XML messages which will be conveted java obejcts and reverse using Jaxb api

jaxb :: Java archicture for xml binding (nothing but converting java object data to xml content and vice-versa)

=>The consumer sends inputs to producer as SOAP message(xml) in http request
=>The Poducer sends outputs to cosumer as SOAP message(xml) in http response
=>The http request and http response contains two parts

a) HEAD part
|—> contains two sections
a) Initial Line
b) headers
b) Body part

Structure of http request
==========

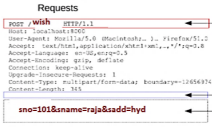


http methods or modes are (9)

GET
POST
HEAD
TRACE
PUT
DELETE
OPTIONS
CONNECT (reserved for future)
PATCH (for partial Updates)

In Normal http requeust , the request body will be query String having request param names and values like sno=101&sname=raja&sadd=hyd
In soap over http request , the request body will be xml based SOAP message (xml tags)

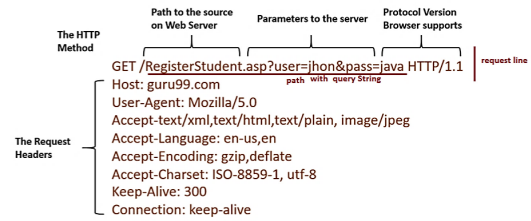
Requests



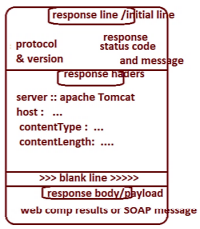
Http response status codes
==========

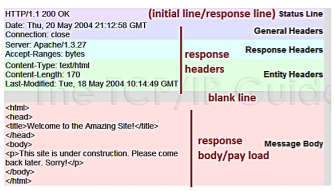
1xx (100-199) :: Informational
2xx (200-299) :: Success
3xx (300-399) :: Redirection
4xx (400-499) :: Incomplete (Client side errors)
5xx (500-599) :: Server side errors

=>POST mode reuqest contain request body representing query String where as GET Mode request does not request body becoz its carries inputs as query String appended to the request url.



Http response structure
==========

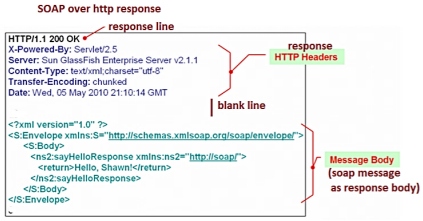




Normal http response contains the web comp genertad ouput (generally html code or plain text) as response body where as soap over http response contains the soap message(xml) as the response body

soap over http request
==========

SOAP over http response



In Java env... Jax-ws is api or technology (part of JSE module) that is given to developing SOAP based web services and the frameworks implementing SOAP based web services are a) apache axis b) apache cxf and etc..

Limitations of SOAP based web services
==========

a) These are based on SOA where seperate registry is required to publish/find web services ..
b) The service registries in SOA are very heavy maintain
c) The SOAP messages in http request and http response are complex to build and heavy to send over the network
d) Understanding about web service by reading WSDL document is not easy process.
e) SOAP message use complex Schema .. So building SOAP message is costly
f) Does not flexiblity to send data in different global formats .. should always use the heavy SOAP message as xml messages. and etc..

To overcome these problems use RestFull webServices.

Dec 02.2 Intro to Distributed Apps

b) RESTfull WebServices

=>REST is not a protocol .. REST is the mechasim or archicteture REpresenting the STate of web comp in http request and in http response So it is called RESTfull web services.

=> Allows to send data in both global formats XML , JSON

XML :: eXtensible Markup language
To convert Xml tags data to java object (unmarshalling) and reverse(marshalling) we use JaxB API (Java architecture for Xml binding)

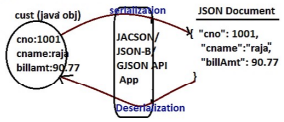
JSON :: Java Scripot Object Notation (nothing Object in Java script)

The process of converting Java object to JSON content (key: value pairs) is called Serialization and rerverse called DeSerialization

java data representation	java script data representation

String name="raja"
int age=30;
 	
Customer cust=new Customer();
cust.setCno(1001);
cust.setCname("raja");
cust.setBillAmt(90.77);
	var/let name="raja";
	var/let age=30;
	
	var/let cust= { "cno": 1001,
	"cname":"raja",
	"billAmt": 90.77
	}
	JSON doc means obj in Java script

To convert JSON content into Java Object and for reverse operation we use JSON APIs
eg: JACSON, JSON-B , GJSON and etc..



=> JSON is ligght weight compare to XML to represent the data ..

xml doc
```
<customers>
  <customer>
    <cno>101</cno>
    <cname>raja </cname>
    <billAmt>90.77 </billAmt>
  </customer>
</customers>
```

=>here compare to data,.. the structure is more like closing tag for every opening tag is bit extra

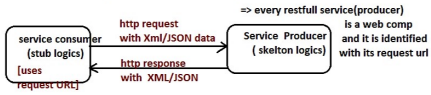
=>Gives structured data having relationship among the data memebers in bit heavy env..

JSON Doc
```
{ "cno": 1001,
  "cname":"raja",
  "billAmt": 90.77
}
```

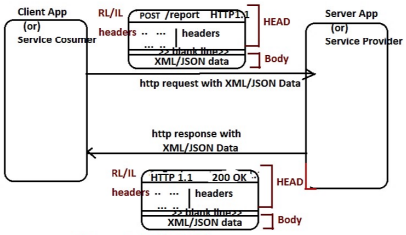
=>Here data and structure both are in good combination

=>Gives structured data having relationship among the data memebers in light env..

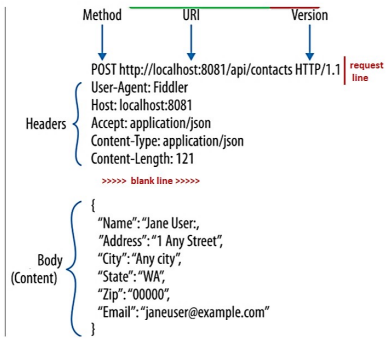
=> RESTfull is not based on SOA (SErvice oriented archicture) .. It is based on Consume- Producure Archicteture with FrontController support i.e no service registry is required here..



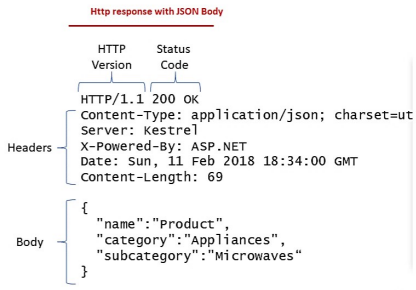
=> In RESTfull web services we prefer JSON data alot compare to xml data..
=> The service Producer (Restfull comp) can be developed in any language like java ,.net, php, phyton and etc..
=> The service consumer (Client) can be developed in any language like java ,.net, php, phyton and etc.. also can be as any app like Java app, andriod app, destktop app, php app, angular App, React App, IOS App and etc..



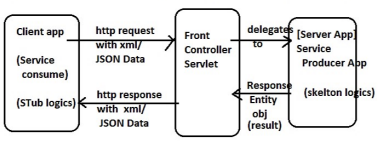
Http request with JSON BODY



Http response with JSON Body



=>The service provider in Restfull App is java class and it can not take http requests directly.. So we put one FrontController servlet like DispatcherServlet (spring) , ActionServlet(struts) and etc..
=> The special servlet that acts as entry and exit point for all requests and reponses is called Frontroller Servlet



=>In Java , Jax-RS is technology /api to develop Restfull web services where as Spring Rest , Rest easy , Restlet,Jersy and etc.. are the f/w to support Restfull web services implementation.

=>Spring Rest is given on top of spring MVC

Spring Rest =spring MVC ++

Summary on webservices

web services

SOAP Based web serivces (SOA) (Old/Legacy) ← Desiging Models → RestFull webSErvices (Consumer-Producer with FrontController) (Modren /Relatively New)

Jax-WS ← Programming APIs → Jax-RS

Apache Axis, Apache CXF ← Proramming Fraeworks → Spring Rest, Rest easy, Restlet, Jersy

SOAP over Http ← protocol → Http with JSON/Xml data

note:: Developing service providers in web services is also called as devleoping APIs.

note:: gathering and providing info that are required to develop service cosumer for invoking the services of spefic service provider is nothing but providing endpoints.

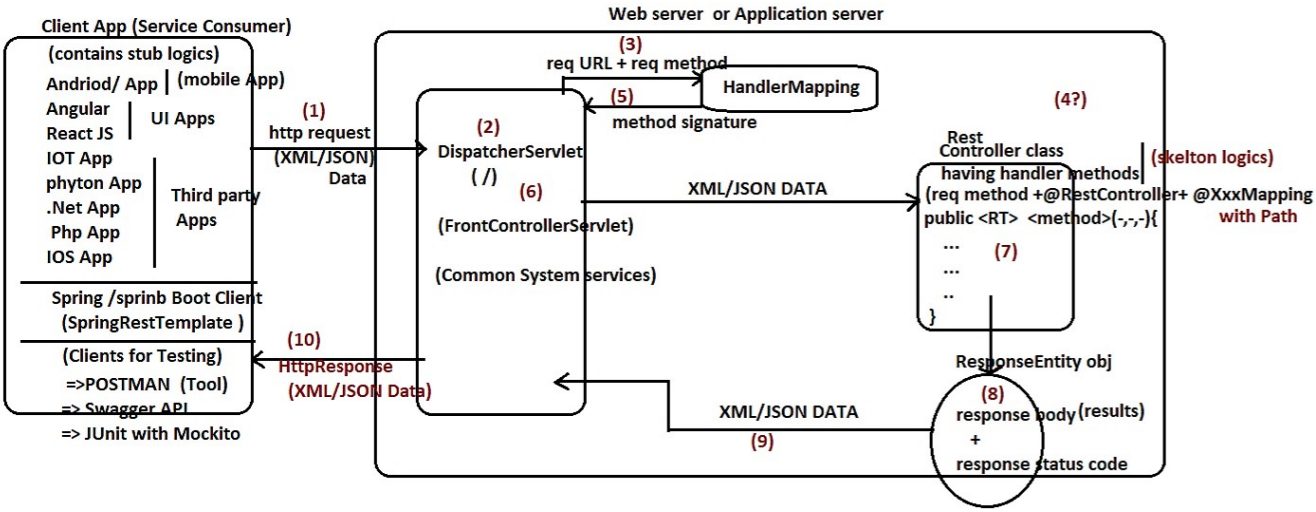
Dec 03 Spring Rest Apps

=>Spring Rest = spring mvc ++
That means spring mvc can be used to develop web application and also
RESTfull web service comps

=> Spring Rest is given based Consumer - Producer + FrontController Pattern
=> RESTfull webservices does not fall under SOA (Service Oriented Architecture) becoz there is not service registry in RESTFull Webservices .. More importantly the service provider comp of Restfull webSErvice is identified with it request url or path

=>FrontController is special web comp of web application that traps and takes either all or multiple http requests applies common system services and delegates requests to appropriate comps(controller classes) and gathers the results to send to clients.. Infact it act as entry and exit point for all the requests and responses of the web application.

Spring MVC Flow to towards developing RESTFul webservices (Spring Rest Flow)
==========



=> In RestFull web applications , there will not be any kind of view comps .. the methods of controller or RestController class directly send response to Client App through Dispatcher Servlet having XML/JSON Data which intially gernated as ResponseEntity object and later converted to XML/JSON Data

note:: In web application the client is browser window (Not programmable) giving request and getting response
note :: In Distributed Apps the Client/serivice cosumer is programmable App having logics to invoke methods of service producer(webSErvice comps) (Here browser can not be taken as client)

With respect diagram
==========

(1) The service cosumer App write stub logic by invoking service producer methods .. In this process Http request will be generated having XML/JSON Data

(2) The FrontController , DispatcherServlet (DS) traps and takes requests to apply common system services like logging, auditing and etc..

(3) The DS handovers the request to handler mapping comp giving request uri/path and request mode

(4?) The handler mapping comp searches for @Controller or @RestController classes for matching handler method and gets its signature using Reflection API

(5) HandlerMapping comp gives method signature to DS

(6) DS prepares necessary objects as required for the signature , gets controller or RestController obj and calls method

(7) Handler method executes either to process request directly or by taking the support of Service,DAO classes

(8) Handler method prepares HttpResponseEntity<T>object having response body(results) and http status code

(9) Handler method returns HttpResponseEntity<T> object to DS having XML/JSON data

(10) DS sends http resonse to Client App having XML/JSON data..

The methods of @RestController class does not need ViewResolver and View comps becoz they have natural behaviour of sending response to Client App directly through DipsatcherServlet.

Spring Rest = spring mvc ++ **Dec 06 Spring Rest 1st Apps**

=>Developing spring mvc handler method of controller class with out returning logical view name and making it returning ResponseEntity<T> object makes the handler method ready for Spring Rest env.. (Infact it becomes indirectly b.method of Restfull webservices)

=>To send response to browser /Client with out involving View .. we need to place @ResponseBody on the top of handler method in @Controller class.. Instread of writing two annotations we can just use @RestController.

@RestController = @Controller +@ResponseBody

eg1:

```
@Controller
@RequestMapping("/customer")
public class CustomerController{

 @ResponseBody
 @GetMaping("/display")
 public ResponseEntity<String> dispplayMessage(){
    ....
    ....
     ...
  return obj of ResponseEntity;
 }
}
```

(or)

eg2:

```
@RestController
@RequestMapping("/customer")
public class CustomerController{

 @GetMaping("/display")
 public ResponseEntity<String> dispplayMessage(){
    ....
    ....
     ...
  return obj of ResponseEntity;
 }
}
```

In spring MVC /Spring Rest the following cfgs takes place automatically

=> DispacherServlet cfg as FrontController with "/" url pattern
=> HandlerMapping (Default is RequestMappingHadlerMapping)
=> ErrorFilters dispplaying white label error pages
=> ViewResolver (default InternalResourceViewResolver)
and etc...

(we do not them mostly in Spring Rest programming becoz it does not use browser as client)

ResponseEntity<T> object contains two parts

a) Response body (String /object/collection)
b) Response Status code (we use ResponseStatus enum constants for like ResponseStatus.OK and etc.)

=>RepsonseStatus/Http Status indicates wheather the generated response/output/result should be given to Client as success output(200-299) or as Client Side error(400-499) or as server side error (500-599) and etc..

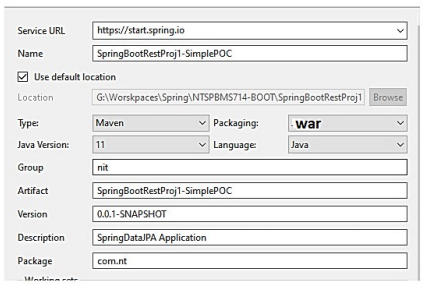
where as,

ResponseEntity<T> indicates (as the return type of hanlder method) that the generated ouput/result should go to client/browser directlty through DispatcherServlet with out involving any ViewResolvers and View Comps.

=>if ResponseEntity object is having other than <String> type as genertic type(object/collection) , then intenrally converts and holds output as JSON data (key-value) pairs.. Where as <String> type contains normal text content

Procedure to develope and Test First SpringRest App

step1) create spring boot starter project
adding spring web stater and taking packing as war file



-->next ---> select spring web starter ----> finish.

step2) Develop the @RestController class as shown below.

MessageRenderController *.java*

```
package com.nt.controller;

import java.time.LocalDateTime;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController  // (@Controller + ( @ResponseBody  on all the handler methods))
@RequestMapping("/message")  //Global path (purely optional)
public class MessageRenderController {

    @GetMapping("/generate")
    public  ResponseEntity<String>  generateMessage(){
        //get Sys date and time
        LocalDateTime ldt= LocalDateTime.now();
         //get current hour of the day  (response body)
        int hour=ldt.getHour();
        String body=null;
        if(hour<12)
             body="Good Morning";
        else if(hour<16)
            body="Good AfterNoon";
        else if(hour<20)
            body="Good Evening";
        else
            body="Good Night";
        // response status
        HttpStatus  status=HttpStatus.OK;
        //create ResponseEntity object
        ResponseEntity<String> entity=new ResponseEntity<String>(body, status);
        return entity;
    }//method
}//class
```

Dec 06.1 Spring Rest 1st Apps

step3) Test the app using browser or POSTMAN tool

note:: since ResponseEntity<String> is taken and handler method type is "GET" (in this controller)
we can send that request even from browser (otherwise not possible from browser)

Using browser (Not recomanded to use becoz it can send only GET or POST mode requests
========== where as @RestController can have GET ,POST, DELETE,PUT,PATCH and etc.. modes handler methods)

a) Run the Application using Run as server option
b) use the following from the browser

http://localhost:3030/SpringBootRestProj01-SimplePOC/message/generate

Using POSTMAN Tool (Recomanded to use) (A good tool to Test the Resftull web services /service providers developed in any language/technology /framework having capability generate different modes of request and ability recieve text/JSON/XML content based response)

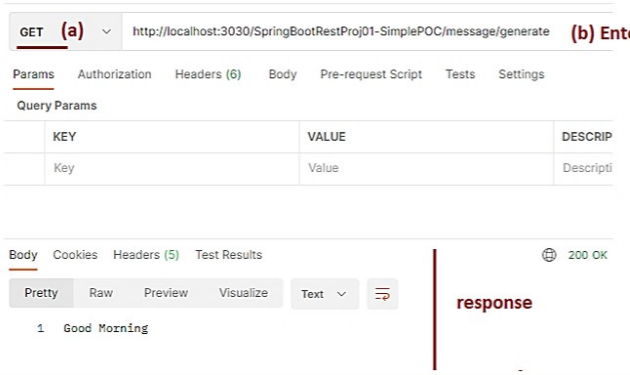
a) download and install postman tool
(by skipping the registration)

https://www.postman.com/downloads/ ----> use windows 32/64 bit for download

b) Create a new Collection
=> Postman home page ===> skip registration and go to the app (look at botton the page)===>
use (+) symbol to create the collection.

c) Send request by using the request url

Right click on the above created Collection --> Add Request --->



What is the difference between @Controller and @ResstController

@Controller	@RestController
a) Given in the spring 2.5 version (bit old annotation)	a) Given in the version spring 4.0 (relatively new annotation)
b) Specialization @Component	b) Specialization of @Controller
c) makes the java class webController class having capability to handle http requests by taking them through DispatcherServlet	c) makes the java class Rest Controller or Restfull service provider class having capability to handle http requests by taking them through DispatcherServlet
d) Can be used in both spring MVC and spring Rest Apps	d) Recomanded to use only in spring Rest Apps (Restful Apps)
e) Every Handler method does not get @ResponseBody automtically, So we need to add it expicitly	e) Every handle method automatically gets @ReponseBody So there is no need adding it seperately
of the f) Based on the return type handler method it decides wheather view comp should be invovled or not (if the return type is otherthan ResponseEntity<T> then it involves ViewResolver and view comp)	f) Makes the handler method sending its output/results as response to client direcly through DispagcherServlet out with involving ViewResolver and View comp

note:: Instead of comparing @RestController with @Controller .. plese use it as convient annotation given over @Controller providing easyness to developer Restfull Service providers (@Contrller + @ResponseBody)

Dec 07 Working with JSON Data

Http request methods/modes
==========
GET
POST
PUT
DELETE
OPTIONS
TRACE
PATCH
HEAD
CONNECT (Resrved for Future)

Generally we use the following Http request methods/modes in Restful applications while performing CURD Operations

GET —> for Read Operations (R) ->selecting records
POST —> for Create opertions (C) -> inserting records
DELETE —> for DELETE opertions (D) -> deleting records
PUT —> for Update opertions (U) -> modifying records
PATCH —> for Update opertions (U) -> modifying records

note: PUT for complete modification of the record.
PATCH for partial modification of the record.

```
public <RT> updateEmployee(Employee emp){
   ....                                          use  | PUT
   ...                                               | mode request
}
```

```
public <RT> updateEmployee Email(String newEmail){
   ....                                          use  | PATCH
   ...                                               | mode request
}
```

=>Since HEAD request mode HttpResponse does not contain body /output/results , So it can not be used for Read /Select Opertions
=> Since TRACE is given to trace/debug componets involved for the SUCCESS or FAILURE of request processing it can not be used for CURD Operations
=>Since OPTIONS mode request gives the possible Http request methods/modes that can be used to generate request to web comp .. its HttpResponse contains purely list Modes/methods that are possible to give request to web comp..

```
@WebServlet("/testurl")
public class TestServlet{

  public void doGet(req,res){
     ...
     ...
  }
}
```

if we give OPTIONS mode request to this web comps we get response as
Allow : GET,HEAD,OPTIONS

note : when we give OPTIONS mode in postman you can type urlmapping of GetMapping , PostMapping, DeleteMapping, PutMapping, PatchMapping then you don't get error and you'll get the success response of 2XX status. but you dont get the response body/result.

```
@WebServlet("/testurl")
public class TestServlet{

  public void doPost(req,res){
     ...
     ...
  }
}
```

if we give OPTIONS mode request to this web comps we get response as
Allow : POST,OPTIONS,

=>if the request url is not valid then we get 404 error (requested resource not found)
=> if the request url based web comp is not ready
to process given mode request then we get 405 error (method not allowed)
=> if the request fails in Authentication then we get 401 error
=> if the request fails in Authorization then we get 403 error (resource is forbidden)
=> if the web comp(servlet/jsp/producer) fails to instantiate for the given request then we get 500 error

=>if the return type of producure method is other than <String> generic in ResponseEntity object then the producer methods send JSON data along with the HttpResponse body

=>if the return type of producer method is <String> generic in ResponseEntity object then the producer methods sends text data along with the HttpResponse body

Every Restfull application /Project contains
a) server App /prodcuder App/ Service provider App (spring MVC App with @RestController with methods) | (also called Rest API) [The request url of @RestController and other required infomation for sending request are called End points]

b) Client App /Service Cosumer/ Consumer App

Angular
ReactJS
PHP
IOS
IOT Devices
.net
python
andriod
| (Programable client Apps)

POSTMAN
Swagger
| Tools for Testing

spring RestTemplate (spring basedClient) (Programmable Client Apps)

request headers vs req pameters

=>req headers are Client generated
inputs that go along with request
automatically having fixed names(header names)
eg:: accept , accept-language, user-agent,referer ,contentType and etc.

=>req params are enduser supplied values
as query String /form data ..req param names not fixed ..
and they are user-defined
sno=101&sname=raja&sadd=hyd
req params /query params

==> API development means Developing Spring Rest Server App/Service provider App

=> Giving API End points means providing request url and other related information to developers for developing client Apps/service consumer App for cosuming the services offered by Service provider

note:: The API/ services developed in SOAP based webservices can not consumed using Rest Client and vice-versa.

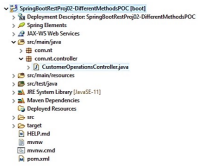
note: The Restful webSErvices developed using one kind of Rest API/framework (like jesry /spring rest, ja x-RS,Restlet and etc..) can be consumed using same Rest API or different Rest APIS (becoz both are in Restfull webService env..)

(public apis)
eg:: There are multiple open /free apis /service providers devleoped in different technologes/frameworks of Restfull webservices and they consumed in our Apps using our choice rest apis..

eg:: weather report api
Google Maps api
Gpay APIs
ICC API
Covid APIs
pay APIS

RestController class
==========

```
package com.nt.controller;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PatchMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/customer")
public class CustomerOperationsController {

    @GetMapping("/report")
    public ResponseEntity<String> showCustomersReport(){
        return new ResponseEntity<String>("From GET-ShowReport Method" , HttpStatus.OK);

    }

    @PostMapping("/register")
    public ResponseEntity<String> registerCustomer(){
        return new ResponseEntity<String>("From POST-RegisterCustomer Method" , HttpStatus.OK);

    }

    @PutMapping("/modify")
    public ResponseEntity<String> updateCustomer(){
        return new ResponseEntity<String>("From PUT-UpdateCustomer() Method" , HttpStatus.OK);

    }

    @PatchMapping("/pmodify")
    public ResponseEntity<String> updateCustomerByNo(){
        return new ResponseEntity<String>("From PATH-UpdateCustomerByNo() Method" , HttpStatus.OK);

    }

    @DeleteMapping("/delete")
    public ResponseEntity<String> deleteCustomer(){
        return new ResponseEntity<String>("From DELETE-deleteCustomer Method" , HttpStatus.OK);

    }
}
```



GET http://localhost:3030/SpringBootRestProj02-DifferentMethodsPOC/customer/report Send

POST http://localhost:3030/SpringBootRestProj02-DifferentMethodsPOC/customer/register Send

DELETE http://localhost:3030/SpringBootRestProj02-DifferentMethodsPOC/customer/delete Send

PUT http://localhost:3030/SpringBootRestProj02-DifferentMethodsPOC/customer/modify Send

PATCH http://localhost:3030/SpringBootRestProj02-DifferentMethodsPOC/customer/pmodify Send

Dec 08 Working with JSON Data

Sending JSON Data as Http Response body to Client from service Provider App

=> if the method of Service Provider is returns ResponseEntity object having other than <String> generic (can be any other object) then given object will be converted to JSON data and will be placed in HttpResonse as body automatically..

JSON :: Java Script Object Notation

=>It is a way of representing Object data using key: value format

=> Key must be in "" and value can be any thing.. if the value is "string" content then it also should be in ""

=> one {} (flower bracket) represents one object or sub object

=> "[]" represents array/list/set element values

=> In JSON array/list/set collection will be treated as array only ,so their elements will be represented using [-,-,-]

=> In JSON array/list/set collection are called 1D arrays

=> In JSON map collection is called 2D array

=> In JSON Map collection elements and HAS-A property elements will be represented using sub object/node { "key":value , "key":value,....}

```
Customer cust=new Customer(1001,"raja","hyd",67877);

In Json::
    {
      "cno": 1001,
      "cname":"raja",
       "cadd" : "hyd",
       "billAmt": 67877
      }
```

cust (Customer obj)
cno:1001
cname:raja
cadd:hyd
billAmt:67877

=>HTML (Hyper Text Markup Language) is given to display data on browser by applying styles

=> JSON/XML are given to describe data (To construct data having structure)

What is the difference b/w JSON and XML?

YML , MongoDB docs are inspired from JSON

```
Student st=new Student(101,"jani",67.88);
```

st(Student obj)
sno:101
sname: jani
avg: 67.88

JSON way of representing data

```
{
 "sno":101,
 "sname":"jani",
"avg":67.88
}
```

(Both are Global format for representing Data)

Xml way of representing Data

```
..... //schema or DTD import (optional)

<student>
   <sno>101</sno>
   <sname>jani</sname>
    <avg> 67.88 </avg>
  </student>
```

JSON	XML
(a) It is Java Script Object Notation	(a) It is eXtensible Markup language
(b) It is the way of represnting object data	(b) it is way constructing /descring data using tags
(c) maintains th data as "key": "value" pairs { "key":"value", "key" : "vlaue" }	(c) maintains the data using tags as the tags body and attributes [<> </>]
(d) Does not Provides namespaces to validate the data	(d) provides namespaces to validate data (namespace is a library that contains set tags , attributes and rules to construct xml data)
(e) This format is given by Java Script laguage (Sun Ms + Netscape)	(e) This is given according to w3c recomadations and inspired from SGML (Standard Generalized Markup Language)
(f) JSON files are easy to read and process	(f) Xml files are bit complex to read and process.
(g) To handle JSON Data we use the support of JACSON api	(g) To handler Xml data we take support of xml apis like Jaxp (java api for xml processing) Jaxp deals with SAX api, DOM api, JDOM api and etc..
(h) To covert JSON data to Object and vice-versa we can use GSON , JSON-B and etc.. api	(i) To convert Xml doc to objects and vice-versa we use JaxB api (Java api for Xml Binding)
(i) The Process converting object to JSON Data is called serialization and reverse is called Deserilziation	(i) The Process of converting object to xml data is called marshalling and reverse is called unMarshalling
(j) JSON fomat/ files are ligh weight compare to XML format/ files	j) XML fomat/ files are bit heavy weight compare to JSON format/ files
(k) Very much used is Restfull web serices as alternate to XML to send and recive data	(k) Very much used in SOAP based web services to send and recieve data (SOAP message are xml messages)
(l) JSON is less structured compare to Xml	(l) Xml is more structured..
(m) does not support commenting	(m) supports commenting
(n) Less secured becoz it just contains key and values	(n) More secured becoz data is having hierachy strcuture and namespaces procted..
(o) Allows only UTF-8 Characters	(o) Allows lots of Charsets including UTF-8

Dec 08.1 Working with JSON Data

Making RestController methods (service methods) sending Java Object data as JSON data

```
//RestController
package com.nt.controller;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import com.nt.model.Customer;
@RestController
public class CustomerOperationsController {
    @GetMapping("/report")
    public ResponseEntity<Customer> showData(){
        Customer cust=new Customer(1001,"raja",54566.66f);
        HttpStatus status=HttpStatus.OK;
        return new ResponseEntity<Customer>(cust,status);
    }
}
```

```
//model class
package com.nt.model;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class Customer {
    private Integer cno;
    private String cname;
    private Float billAmt;

}
```

from POSTMAN

GET http://localhost:3030/SpringBootRestProj03-SendingJSONDataAsResponse/report (SEND)

response body

```
{
   "cno": 1001,
   "cname": "raja",
   "billAmt": 54566.66
}
```

Sending Complext obj data as the complext JSON Data to Client from RestController

```
// In RestControler class
@GetMapping("/report1")
    public ResponseEntity<Customer> showData1(){
        //body
        Customer cust=new Customer(
                1001,"raja",54566.66f,
                    new String[] {"read","green","blue"},
                    List.of("10th","10+2","B.Tech"),
                    Set.of(544535345L,7576575L,6465654L),
                    Map.of("aadhar", 35345435L, "panNo", 354353534L),
                    new Company("SAMSUNG","hyd","Eletronics",4000));

        //status
    HttpStatus status=HttpStatus.OK;
    return new ResponseEntity<Customer>(cust,status);
  }
```

```
package com.nt.model;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class Company {
  private String name;
  private String addrs;
  private String type;
  private Integer size;
}
```

Model class

Customer.java

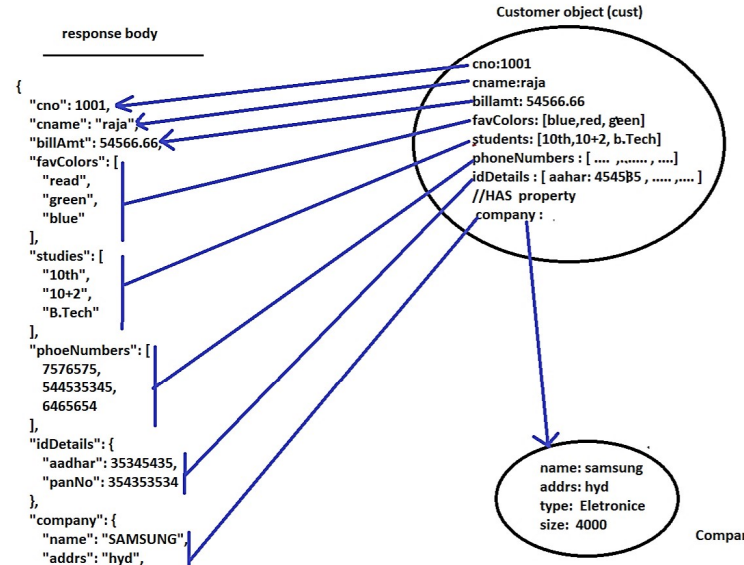
```
package com.nt.model;

import java.util.List;
import java.util.Map;
import java.util.Set;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class Customer {
    private Integer cno;
    private String cname;
    private Float billAmt;
    private String[] favColors;
    private List<String> studies;
    private Set<Long> phoeNumbers;
    private Map<String,Object> idDetails;
    //HAS- A property
    private Company company;

}
```

POSTMAN Tool

GET http://localhost:3030/SpringBootRestProj03-SendingJSONDataAsResponse/report1 Send

response body

```
{
  "cno": 1001,
  "cname": "raja",
  "billAmt": 54566.66,
  "favColors": [
    "read",
    "green",
    "blue"
  ],
  "studies": [
    "10th",
    "10+2",
    "B.Tech"
  ],
  "phoeNumbers": [
    7576575,
    544535345,
    6465654
  ],
  "idDetails": {
    "aadhar": 35345435,
    "panNo": 354353534
  },
  "company": {
    "name": "SAMSUNG",
    "addrs": "hyd",
    "type": "Eletronics",
    "size": 4000
  }
}
```

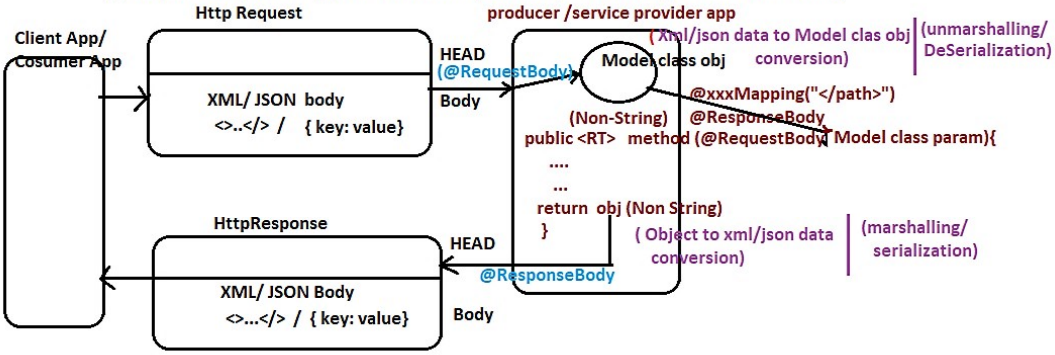


Dec 10 Response as XML or JSON Data

Working with Media Type Annotations (@RequestBody and @ResponseBody)

=>Using @RequestBody for the service provider/producer method param we can convert given xml or JSON data into given Model class (java bean class obj).

=>Using @ResponseBody on the top of service provider/producer method we can convert non-String content into xml or JSON data and can send to client as response body.
(@ResponseBody will be applied automatically on every method of @RestController class)



Making Spring Rest Producer App Sending XML Response body to Client App (Consumer App)

=> Based on given request type, making producer App sending different type of content with content type is techically called Content negotiation.

=> For this we need to use "Accept" request header specifying content type like "JSON", "XML" and etc.. This make producer application given content in response in the requested format ..
Though "Accept" is request header , it makes the Producer/server/service provider App sending its content in required format like JSON, XML and etc..

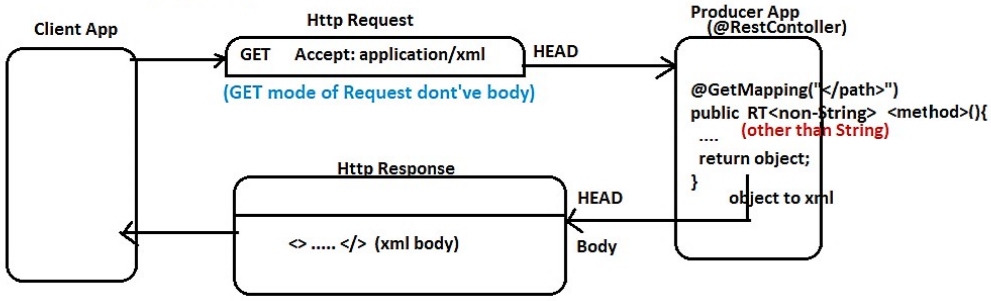
=>By default every method in Producer App sends palin text response if the return type is ResponseEntity<String> otherwise (for RespinseEntity<non-String> it sends JSON response)

Making Producer Application Sending XML response

=>For this the Client app/consume App must set "Acccept" request header to "applicaiton/xml" (default is */*)
Accept: application/xml tells to producer App give me response content as xml content
Accept: */* tells to Producer Apps give the response in any format (what u want, that u can give)

=> We need to add the following addtional depenency(jar file) to the project for converting Object data to xml format

```
<!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.dataformat/jackson-
dataformat-xml -->
<dependency>
  <groupId>com.fasterxml.jackson.dataformat</groupId>
  <artifactId>jackson-dataformat-xml</artifactId>
  <version>2.12.5</version>
</dependency>
```



Example App

step1) create Spring Boot starter project having following starters, dependencices (spring web , lombok, jackson-dataformat-xml)
(starters) (dependencies)

step2) Develioe the Model class

```
//Model class
=============
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Customer {
      private Integer cno;
      private String cname;
      private String cadd;
      private Float billamt;
}
```

Dec 10.1 Response as XML or JSON Data

step3) Develop Producer as @RestController having method with other String Return type.

//Producer App

```
package com.nt.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import com.nt.model.Customer;
@RestController
public class CustomerOperationsController {

    @GetMapping("/report")
    public Customer showData() {
        Customer cust=new Customer(1001,"raja","hyd",44545.77f);
        return cust;

    }
}
```

not a recommended approach and moreover Default response status code 200 will be taken

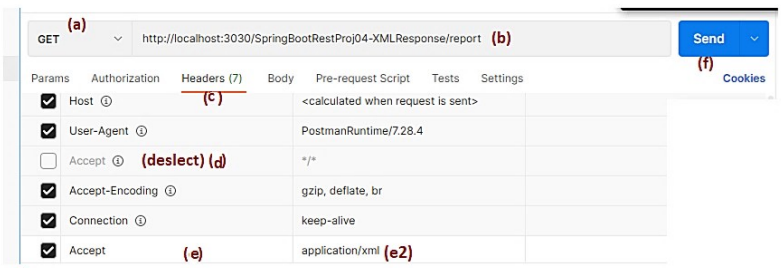
(or)

```
@GetMapping("/report")
    public ResponseEntity<Customer> showData() {
        Customer cust=new Customer(1001,"raja","hyd",44545.77f);
        return new ResponseEntity<Customer>(cust,HttpStatus.OK);

    }
```

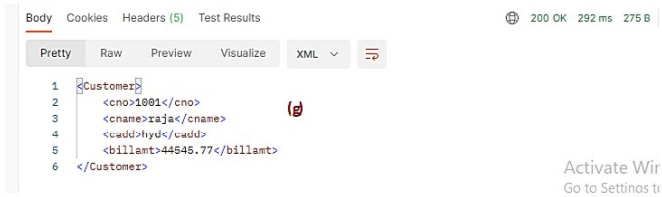
Taking ResponseEntity<T> as Return is recomanded becoz it gives control on response status code.

step4) Run the Applicaiton
(RunAs --- Run on server)

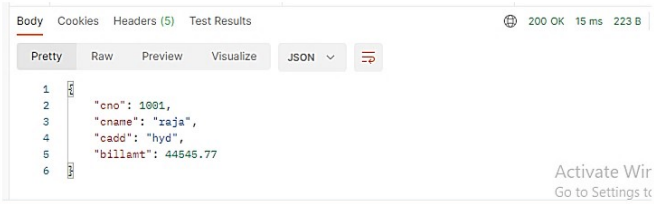
step5) Give Request from POSTMAN Tool by setting "Accept" header with the value "application/xml"



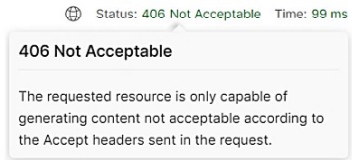
Response



=>if we send the above request with out placing Accept: application/xml request header then Accept:*/* will be take as default value and we get Json response as show below

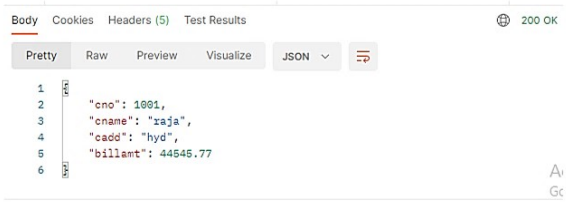


=>In the above setup where we asking for XML response using "Accept:application/xml" request header , if we are not placing "jackson-dataformat-xml" then we get 406 (Media Type Not Acceptable) Error



=> we run the same Application with the above setup (not adding jackson-dataformat-xml jar file) and with out chaging "Accept" header value to "applicaiton/xml" then we get JSON response fron producer App becoz the method return non-String type

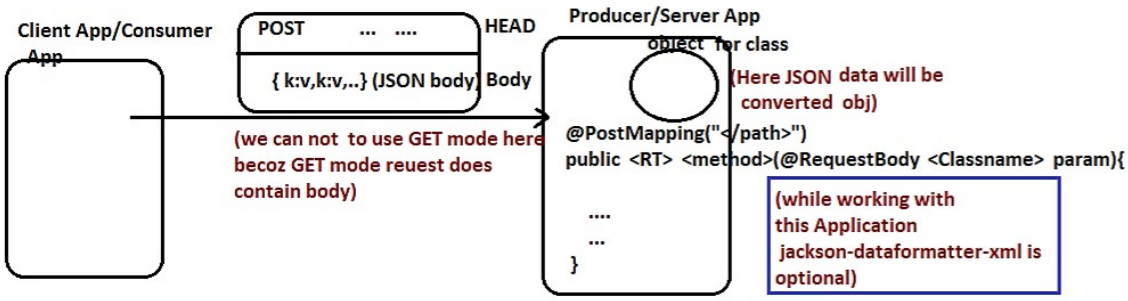
note:- Becoz giving the response in the JSON format is the default



Dec 11 Working with JSON - XML Data with @RequestBody

Working with @RequestBody to convert given XML/JSON content into Model class obj

=> if Client generated http request is having XML/JSON content as the body and if want to construct java class object having that data then we need to specify @RequestBody for the parameter of Producer method whose type is Model class name/Java class name



Syntax to use @RequestBody in method param

```
@RequestBody <ClassName> <paramName>
```

becomes obj name internally

=>GET, HEAD mode request does not contains body part.. but they send little amount of data as request param names and values in query String

=> The response of GET mode request contains both HEAD , BODY part where the response of HEAD mode request does not contains BODY part .. So we use GET mode request to get data from server and HEAD mode request to check wheather certains web comp is avaiable or not (Debugging purpose)

=> if JSON body is comming as { } (Empty folower bracket) then the Object for class given in @RequestBody will be ceated using 0-param constructor with default values given JVM (0/0.0/ null/false)

=> The keys in JSON body and the Property names in the give class of @RequestBody must match in order to see Data binding to object.. if few keys and property names are matching and others not matching then binding will happen only for few properties the remaining properties hold default values.

Example App

step1) Create spring Boot Project
adding the following starters , dependendencies
a) Lombok b) spring web c) jackson-dataformat-xml

note:- here for the dependency of jackson-dataformat-xml version not required

step2) Develop the Model class

```
//Model class
package com.nt.model;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Customer {
      private Integer cno;
      private  String cname;
      private  String cadd;
      private  Float billAmt;
}
```

step3) Devleop Producer App as @RestController

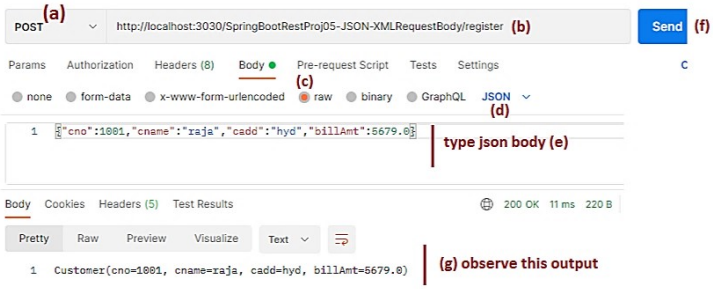
```
package com.nt.controller;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;
import com.nt.model.Customer;
@RestController
public class CustomerOperationsController {
      @PostMapping("/register")
      public  String  registerCustomer(@RequestBody  Customer cust) {
            return cust.toString();
      }
}
```

@RequestBody of spring Rest is very much similar to @ModelAttribute(-) of spring MVC

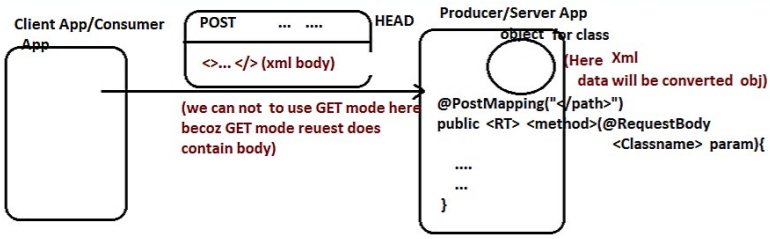
step4) Run the Application

Dec 11.1 Working with JSON - XML Data with @RequestBody

step5) Send the following request from POSTMAN



=> After taking request body as JSON/XML and if we do not send correct format body content along with the HttpRequest then we get 400 error response (Bad Request)

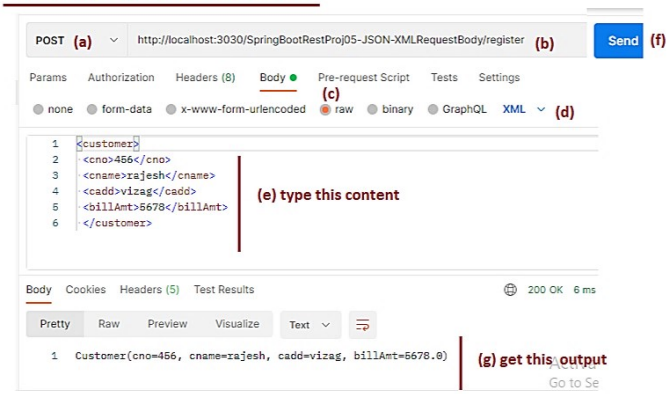


(while working with this Application jackson-dataformatter-xml is mandatory)

Example App (same as above)

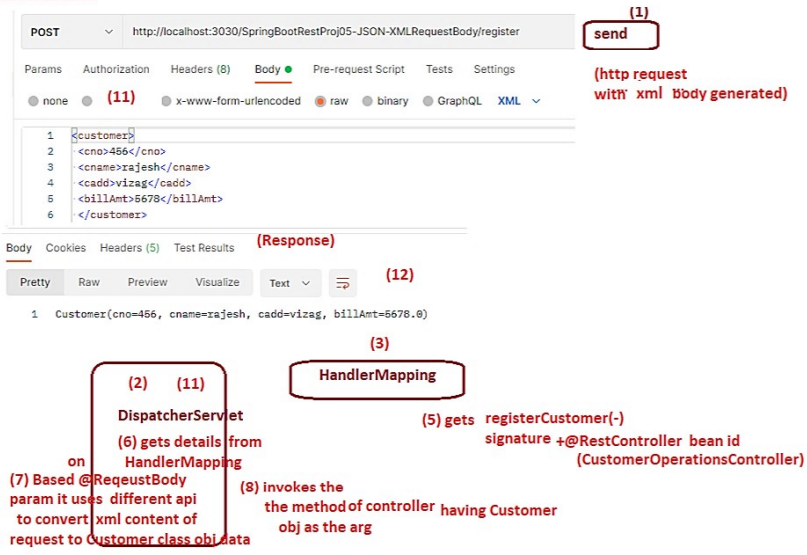
==========

Test the App using Post Main as show below



Flow of execution

==========




```
public class CustomerOperationsController {
                                                    (4?) searchê for Post mapping method with /register as reuest path
    @PostMapping("/register")   (9)
    public String registerCustomer(@RequestBody Customer cust) {
        return cust.toString();  (10)
    }
```

=> if request body content type is XML and the producer is not having jackson-datafromatter-xml iar file in the build path then we get 415 error (415 Unsupported Media Type)

conclusion

==========

406 :: we want Xml response through "Accept:application/xml" but to get jackson-dataformatter-xml jar is not added in the producer application (related @ResponseBody)

415 :: we want to send xml body along with request to store into Object using @RequestBody but jackson-dataformatter-xml jar is not added in the producer application

Dec 14 Working with JSON - XML Data with @RequestBody

Converting JSON body of Http request to Java class obj using @RequestBody

=>@ReqeustBody and @ResponseBody Annotations are called Media type annotations becoz they are useful to decide media of the incoming and outgoing content

=>@RequestBody is useful to convert JSON/XML data to Java class object and @ResponseBody is useful to convert Non-String data to JSON/XML data

=>@RequestBody dealing 1D and 2D Arrays JSON

1D array:: array/List/Set Collections "<variable>": [<val1>,<val2>,<val3>,...]

2D array :: Map "<variable>":{"<key1>":<val1>,"<key2>":val2,...}

2D Array :: HAS-A property "<variable>":{"<subProp1>":<val1>,"<subProp2>":val2,..}

Customer.java

```
package com.nt.model;

import java.util.List;
import java.util.Map;
import java.util.Set;

import lombok.Data;

@Data
public class Customer {
	private Integer cno;
	private String cname;
	private String[] favColors;
	private List<String> academics;
	private Set<Long> phoneNumbers;
	private Map<String,Double> billDetails;
	private Address addrs;

}
```

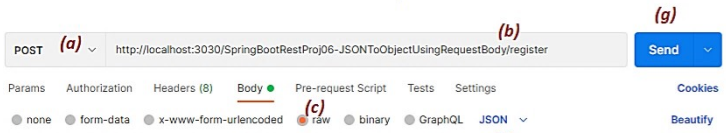
Address.java

```
package com.nt.model;
import lombok.Data;

@Data
public class Address {
	private String houseNo;
	private String streetName;
	private String location;
	private Long pinCode;

}
```

//Rest Controller class

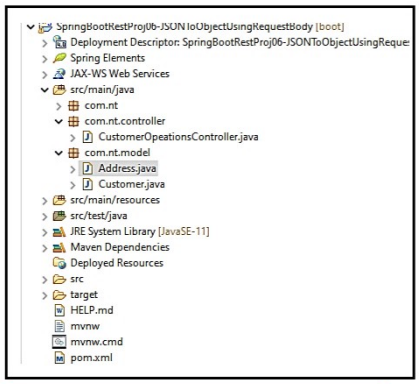
```
package com.nt.controller;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

import com.nt.model.Customer;

@RestController
public class CustomerOpeationsController {

   @PostMapping("/register")
   public ResponseEntity<String>  saveCustomer(@RequestBody Customer cust) {
	return new ResponseEntity<String>(cust.toString(), HttpStatus.OK);
   }
}
```

run as run on server




```
{
   "cno": 101,
   "cname": "rajesh",
   "favColors": [
      "red",
      "blue",
      "gree"
   ],
   "academics": [
      "10+2",
      "B.Tech",
      "M.Tech"
   ],
   "phoneNumbers": [
      5535353435,
      54353455,
      5353533543
   ],
   "billDetails": {
      "x-mas tree": 6546.77,
      "cake": 5345.5,
      "chocolates": 6789.6
   },
   "addrs": {
      "houseNo": "1-2-4/566",
      "streetName": "RKStreet",
      "location": "hyd",
      "pinCode": 522345
   }
}
```

(e) type this cotent



Note:- When you use List of Map objects then in json type like this

eg: List<Map<String,Integer>> ages;

```
"ages":[{
        "age1":"28",
        "age2":"29"
      }],
```

wait for this outoput

Dec 14.1 Working with JSON - XML Data with @RequestBody

Converting JSON Data to Java class object using @RequestBody dealing with List<Object>, date, time values

For List<Object> or Collection<Object> we need to take

```
"<variable>": [ {"key":value,"key":value,...} ,
               {"key":value,"key":value,...},
               {"key":value,"key":value,...}
               ]
```

for List of Map objects:-

in java class:-- List<Map<String,Integer>> xyz;

in Json of Postman:-

```
"<var name>":[{ "key1":"val1",    "key2":"val2"  }],
```

Company.java

```
package com.nt.model;

import lombok.Data;

@Data
public class Company {
    private String name;
    private String location;
    private Integer size;

}
```

application.properties

```
server.port=4040
server.servlet.context-path=/JSONToJavaObjectApp1
```

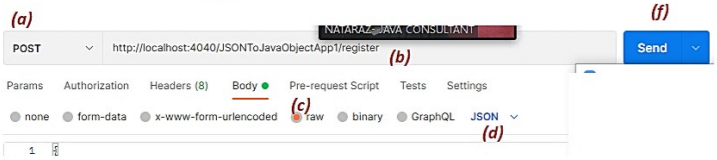
Customer.java

```
package com.nt.model;

import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.List;
import java.util.Map;

import com.fasterxml.jackson.annotation.JsonFormat;

import lombok.Data;

@Data
public class Customer {
    private Integer cno;
    private String cname;
    private List<Company> companiesList;
    @JsonFormat(pattern="yyyy-MM-dd")
    private LocalDate dob;
    //@JsonFormat(pattern="yyyy-MM-dd HH:mm:ss")
    private LocalDateTime purchaseDate;
    private List<Map<String,Long>> familyIds;

}
```

//RestController

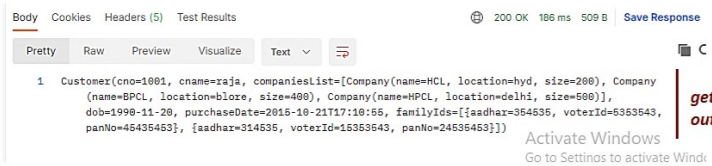
```
package com.nt.controller;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

import com.nt.model.Customer;

@RestController
public class CustomerOpeationsController {

    @PostMapping("/register")
    public ResponseEntity<String>  saveCustomer(@RequestBody Customer cust) {
        return new ResponseEntity<String>(cust.toString(), HttpStatus.OK);
    }
}
```

Note:- Run this application as spring Boo App




```
{
 "cno":1001,
 "cname":"raja",
 "companiesList":[
   {"name":"HCL","location":"hyd","size":200 },
   {"name":"BPCL","location":"blore","size":400 },
   {"name":"HPCL","location":"delhi","size":500 }
 ],
 "dob":"1990-11-20",
 "purchaseDate":"2015-10-21 17:10:55",
 "familyIds":[
   {"aadhar":354535,"voterId":5353543,"panNo":45435453},
   {"aadhar":314535,"voterId":15353543,"panNo":24535453}
 ]
}
```

(e) type this content



get this output

---

note:: if the request body contains invalid JSON Pattern content which can not be parsed by JSON parser then the RestController sends 400 (Bad Request) error response to browser.

Assingment ::

=> Build Json format body for the class Structure

```
class E_Commerce{
  private String name;
  private String addrs;
  private Integer size;
  private List<Courier> couriers;
  private Set<PaymentGateway> gatewaysInfo;
   ...
   ..
}
```

```
class Courier{
  ...
  ...
}
```

```
class PaymentGateway{
  ...
  ...
  ...
}
```

Dec 15 Passing RequestParam to SpringRestApp

Passing request Params to Spring Rest App

=>The GET mode request does not contain body .. So the data we want to send in GET mode request should always be in the form of Query String

=>The GET mode request can carry limited amount of data (max of 2KB) that to as Query String params

=> In other mode requests (like POST,PUT,DELETE and etc..) the content in the request goes to server in the form of request body.

For Spring RestController we can pass data in GET Mode request in two ways

a) As request Params/ Query param in the Query String of request URL
(eg: url?key1=val1&key2=val2&key3=val3
(Supported by Spring MVC and Spring Rest)

b) As Path variable values in the requst URL
(Supported by Spring Rest .. not in Spring MVC)
(eg: url/value/value/value

{key} will be given in request path of b.methods placed @RestController.

note:: All web technologies /frameworks (Both java and Non-Java) supports request params as the BASIC concept web programming
(eg: servlet,jsp,php,asp.net , nodejs, express js and etc..)

note:: Passing values in the request URL as PATH variable values in introduced in RestFull progrmaming , So the all tehcnologies and frameworks supporting the Restfull programming (both java and non-java) gives provision to work with this path variable concept

a) Request Params/ Query Params in query String

=>To place then in request url ,we need
URL?key1=val1

=>To read in the method of @RestController
@RequestParam("key") paramType paramName

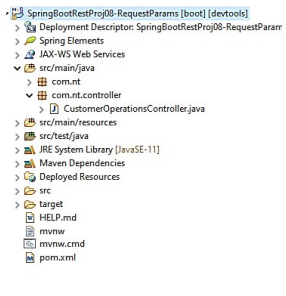
in the request url we must use "key" instead of "paramName"

(or) if you use "paramName" by having sprcific "key" you'll get 400 bad request

@RequestParam ParamType paramname

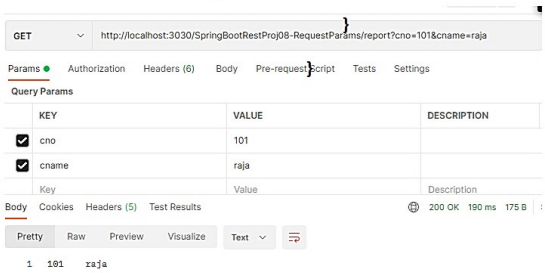
in the request url we must match with key of the request parameter

Example App



RestController class

```
package com.nt.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class CustomerOperationsController {

 @GetMapping("/report")
 public String reportData(@RequestParam("cno") Integer no,
                          @RequestParam String cname) {
      return no+"   "+cname;
 }
}
```



=>while using request params in query String we can change the order of passing req param values

report?cname=raja&cno=101
(or)
report?cno=101&cname=raja

Both are correct

=> while using request params in query String if we pass addtional params that expected that will not generate any eror

?cname=raja&cno=101&cadd=hyd
Extra but does not generate error

=>when we pass queryString to the url in POST mode request.. they internally become request body content becoz the POST carries data as request body.

```
@RestController
public class CustomerOperationsController {
     @GetMapping("/report")
     public String reportData(@RequestParam("cno") Integer no,
                              @RequestParam(required = false) String cname) {
          return no+"   "+cname;
     }
}
```

URL :: http://localhost:3030/SpringBootRestProj08-RequestParams/report?cno=101
gives 101 NULL as response

```
@RestController
public class CustomerOperationsController {

     @GetMapping("/report")
     public String reportData(@RequestParam("cno") Integer no,
                              @RequestParam(required = true) String cname) {
          return no+"   "+cname;
     }
}
```

(default value of required param is true)

URL :: http://localhost:3030/SpringBootRestProj08-RequestParams/report?cno=101

=>Gives 400 bad request

Dec 15.1 Passing RequestParam to SpringRestApp

Send data as Path variable values

==========

=>Supports to minimise characters in request url while sending data

=> No need of passing seperate query String in the request URL to send data, we can pass data directly in the request url it self

=>passing "&,=" as the values in request param values is not possible .. but can be done easily using path variable

=> request URL with query String is not Clean url becoz it needs more chars (becoz both keys and values are required) and queyString needs seperate syntax tp fallow

=> Rquest URL with path variable values are part of request url itself and no need of following seperate syntax for it.

syntax :: rquest url (or) Path/<value1>/<value2>/<value3>/ .....

To read path variable values in @RestController methods

{key} for those values will give at @RestController in XxxMapping(...) method while defining the request path

@PathVariable Datatype paramname (or) @PathVariable("key") dataType param name → param name must match {key} of request path

here we use this "key" only in the path of @GetMapping of Java class instead of "paramName" without following this rule if you proceeds to next level at the time of sending data will get 500 error

=>The request path contains two parts while working path variables

a) static path (fixed path) (/<path>)

b) Dynamic path (key name whose value comes from request url) (/{key})

eg:: GetMapping("/report/{no}/{name}")

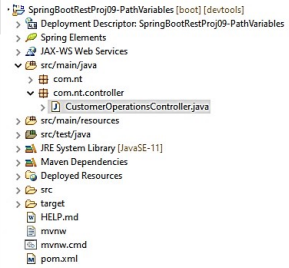
static path: /report — Dynamic path: /{no}/{name}

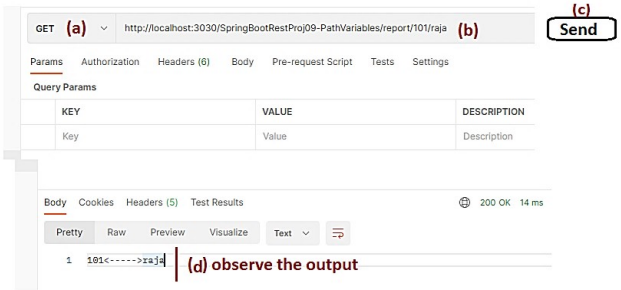
example rquest url

http://localhost:3035/SpringRestProj9-PathVariables/report/101/raja

static path: /report — path variable values (Dynamic path values): /101/raja

```
//RestController
================
package com.nt.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class CustomerOperationsController {

    @GetMapping("/report/{no}/{name}")
    public String fetchData(@PathVariable("name")String cname,
                            @PathVariable Integer no) {
        return no+ "<----->"+cname;
    }

}
```





While working with request param we can change the order of passing their values

/report?cno=101&cname=raja

is same as

/report?cname=raja&cno=101

While working parth variable values we can not change the order of passing values

request path in @RestController method is :: GetMapping("/report/{no}/{name}")

/report/101/raja --> valid

/report/raja/101 ---> invalid -->Gives 400 Bad request if @PathVariable Integer no is taken

```
@GetMapping("/report/{no}/{name}")
    public String fetchData(@PathVariable("name")String cname,
                            @PathVariable Integer no) {
        return no+ "<----->"+cname;
    }
```

Raises Error

/report/raja/101 ---> valid --> if both params are taken as String values but the wrong will be stored in method params so the b.logic will be distrubed

```
@GetMapping("/report/{no}/{name}")
    public String fetchData(@PathVariable("name")String cname,
                            @PathVariable String no) {
        return no+ "<----->"+cname;
    }
```

Does not rais error but cname holds 101 and no holds raja which may distrub b.logics execution.

=>Giving extra path variable values (nothing but extra words path) in request url results 404 error

request path for @RestController method :: /report/{no}/{name}

request url :: http://localhost:3030/SpringBootRestProj09-PathVariables/report/101/raja/hyd/india

Extra dynamic values which are not required So 404 error will come

Dec 16 Path Variables

Working with Path Variables
==========

=>if we give more or less values as path variable values than expected then we get 404 error (requested resource is not found)

eg::

```
@RestController
public class CustomerOperationsController {

    @GetMapping("/report/{no}/{name}")
    public  String  fetchData(@PathVariable("name")String  cname,
                              @PathVariable Integer  no) {
        return  no+ "<----->"+cname;
    }
}
```

request url :: http://localhost:3030/SpringRestProj10/report/101/raja

gives 101<-----> raja

request url :: http://localhost:3030/SpringRestProj10/report/101/raja/hyd

404 error

request url :: http://localhost:3030/SpringRestProj10/report/101

404 error

request url :: http://localhost:3030/SpringRestProj10/report

404 error

```
@RestController
public class CustomerOperationsController {

    @GetMapping("/report/{no}/{name}")
    public  String  fetchData(@PathVariable(name="name",required = false)String  cname,
                              @PathVariable(required = false) Integer  no) {
        return  no+ "<----->"+cname;
    }

}
```

request url :: http://localhost:3030/SpringRestProj10/report/101

we expect 101 <---> null should come as the output but
we get 404 error becoz no.of levels in request path are 3 and we are giving only two ,So it says requested resource is not found

request url :: http://localhost:3030/SpringRestProj10/report,

we expect null <------> null but we get
404 error (requested resource is not found)

---

if multuple methods are having similar request paths having same no.of levels then the request that is having more static level matchings will get priority

```
@RestController
public class CustomerOperationsController {

    @GetMapping("/report/no/name")
    public  String  fetchData1(@PathVariable(name="name",required = false)String  cname,
                               @PathVariable(required = false) Integer  no) {
        return  "from FetchData1";
    }

    @GetMapping("/report/no/{name}")
    public  String  fetchData2(@PathVariable(name="name",required = false)String  cname,
                               @PathVariable(required = false) Integer  no) {
        return  "from FetchData2";
    }

    @GetMapping("/report/{no}/{name}")
    public  String  fetchData3(@PathVariable(name="name",required = false)String  cname,
                               @PathVariable(required = false) Integer  no) {
        return  "from FetchData3";
    }

    @GetMapping("/report/{no}/name")
    public  String  fetchData4(@PathVariable(name="name",required = false)String  cname,
                               @PathVariable(required = false) Integer  no) {
        return  "from FetchData4";
    }
}
```

http://localhost:3030/SpringBootRestProj09-PathVariables/report/101/name
output :: Fetch Data4

http://localhost:3030/SpringBootRestProj09-PathVariables/report/no/name
output :: Fetch Data1

http://localhost:3030/SpringBootRestProj09-PathVariables/report/no/rajesh
output :: Fetch Data2

http://localhost:3030/SpringBootRestProj09-PathVariables/report/101/rajesh
output :: Fetch Data3

Dec 16.1 Path Variables

```
@RestController
public class CustomerOperationsController {
	@GetMapping("/report/101/raja")
	public  String  fetchData1() {
		return  "from FetchData1";
	}

	@GetMapping("/report/{no}/{name}")
	public  String  fetchData2(@PathVariable(name="name",)String  cname,
					@PathVariable Integer  no) {
		return  "from FetchData2";
	}
}
```

url :: http://localhost:3030/SpringBootRestProj09-PathVariables/report/101/raja

output: fetchData1

url :: http://localhost:3030/SpringBootRestProj09-PathVariables/report/101/ramesh

output: fetchData2

---

if two methods of @RestController is having same request path with same no.of levels then there is possibility of getting IllegalStateException during the application startup

```
@RestController
public class CustomerOperationsController {

	@GetMapping("/report/101/raja")
	public  String  fetchData1() {
		return  "from FetchData1";
	}

	@GetMapping("/report/{no}/{name}")
	public  String  fetchData2(@PathVariable(name="name",required = false)String  cname,
					@PathVariable(required = false) Integer  no) {
		return  "from FetchData2";
	}

	@GetMapping("/report/101/raja")
	public  String  fetchData3(@PathVariable(name="name",required = false)String  cname,
					@PathVariable(required = false) Integer  no) {
		return  "from FetchData3";
	}
}
```

gives

Caused by: java.lang.IllegalStateException: Ambiguous mapping.
Cannot map'customerOperationsController' method

this method also gives on the specified url path

Caused by: java.lang.IllegalStateException: Ambiguous mapping.
Cannot map'customerOperationsController' method

What is the difference b/w Request params and path variables way of passing data in spring Rest App ?

Request params	Path variables
a) Syntax to pass data along with request url is url?key=val&key=val&.....	a) Syntax to pass data along with request url is url/<static path>/val1/val2/... {key}/{key}/.. will be defined in @RestController class
b) syntax to read request param values is @RequestParam datatype param (or) @RequestParam("key") datatype param	b) syntax to read path variabe values is @PathVariable datatype param (or) @PathVariable("key") datatype param
c) while passing request param values in the query string of request url the order need not to match we	c) while passing path variables values in the request url the order must be matched
d) if pass more than required request params in query String the we will not error	d) if we pass more than required path variable values in the request url then we get 404 error
e) if we pass less than required request params in query Strung then we get error only when required=false is not taken @RequestParam	e) if we pass less than required path variable values in the request url then we get 404 error
f) Does not allow to pass '&' as value direct value (we vmust use %26 for that)	f) Does not allows "/" as value
g) Supported by both spring MVC and Spring Rest	g) Supported only in Spring Rest
h) In all web technologies that are there to develop web applicaitons in different domains supports this feature basic concept to pass data	h) supported only in Restfull programming of all domains..
i) URL is not clean URL (more characters required in the url to pass data)	i) URL is clean URL (less characters required in the url to pass data)
j) Easy to read and interpet	j) complex to read to interpet.
k) Bit slow while sending data	k) Bit faster while sending the data
l) Recomaned to use in non RestFull Apps	l) very useful in Restful apps..
m) Generally used while working GET mode requests becoz data goes as query String	M) works with all modes of of request becoz data goes directly from url itself

Dec 17 MiniProject on Spring Rest

Mini Project using Spring Rest + Spring data JPA + Oracle + POSTMAN tool

=> The methods is @RestController are called b.methods or opertaions (best) or web operations
=> The opeations of @RestController can have flexible signatures.
=> we can place b.logic directly in the operations of @RestController .. but its recomanded to place in service class methods and invoke them from @RestController operations
=> Developing @RestController with multiple operations linked with Service ,DAO class methods performing CURD Operations is nothing but developing Server /Service Provider/ Rest API/ API/ Producer .

=> Getting API and its operation details nothing but Basic URL + request path + path variables info together is called gathering API and its end points

API development /Producer/Server /ServiceProvider

our Layered App

(UI to send req)
POSTMAN Tool --------> (Front Controller) DispatcherServlet ---> (Exception handling logic) RestController ---> (b.logic) Service class ---> (persistence logic) DAO class ----->DB s/w (repository)
(UI to get resp)

[Tool based consumer)

step1) Create spring Boot project adding
the following staters oracle driver, spring data jpa,
spring web , devtools, lombok api

step2) Develop Entity Class /Model class that is required in o-r mapping operations

Tourist.java

```
package com.nt.entity;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.NonNull;
import lombok.RequiredArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
@RequiredArgsConstructor
@Entity
@Table(name="REST_TOURIST")
public class Tourist {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Integer tid;

    @Column(length = 20)
    @NonNull
    private String name;
    @Column(length = 20)
    @NonNull
    private String city;

    @Column(length = 20)
    @NonNull
    private String packageType;
    @NonNull
    private Double budget;
}
```

step3) add the following properties in application.properties file
for data source cfg and jpa cfgs

application.properties

```
#Datasource
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=system
spring.datasource.password=manager

#Spring rest cfgs
server.port=4040
server.servlet.context-path=/RestMiniProject

# JPA cfgs
spring.jpa.database-platform=org.hibernate.dialect.OracleDialect
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
```

Dec 17.1 MiniProject on Spring Rest

step4) Develop the Respository Interface extending from JpaRepository

IToursitRepo.java

```
package com.nt.repo;
import org.springframework.data.jpa.repository.JpaRepository;
import com.nt.entity.Tourist;

public interface ITouristRepo extends JpaRepository<Tourist,Integer> {

}
```

Entity class name — @Id type

The spring boot generates InMemory Impl class for our ITourist Interface having Tourist Entity class based o-r mapping persistence logic for all methods available or inherited for JpaRepository

Repo

Note:- This kind of dynamic InMemory classes will be generated in the memory where apps run nothing but JVM memory of RAM.

Note:- In JVM Memory heap for objects & statck for methods/local varibles & premgen /metaspace (from java8 metaspace is named) for InMemory code generation/ native code generation

step5) Develop Service Interface adn Service Impl class for save Object operation

Service Interface

```
package com.nt.service;
import com.nt.entity.Tourist;
public interface ITouristMgmtService {
   public  String  registerTourist(Tourist tourist);
}
```

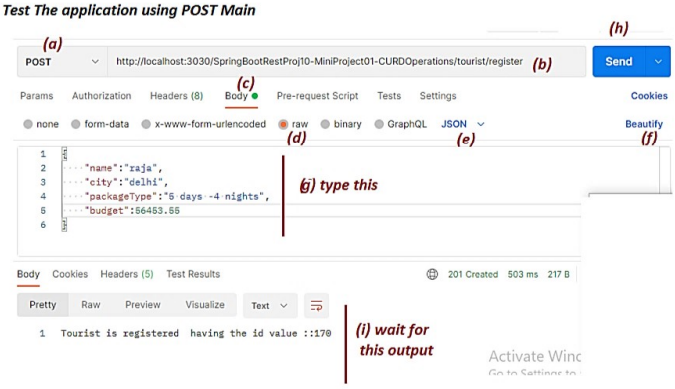
```
//service Impl class
package com.nt.service;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.nt.entity.Tourist;
import com.nt.repo.ITouristRepo;
@Service("touristService")
public class TouristMgmtServiceImpl implements ITouristMgmtService {
      @Autowired
     private ITouristRepo  touristRepo;
     @Override
     public String registerTourist(Tourist tourist) {
         int idVal=touristRepo.save(tourist).getTid();
          return "Tourist is registered  having the id value ::"+idVal;
     }
}
```

step6) Develope the Restcontroller having method for save operation

```
//Controller class
ackage com.nt.controller;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.nt.entity.Tourist;
import com.nt.service.ITouristMgmtService;
@RestController
@RequestMapping("/tourist")
public class TouristOperationsController {
     @Autowired
     private ITouristMgmtService  service;
     @PostMapping("/register")
     public ResponseEntity<String>  enrollTourist(@RequestBody Tourist tourist){
          try {
          //use service
          String resultMsg=service.registerTourist(tourist);
           return  new ResponseEntity<String>(resultMsg,
                                  HttpStatus.CREATED); //201 content created successfully
        }
        catch(Exception e) {
             return  new ResponseEntity<String>("problem in tourist enrollment",
         HttpStatus.INTERNAL_SERVER_ERROR); //500 error
        }
    }//method
}//class
```

step7) Run on Server application.

step8) Test The application using POST Main



Dec 18 MiniProject on Spring Rest

finalAll() operation in Mini Project

In service Interface

```
public List<Tourist> fetchAllTourists();
```

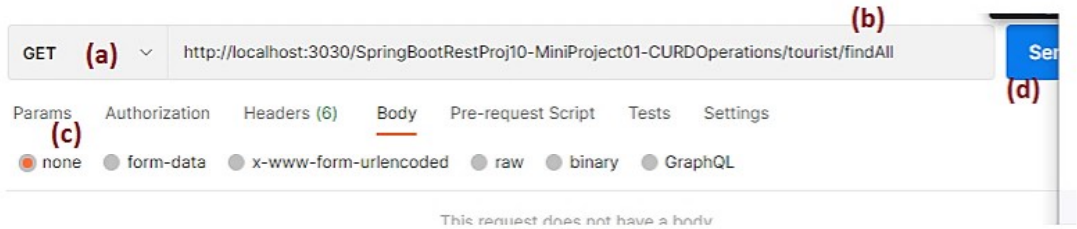
In service Impl class

```
@Override
public List<Tourist> fetchAllTourists() {
    List<Tourist> list=touristRepo.findAll();
    list.sort((t1,t2)->t1.getTid().compareTo(t2.getTid()));
    return list;
}
```

In controller class

```
@GetMapping("/findAll")
public ResponseEntity<?> displayToursits(){
    try {
        List<Tourist> list=service.fetchAllTourists();
        return new ResponseEntity<List<Tourist>>(list,HttpStatus.OK);
    }
    catch(Exception e) {
        e.printStackTrace();
        return new ResponseEntity<String>("Problem in fetching Tourists",
                HttpStatus.INTERNAL_SERVER_ERROR); //500 eror
    }
```

request from Postman tool




```
[
  {
    "tid": 170,
    "name": "raja",
    "city": "delhi",
    "packageType": "5 days -4 nights",
    "budget": 56453.55
  },
  {
    "tid": 171,
    "name": "ramesh",
    "city": "hyd",
    "packageType": "3 days -2 nights",
    "budget": 16453.55
  }
]
```

(e) get this output

Dec 18.1 MiniProject on Spring Rest

Performing findTouristByTid() operation

==========

//In service Interface

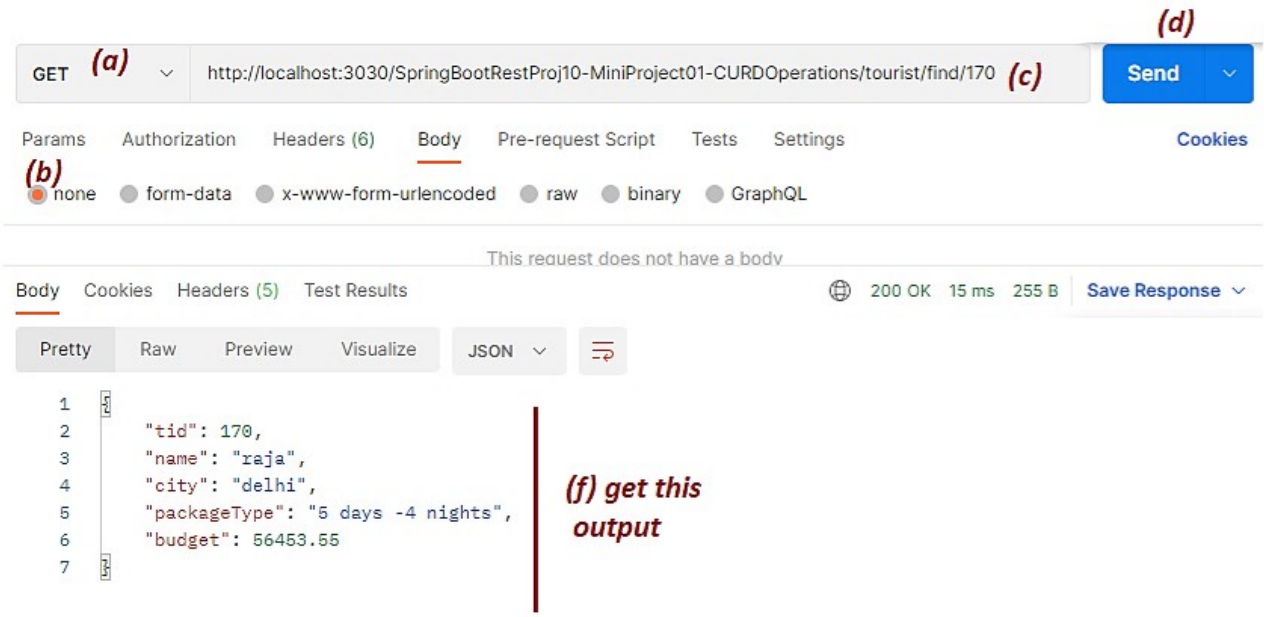
```
public Tourist fetchTouristById(Integer tid)throws TouristNotFoundException;
```

//In service impl class

```
@Override
public Tourist fetchTouristById(Integer tid) throws TouristNotFoundException {
      return touristRepo.findById(tid)
      .orElseThrow(()-> new TouristNotFoundException(tid+" tourist not found"));
}
```

In controller class

```
@GetMapping("/find/{id}")
      public ResponseEntity<?> displayTouristById(@PathVariable("id") Integer id){
            try {
                  Tourist tourist=service.fetchTouristById(id);
                  return new ResponseEntity<Tourist>(tourist,HttpStatus.OK);
            }
            catch(Exception e) {
                        e.printStackTrace();
                  return new ResponseEntity<String>(e.getMessage(),
                              HttpStatus.INTERNAL_SERVER_ERROR); //500
            }

      }//method
```



Dec 18.2 MiniProject on Spring Rest

performing update opeartion in Mini Project

In service Interface

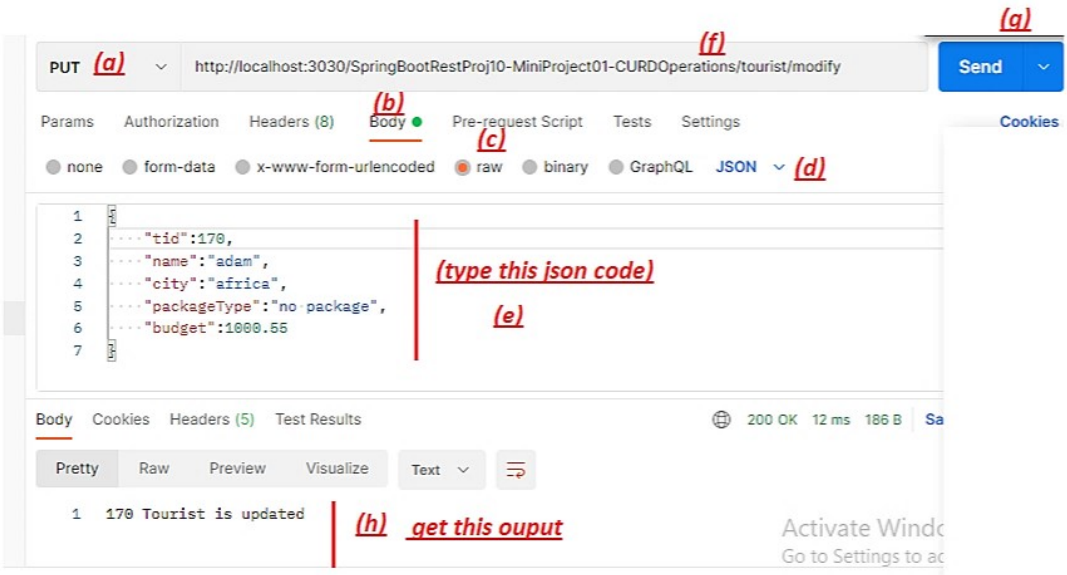
```
public   String   updateTouristDetails(Tourist tourist)throws TouristNotFoundException;
```

In service Impl class

```
@Override
public String updateTouristDetails(Tourist tourist) throws TouristNotFoundException {
      Optional<Tourist> optional=touristRepo.findById(tourist.getTid());
       if(optional.isPresent()) {
     touristRepo.save(tourist);// save(-) performs either save obj or update obj operation
     return  tourist.getTid()+" Tourist is updated";
       }
       else {
          throw new TouristNotFoundException(tourist.getTid()+" Tourist not found ");
       }

    }
```

In controller class

```
@PutMapping("/modify")
      public  ResponseEntity<String>  modifyTourist(@RequestBody Tourist tourist){
            try {
                  String msg=service.updateTouristDetails(tourist);
                  return new ResponseEntity<String>(msg,HttpStatus.OK);
            }

  catch(Exception e) {
e.printStackTrace();
return new ResponseEntity<String>(e.getMessage(),
                                  HttpStatus.INTERNAL_SERVER_ERROR);

 }
}
```



Dec 20 MiniProject on Spring Rest

Delele operation in Mini Project

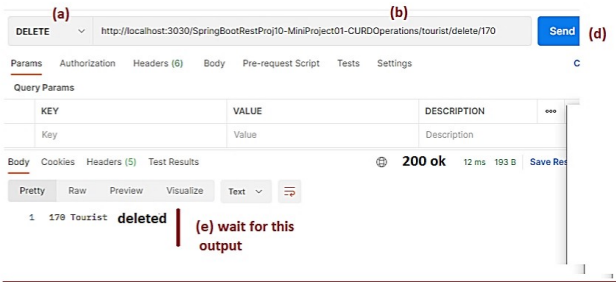
```
// In service Interface
```

```
public String deleteTourist(Integer tid)throws TouristNotFoundException;
```

```
// In service Impl class
@Override
public String deleteTourist(Integer tid) throws TouristNotFoundException {
        Optional<Tourist> opt=touristRepo.findById(tid);
        if(opt.isPresent()) {
              touristRepo.delete(opt.get());
              return tid+" Tourist deleted";
        }
        else {
           throw new TouristNotFoundException(tid+" Tourist not found ");
        }

    }
```

```
// In controller class
@DeleteMapping("/delete/{id}")
      public ResponseEntity<String> removeToursit(@PathVariable("id") Integer id){
            try {
            //use service
            String msg=service.deleteTourist(id);
             return new ResponseEntity<String>(msg,HttpStatus.OK);
            }
            catch(Exception e) {
                  e.printStackTrace();
                  return new ResponseEntity<String>(e.getMessage(),
                                                      HttpStatus.NOT_FOUND);
            }

       }//method
```

From Postman tool



Performing partial update using @PatchMapping

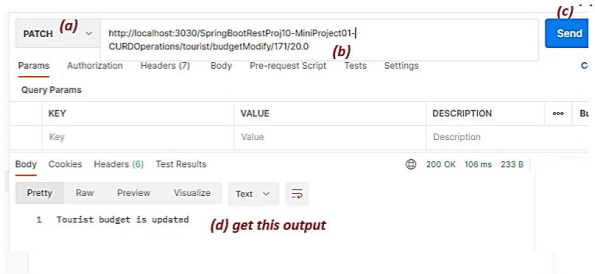
```
// In service Interface
```

```
public String updateTouristBudgetById(Integer id, Float hikePercent)throws
TouristNotFoundException;
```

```
//In service Impl class
@Override
public String updateTouristBudgetById(
                        Integer id, Float hikePercent) throws TouristNotFoundException {

   Optional<Tourist> opt=touristRepo.findById(id);
   if(opt.isPresent()) {
       Tourist tourist=opt.get();
       tourist.setBudget(tourist.getBudget()+(tourist.getBudget()*(hikePercent/100)));
       touristRepo.save(tourist);
       return "Tourist budget is updated";
   }
   else {
      throw new TouristNotFoundException(id+" Tourist not found");
   }
}
```

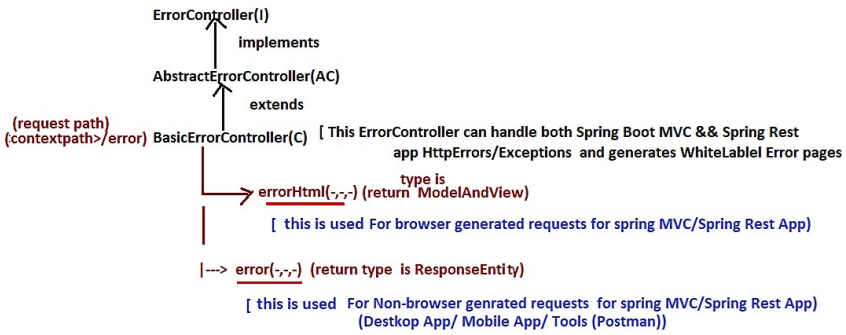
```
// In controller class
@PatchMapping("/budgetModify/{id}/{hike}")
      public ResponseEntity<String> modifyTouristBudgetById(
            @PathVariable("id") Integer id, @PathVariable("hike") Float hikePercent){

     try {
            //use service
             String msg=service.updateTouristBudgetById(id, hikePercent);
             return new ResponseEntity<String>(msg,HttpStatus.OK);
      }//try
     catch(Exception e) {
         e.printStackTrace();
         return new ResponseEntity<String>(e.getMessage(),
                                                HttpStatus.NOT_FOUND);
     }

      }
```

In Postman tool

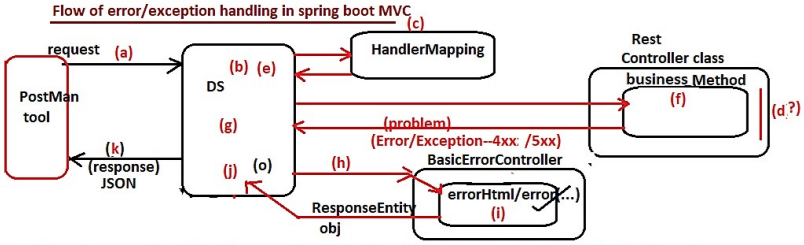


Dec 21 ErrorController-ControllerAdvice in Spring Rest

=> spring Rest/ Spring Boot MVC is having pre-defined Controller to handle these HttpErrors/exceptions and to give default whiteLabel Error pages



Flow of error/exception handling in spring boot MVC

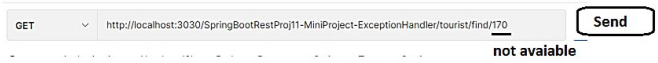


=>if we do not catch and handle exception in the b.methods of RestController (API) class then the DS gets the Propagated Exception and gives to error(-,-,-) of pre-defined controller class BasicErrorController which is mapped with <requestpath>/error (like tourist/find/error) .
=>the error(-,-,-) of BasicErrorController class returns ResponseEntity<Map<String,Object>> obj having default error messages to DS and DS converts those messages to JSON Details to send to Client as error Json Response.

To feel this pratically
==========

a) open BasicErrorController class using ctrl+shit+T option

b) get list of methods using ctrl+ o --> open error(-,-,-) source code
c) keep one breakpoint inside the error(-,-,-) method using ctrl+b or using double click in left margin
d) Run the App using Debug As server option
e) Given give request from POSTMain causing exception



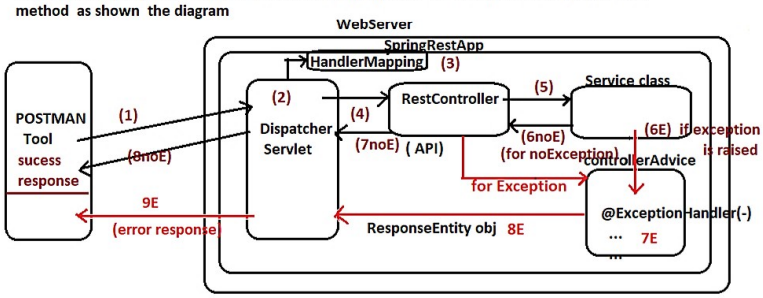
not avaiable

f) press F8 button to move control from default errorHtml(-,-,-) method to error(-,-,-) method then continuosly press f6/f7 button to go further...

g) object ResponseEntity<Map<String,Object>> object based JSON Error response in POSTMAN tool as shown below

f5 is step into
f6 step over
f7 step return
f8 next break point

=>Instead of using BasicErrorController to handle the exceptions raised or propagated to RestController methods we can use custom throws advice class that is developed using @ControlAdvice(or) @RestControllerAdvice + @ExceptionHandler which can return ResponseEntity object to DispatcherSErvlet directly by catch the exception from Service class method as shown the diagram



note:: The @ControllerAdvice or @RestControllerAdvice class @ExceptionHandler methods respond to execute for exceptions raised in Repository, service, RestController classes ..

Example App
==========

step1) Keep Mini Project ready

step2) Develop ErrorDetails class the model class to hold more details about exception

```
//ErrorDetails.java
@Data
@NoArgsConstructor
@AllArgsConstructor
public class ErrorDetails {
      private LocalDateTime time;
      private String msg;
      private String status;

}
```

Dec 21.1 ErrorController-ControllerAdvice in Spring Rest

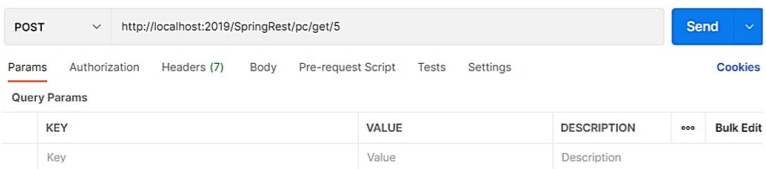
step3) Develop ControllerAdvice class as shown below

```
@ControllerAdvice
@RestControllerAdvice
public class TouristErrorHandler {

 @ExceptionHandler(TouristNotFoundException.class)
 public  ResponseEntity<ErrorDetails>  handleTouristNotFound(TouristNotFoundException tnf ){
       System.out.println("TouristErrorHandler.handleTouristNotFound()");
        ErrorDetails details=new ErrorDetails(LocalDateTime.now(),tnf.getMessage(),"404- Tourist Not Found");
        return new ResponseEntity<ErrorDetails>(details,HttpStatus.NOT_FOUND);
 }

    @ExceptionHandler(Exception.class)
    public  ResponseEntity<ErrorDetails>   handleAllProblems(Exception e){
          System.out.println("TouristErrorHandler.handleAllProblems()");
           ErrorDetails details=new ErrorDetails(LocalDateTime.now(),e.getMessage(),"Problem in execution");
           return new ResponseEntity<ErrorDetails>(details,HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

step4) Remove try catch blocks in the methods of RestController class

```
@RestController
@RequestMapping("/tourist")
public class TouristOperationsController {
      @Autowired
       private ITouristMgmtService  service;

       @PostMapping("/register")
       public ResponseEntity<String>  enrollTourist(@RequestBody Tourist tourist)throws Exception{
             //use service
             String resultMsg=service.registerTourist(tourist);
              return  new ResponseEntity<String>(resultMsg,
                                                 HttpStatus.CREATED); //201 content created successfully
       }//method
       @GetMapping("/findAll")
       public  ResponseEntity<?>  displayToursits() throws Exception{
                  List<Tourist> list=service.fetchAllTourists();
                 return new ResponseEntity<List<Tourist>>(list,HttpStatus.OK);
       }

       @GetMapping("/find/{id}")
       public  ResponseEntity<?> displayTouristById(@PathVariable("id") Integer id)throws Exception{
             System.out.println("TouristOperationsController.displayTouristById() --before");
              Tourist  tourist=service.fetchTouristById(id);
              System.out.println("TouristOperationsController.displayTouristById() --after");
                   return new ResponseEntity<Tourist>(tourist,HttpStatus.OK);

       }//method

      @PutMapping("/modify")
           public  ResponseEntity<String>  modifyTourist(@RequestBody Tourist tourist)throws Exception{
                       String msg=service.updateTouristDetails(tourist);
                       return new ResponseEntity<String>(msg,HttpStatus.OK);

           }

        @DeleteMapping("/delete/{id}")
        public  ResponseEntity<String>  removeToursit(@PathVariable("id") Integer id)throws Exception{
              //use service
             String  msg=service.deleteTourist(id);
              return  new ResponseEntity<String>(msg,HttpStatus.OK);

        }//method

        @PatchMapping("/budgetModify/{id}/{hike}")
        public  ResponseEntity<String>  modifyTouristBudgetById( @PathVariable("id") Integer id,
                                                 @PathVariable("hike") Float  hikePercent)throws Exception{
          //use service
           String msg=service.updateTouristBudgetById(id, hikePercent);
           return new ResponseEntity<String>(msg,HttpStatus.OK);
     }
    }
```

step4) Run the Application causing exception..



note : -- **if you try request to get tourist details which is not available in the db table then also it should give error response but, not in its own / same old format, it should give response having format in our own defined format those are specified details given in ErrorDetails.java" class. becoz we are explicitly created ErrorHandling class.. according to that sholud get error response also**

Same Or Old Error Response Format:--

```
{
    "timestamp": "2021-12-21T02:04:51.396+00:00",
    "status": 500,
    "error": "Internal Server Error",
    "path": "/SpringBootRestProj11-MiniProject-ExceptionHandler/tourist/find/170"
}
```

New Error Response in Our own Format:-

```
{
    "time": "2021-12-21T21:50:08.3870323",
    "msg": "sorry the given id 5 is not valid",
    "status": "404    not found exception"
}
```

tourist

Dec 22 Swagger -Api documentation

API Creation/Development :: developing RestController having different methods/opertions for various http method types like GET,POST,PATCH,PUT,DELETE and etc... is called API Creation/Development.

=>We can generally take one @RestController per 1 module , So developing each RestController is called api creation..

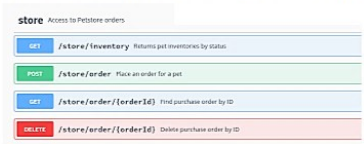
Accounts module ---> AccountsController (@RestController) is called Acccounts API creation
Daily Tx module ---> DialyTxController (@RestController) is called DailyTx API creation and etc..

EndPoints :: Providing muliple details or collection of details that are required to call methods/opertions of @RestController from Client Apps or to send reqeusts from different tools like POSTMAN is called Providing end points..

Eg:: For AccountController nothing but Accounts API the end points are

BaseURL :: http://localhost:3030/RestProj1/tourist
register() :: /register ----> POST
findById(-) :: /find/{id} ----> GET
deleteById(-) : /delete/{id} --->DELETE
and etc..

In End Points of any API we need to provide multiple details like URL ,method names, request paths , http method types ,content type and etc..



API Documentation
==========

=>we can write documentation for java classes in multiple ways
a) using serperate Text docs
b) using API documentation comments and javadoc tool

note:: Both these approaches non-responsive documentations i.e we can read about java classes and methods but we can not test them immediately

=> After developing Rest API we can provide API documentation for Rest API in the following ways

a) Using Seperate Text docs
b) Using API doc comments (/** ... */) | Creates Non-Responsive API documentation
c) Using Swagger /Swagger API (Best)
(or) Open API | Creates Responsive API Documentation.

note:: Open API is alternate to Swagger API .. The indurstry standard is still Swagger API

Swagger API
==========

=> It is an open Source Third Party Library to provide Responsive API documentation for RestController and its methods
=> For All RestControllers of the Project we can create API documentation from single place while working Swagger API
=>Responsive Documnentation means not only we get docs about API and its methods (End points) we can test immediately by providing inputs and by getting outputs..
=> if this is used .. there is no need of using POSTMAN tool seperately.. and also very useful to provide Documentation based Testing env.. from Clients.

=> Spring Fox +swagger together released libraries that are required to use swagger api in spring boot Applications

=>Swagger API documentation provides the following details in the GUI Responsive docs
a) API Info (company, title, license url , and etc..)
b) End points Info
c) Model classes info
and etc..

=> While working swagger documentation for reading and testing we need not to remember and give URL, Http method types , content type and etc.. we just need to give required inputs and get the outputs.

Procedure to work with swagger API
==========

step1) keep RestController /Rest API Project ready..

step2) Add the following two jar file in pom.xml related to swagger API

a) springfox-swagger2
a) springfox-swagger-ui

```
<!-- https://mvnrepository.com/artifact/io.springfox/springfox-swagger-ui -->
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.9.2</version>
</dependency>

<!-- https://mvnrepository.com/artifact/io.springfox/springfox-swagger2 -->
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.9.2</version>
</dependency>
```

Dec 22.1 Swagger -Api documentation

step3) Develope seperate Configuration class Enabling enabling Swagger api

=>In configuration class create Docket object having
->Dcoumention type (screent type)
->specify base package of restControllers
->specify reqeusts paths info
->other details of API (ApiInfo obj having company name, licenseurl and etc.)

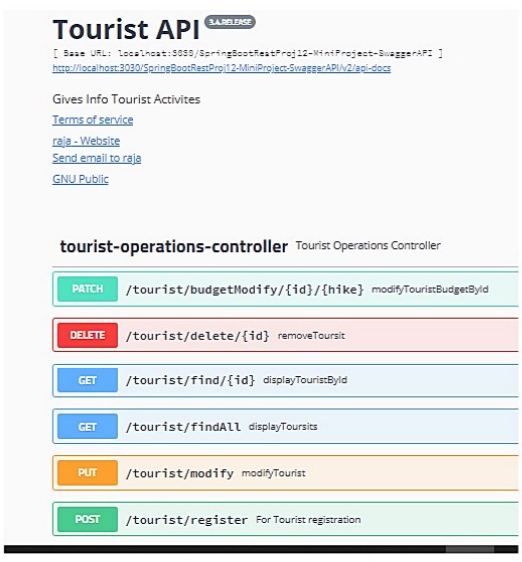
SwaggerDocConfig.java

```
package com.nt.config;

import java.util.Collections;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import springfox.documentation.builders.PathSelectors;
import springfox.documentation.builders.RequestHandlerSelectors;
import springfox.documentation.service.ApiInfo;
import springfox.documentation.service.Contact;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;
import springfox.documentation.swagger2.annotations.EnableSwagger2;

@Configuration
@EnableSwagger2
public class SwaggerDocsConfig {
@Bean
 public  Docket createDocket() {
 return  new Docket(DocumentationType.SWAGGER_2)  //UI screen type
       .select()  //to specify  RestControllers
       .apis(RequestHandlerSelectors.basePackage("com.nt.controller")) //base pkg for RestContrllers
       .paths(PathSelectors.regex("/tourist.*")) // to specify request paths
       .build() // builds the Docket obj
       .useDefaultResponseMessages(true)
       .apiInfo(getApiInfo());
 }

 private  ApiInfo getApiInfo() {
Contact contact=new Contact("raja","http://www.HCL.com/tourist","natarazjavaarena@gmail.com");
 return  new ApiInfo("Tourist API",
                     "Gives Info Tourist Activites",
                     "3.4.RELEASE",
                     "http://www.hcl.com/license",
                     contact,
                     "GNU Public",
                     "http://apache.org/license/gnu",
                     Collections.emptyList());
 }

 }
```

step4) Run the Server app

step5) use the following url to get swagger api docs and to test the api

http://localhost:3030/SpringBootRestProj12-MiniProject-SwaggerAPI/swagger-ui.html



Dec 23 RestTemplate

Developing Consumer App using RestTemplate

=> It allows to develop the consumer/Client App RestFull webService as Programmable Client App in java env..

=> We need to take seperate WebService/MVC Project for this having logics to consume webService /API by calling methods.

=> This object (RestTemplate) does not come through AutoConfiguration Process .. It must be created either using "new" operator or using @Bean method

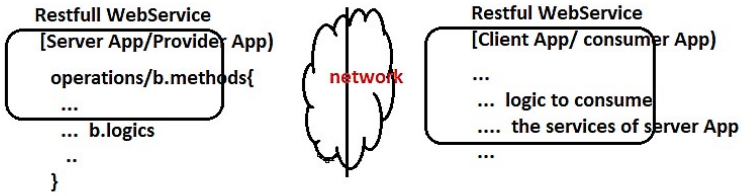
```
RestTemplate template=new RestTemplate();
        (or)
In @Configuration class
----------------------------------
  @Bean("template")
  public  RestTemplate createTemplate(){
        return new  RestTemplate();
   }
```

=> This object provide methods to generate different modes requests like GET/POST/PUT/DELETE/.... to consume the Restfull webService /API.. i.e we can call methods /operations Restful webService Server App/provider App

=> While using this object to cosume RestFull WebService/API we need detailed inputs (nothing but end points) like base url , http method type, http header info like content type and etc..

=> It provides xxxForEntity(....) methods like getForEntity(...) , postForEntity(...) and etc.. taking url, request obj(body,header) to send different modes http requests as method calls to cosume the the Restfull web service (Server/Provider App)

=> Do not forget WebService is given to link two different Apps that are developed either in same language or in diferent languages and in running same server or different servers belonging to same machine or different machines.



Server/Provider App — Client /Cosumer App

PaymentGateway <----------> PayTM App

PayTM/GooglePay/PhonePay<----------> Flipkart/Amazon

WeatherReport App <----------> yahoo.com/Tourist.com

ICC Score Comp <----------> CrickInfo.com, CrickBuzz.com and etc..

=>The RestFullWebService (Server/ provider App) must be the web application

=> The consumer App can be the standalone App or mobile App or IOT App or Web application or etc..

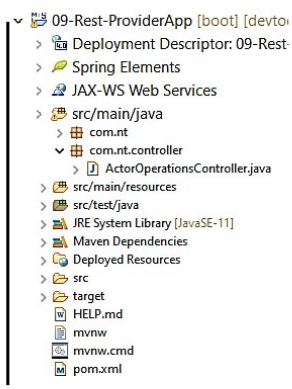
So far we have devleop only RestFul WebService(Server/provider App) and we tested that server App using tools like POSTMAN/Swagger ... Instread of using these tools we can develop programmable Real client Apps with the support RestTemplate in spring Env..

=>RestTemplate can be used only in Spring or Spring Boot env..

Example App

step1) Develop Restful WebService App as web application (Server /provider App)
(old style App)

staters :: web, dev tools, lombok api



step2) Develop API/Rest controller

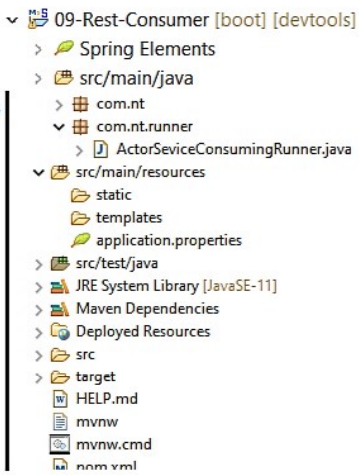
```
package com.nt.controller;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/actor")
public class ActorOperationsController {

   @GetMapping("/wish")
   public  ResponseEntity<String>  displayWishMessage(){
         return  new ResponseEntity<String>("Good Morning",HttpStatus.OK);
   }
  }
```

step3) Run The application...on server (Run As --->Run on SErver)

Dec 23.1 RestTemplate

Process-1: run both Server(Provider) & Consumer(Client) Apps on External Server

step4) Develop the Consumer App as seperte Project
(starters :: web , dev tools , lombok)



step5) place the following entries in application. properties
server.port=4040

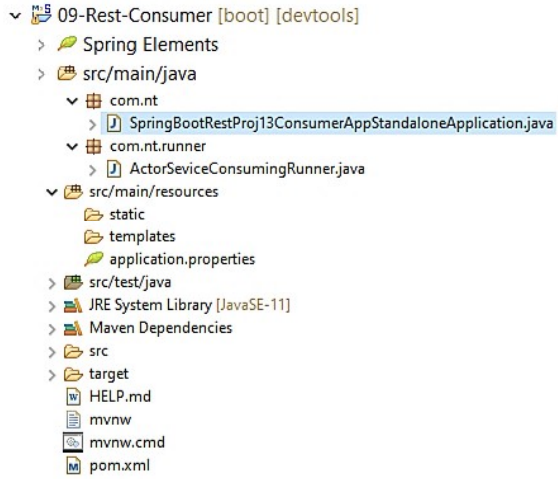
step6) Develop the Runner App

```
package com.nt.runner;

import org.springframework.boot.CommandLineRunner;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Component;
import org.springframework.web.client.RestTemplate;

@Component
public class ActorSeviceConsumingRunner implements CommandLineRunner {

@Override
public void run(String... args) throws Exception {
      //create RestTemplate class object
      RestTemplate  template=new RestTemplate();
      //Define  service url
       String serviceUrl="http://localhost:3030/SpringBootRestProj13-ProviderApp/actor/wish";
       // Generate Http reqeust   with GET mode to consume the web service(API)
       ResponseEntity<String> response=template.getForEntity(serviceUrl, String.class);
       //display the recieved details from the response
       System.out.println("Response body(output) ::"+response.getBody());
       System.out.println("Response status code value ::"+response.getStatusCodeValue());
       System.out.println("Response satus code ::"+response.getStatusCode().name());

       //System.exit(0); //optional

    }
 }
```

Process-2: Run Provider on Server & Consumer on Embedded Server(as Boot App)

step4) Run Consumer App as spring boot App that uses Embedeed Tomcat server..
Run As ----> spring Boot App/ jav aApp

note:: The above consumer App can also be develop as standalone app (package type is jar)
adding spriing web starters.. as shown below



Runner class code
and application.propeeties
content is same as the above
Consumer App..

Dec 24 RestTemplate

What is the difference b/w getforEntity(..) and getForObject(..) methods?

Ans) getForEntity(..) return type is ResponseEntity<T> which contains response body(output), headers, status code and etc.. (Best Approach)

getForObject(..) return type is Object which gives only response body(output) i.e it does not give response headers, status code and etc..

note:: when we develop spring web mvc/spring rest app as war file(web application) we can use both extenal server and Embedded Server (like Embedded Tomcat) for deployment.

note:: when we develop spring web mvc/spring rest app as jar file(standalone web application) we can use only Embedded server (like Embedded Tomcat) for deployment.

Passing Path variable values along with http request calls using RestTemplate :-

=>Both getForObject(...), getForEntity(...) are having multiple overloaded forms .. the form that is taking more params is maintaining last param as var args param which is given to pass any no.of path variable values along with http request call.

`<T> T`	`getForObject(String url, Class<T> responseType, Map<String,?> uriVariables)` Retrieve a representation by doing a GET on the URI template.	To pass path variable names and vlaues as map collection	Need not to fallow the order while passing values
`<T> T`	`getForObject(String url, Class<T> responseType, Object... uriVariables)` Retrieve a representation by doing a GET on the specified URL.	To pass path variable vlaues as var args..	Needs to fallow the order while passing values

`<T> ResponseEntity<T>`	`getForEntity(String url, Class<T> responseType, Map<String,?> uriVariables)` Retrieve a representation by doing a GET on the URI template.	To pass path variable names and vlaues as map collection	Need not to fallow the order while passing values
`<T> ResponseEntity<T>`	`getForEntity(String url, Class<T> responseType, Object... uriVariables)` Retrieve an entity by doing a GET on the specified URL.	To pass path variable vlaues as var args..	Needs to fallow the order while passing values

Example App

In server /producer / Provider App

RestControlller

```
@RestController
@RequestMapping("/actor")
public class ActorOperationsController {

    @GetMapping("/wish/{id}/{name}")
    public ResponseEntity<String> displayWishMessage(@PathVariable Integer id,
                                                     @PathVariable String name){
        return new ResponseEntity<String>("Good Morning::"+id+"..."+name,HttpStatus.OK);
    }

}
```

In Consumer App/Client App /

runner class

```
@Component
public class ActorSeviceConsumingRunner_PathVariables implements CommandLineRunner {

@Override
public void run(String... args) throws Exception {
    //create RestTemplate class object
    RestTemplate template=new RestTemplate();
    //Define service url
    String serviceUrl="http://localhost:3030/SpringBootRestProj13-ProviderApp/actor/wish/{id}/{name}";

    // Generate Http reqeust with GET mode to consume the web service(API)
    //ResponseEntity<String> response=template.getForEntity(serviceUrl, String.class,1001,"raja"); (or)
    ResponseEntity<String> response=template.getForEntity(serviceUrl, String.class,Map.of("name","rajesh","id",1001));

    //display the recieved details from the response
    System.out.println("Response body(output) ::"+response.getBody());
    System.out.println("Response status code value ::"+response.getStatusCodeValue());
    System.out.println("Response headers ::"+response.getHeaders().toString());
    System.out.println("Response satus code ::"+response.getStatusCode().name());


    System.exit(0); //optional .. given for to stop server automatically

}

}
```

Dec 24.1 RestTemplate

Sending JSON Data from Consumer app along with POST mode request using RestTemplate

=> POST mode request contains request body, request headers and initial line
where as GET mode request contains only request headers and initial line.

Every request sturcture contains 2 parts
HEAD part (initial line + request headers)
BODY/Payload part (request body)

HEAD, BODY of request structure will be seperate with blankline

=> Here we need to use postForEntity(..) or postForObject(...) methods of RestTemplate to send post mode request and to get response completelty or partially..

`<T> ResponseEntity<T>`	`postForEntity(String url, Object request, Class<T> responseType, Map<String,?> uriVariables)` Create a new resource by POSTing the given object to the URI template, and returns the response as `HttpEntity`.
`<T> ResponseEntity<T>`	`postForEntity(String url, Object request, Class<T> responseType, Object... uriVariables)` Create a new resource by POSTing the given object to the URI template, and returns the response as `ResponseEntity`.
`<T> ResponseEntity<T>`	`postForEntity(URI url, Object request, Class<T> responseType)` Create a new resource by POSTing the given object to the URL, and returns the response as `ResponseEntity`.

`<T> T`	`postForObject(String url, Object request, Class<T> responseType, Map<String,?> uriVariables)` Create a new resource by POSTing the given object to the URI template, and returns the representation found in the response.
`<T> T`	`postForObject(String url, Object request, Class<T> responseType, Object... uriVariables)` Create a new resource by POSTing the given object to the URI template, and returns the representation found in the response.
`<T> T`	`postForObject(URI url, Object request, Class<T> responseType)` Create a new resource by POSTing the given object to the URL, and returns the representation found in the response.

In server/ Producer/Provider App

Create a model class

```
package ss.it.model;
import lombok.Data;
@Data
public class Actress {
      private Integer aid;
      private String name;
      private Float age;
      private String type;
}
```

RestController

```
@RestController
@RequestMapping("/actor")
public class ActorOperationsController {


    @PostMapping("/register")
    public ResponseEntity<String> registerActor(@RequestBody Actor actor){
          return new ResponseEntity<String>("Actor data"+actor.toString(),HttpStatus.OK);
    }
}
```

In Consumer App/Client App

Runner class

Note:- Setting media type is optional for JSON, Text. only for XML we use it

```
@Component
public class ActorSeviceConsumingRunner_PostingJSON_Data implements CommandLineRunner {

    @Override
    public void run(String... args) throws Exception {
          //create RestTemplate class object
          RestTemplate template=new RestTemplate();
          //Define service url
          String serviceUrl="http://localhost:3030/SpringBootRestProj13-ProviderApp/actor/register";
          //prepare JSON data (request body)
          String json_body="{ \"aid\" : 1001, \"name\": \"suresh\" ,\"age\": 30.0 ,\"type\":\"hero\" }";
          //prepare headers
          HttpHeaders headers= new HttpHeaders();
          headers.setContentType(MediaType.APPLICATION_JSON);
          // prepare Http request as HttpEntity obj having head , body
          HttpEntity<String> request= new HttpEntity<String>(json_body,headers);
          // make http request call in post mode
      ResponseEntity<String> response=template.postForEntity(serviceUrl,request,String.class); //url,request, output type

          //display the recieved details from the response
          System.out.println("Response body(output) ::"+response.getBody());
          System.out.println("Response headers ::"+response.getHeaders().toString());
          System.out.println("Response status code value ::"+response.getStatusCodeValue());
          System.out.println("Response satus code ::"+response.getStatusCode().name());


          System.exit(0); //optional .. given for to stop server automatically

    }
}
```

note:-

plese check the json data before running the application if you type wrongly you may get 400 error

if you getting the problem while perparing json data as request body follow the steps::::

step 1:- put " ";
step 2:- put "{ }";
step 3:- put "{ key : value }";
final step :- put this symbols at the at starting of key and value, ending of key and value

Dec 26 RestTemplate exchange(-) method

Working with RestTemplate

=>Instead of calling getForXxx() ,postForXxx() , put(..),delete() and etc.. as seperate methods to generate different modes of requests .. we can use single exchange(....) for all operations. (one method to generate all modes of requets)

```
public <T> ResponseEntity<T> exchange(String url,
                                      HttpMethod method,
                                      @Nullable
                                      HttpEntity<?> requestEntity,
                                      Class<T> responseType,
                                      Object... uriVariables)
                               throws RestClientException
```

Execute the HTTP method to the given URI template, writing the given request entity to the request, and returns the response as ResponseEntity.

URI Template variables are expanded using the given URI variables, if any.

Specified by:
exchange in interface RestOperations

Parameters:
url - the URL
method - the HTTP method (GET, POST, etc)
requestEntity - the entity (headers and/or body) to write to the request may be null) **(headers+ body)**
responseType - the type to convert the response to, or Void.class for no body **(required response type)**
uriVariables - the variables to expand in the template **(path variables values)**

Returns:
the response as entity

Throws:
RestClientException

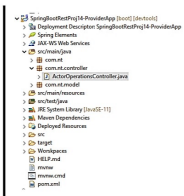
exchnage(.....) is alternate for the following methods

getForEntity(...) , postForEntity(...) , getForObject(...) , postForObject(...), delete(...) ,put(...) , patchForEntity(...) , patchForObject(...) and etc..

Example app

Provider/Producer/Server App

```
@RestController
@RequestMapping("/actor")
public class ActorOperationsController {
      @GetMapping("/wish")
      public ResponseEntity<String> displayWishMessage(){
            return new ResponseEntity<String>("Good Morning",HttpStatus.OK);
      }
      @GetMapping("/wish/{id}/{name}")
      public ResponseEntity<String> displayWishMessage(@PathVariable Integer id,
                                                        @PathVariable String name){
            return new ResponseEntity<String>("Good Morning::"+id+"..."+name,HttpStatus.OK);
      }
      @PostMapping("/register")
      public ResponseEntity<String> registerActor(@RequestBody Actor actor){
            return new ResponseEntity<String>("Actor data"+actor.toString(),HttpStatus.OK);
      }
}
```



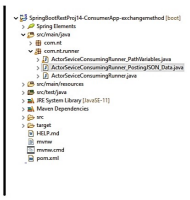
Use Run on server .....

Client App/ consumer App

```
@Component
public class ActorSeviceConsumingRunner implements CommandLineRunner {
@Override
public void run(String... args) throws Exception {
 System.out.println("ActorSeviceConsumingRunner.run()");
 //create RestTemplate class object
 RestTemplate  template=new RestTemplate();
 //Define  service url
  String serviceUrl="http://localhost:3030/SpringBootRestProj14-ProviderApp/actor/wish";

  //  invoke  service method/operation using exchange(-,-,-) method
  ResponseEntity<String>  resp=template.exchange(
                                serviceUrl,
                                HttpMethod.GET,
                                null,  // no  body for GET moode request and
                                       // we do not want to pass  any header values
                                String.class);

//  display the details
System.out.println("response body (output)::"+resp.getBody());
System.out.println("response status code ::"+resp.getStatusCode());
System.out.println("response status code value ::"+resp.getStatusCodeValue());
System.out.println("response header values ::"+resp.getHeaders());


System.out.println("-----------------------------------------");
    }
}
```

```
@Component
public class ActorSeviceConsumingRunner_PathVariables implements CommandLineRunner {
@Override
public void run(String... args) throws Exception {
 //create RestTemplate class object
RestTemplate  template=new RestTemplate();
//Define  service url
 String serviceUrl="http://localhost:3030/SpringBootRestProj14-ProviderApp/actor/wish/{id}/{name}";

 //  invoke  service method/operation using exchange(-,-,-) method
 ResponseEntity<String>  resp=template.exchange(
                               serviceUrl,
                               HttpMethod.GET,
                               null,  // no  body for GET moode request and
                                      // we do not want to pass any header values
                               String.class,  //required type
                               7890, "anitha");  //path or uri variable values

 //  display the details
 System.out.println("response body (output)::"+resp.getBody());
 System.out.println("response status code ::"+resp.getStatusCode());
 System.out.println("response status code value ::"+resp.getStatusCodeValue());
 System.out.println("response header values ::"+resp.getHeaders());

System.out.println("-----------------------------------------");
    }
}
```

```
@Component
public class ActorSeviceConsumingRunner_PostingJSON_Data implements CommandLineRunner {

    @Override
    public void run(String... args) throws Exception {
          //create RestTemplate class object
          RestTemplate  template=new RestTemplate();
          //Define  service url
           String serviceUrl="http://localhost:3030/SpringBootRestProj14-ProviderApp/actor/register";
           //prepare JSON data  (request body)
           String json_body="{ \"aid\" : 1002, \"name\": \"ranveer\" ,\"age\": 30.0 ,\"type\":\"hero\" }";
           //prepare headers
           HttpHeaders headers= new HttpHeaders();
           headers.setContentType(MediaType.APPLICATION_JSON);
           // prepare  Http request as HttpEntity obj having head , body
           HttpEntity<String> entity= new HttpEntity<String>(json_body,headers);  //headers + body
           // make http request call  in post mode
     ResponseEntity<String>  response=template.exchange(serviceUrl,
                                                        HttpMethod.POST,
                                                        entity,
                                                        String.class) ;

          //display the recieved details from the response
          System.out.println("Response body(output) ::"+response.getBody());
          System.out.println("Response headers ::"+response.getHeaders().toString());
          System.out.println("Response status code value ::"+response.getStatusCodeValue());
          System.out.println("Response satus code ::"+response.getStatusCode().name());

          System.out.println("-----------------------------------------");
    }
}
```



note: Instead of creating Multiple objects for RestTemplate class in different runner classes of Cosumer App .. create RestTemplate class obj in main class using @Bean method and inject the spring bean obj in runner classes with the support @Autowired annotation.

Recieving and processing diferent return type values to webService methods/operations using RestTemplate

=> If web service method /operation return type is ResponseEntity<Strinng> /String then we get plain text as the response..

=> If web service method /operation return type is othe than ResponseEntity<String> /String then we get JSON response as text content.. by default .. this JSON text content can be used directly or can be converted to Java object using JACKSON API (built -in spring boot Rest Applications)

=>Once we add spring MVC starter to the Project ... we get JACSON API automatically we can be used to JSON data to Object (DeSerilization) and Object to JSON data (Serialization)

=> Irrespective of the return value of web service method/operation (String or non-String) we generally take the return value of exchnage(...) or getForXxx(...) or postForXxx(...) and etc.. methods as String based return value like ResponseEntity<String> object.. i.e recieve every thing from provider App as String content and convert it to other types if needed.

=>The ObjectMapper class of Jackson api gives methods for Serilziation and DeSerilizatation

a) mapper.readValue(-) :: JSON text to Object (DeSerialization)
b) mapper.writeValue(-) :: Object to JSON text (Serialization)

Sever App/proividerApp/Producer App

```
@RestController
@RequestMapping("/actor")
public class ActorOperationsController {
      @GetMapping("/find")
      public  ResponseEntity<Actor>  searchActor( ){
            return new ResponseEntity<Actor>(new Actor(200,"salman",56.0f,"hero"),  //body
                                             HttpStatus.OK);  // status code

      }
}
```

Client App/Consumer App

```
@Component
public class ActorSeviceConsuming_GettingJSONData_Runner implements CommandLineRunner {
      @Override
      public void run(String... args) throws Exception {
            System.out.println("ActorSeviceConsuming_GettingJSONData_Runner.run()");
            //create RestTemplate class object
            RestTemplate  template=new RestTemplate();
            //Define  service url
             String serviceUrl="http://localhost:3030/SpringBootRestProj14-ProviderApp/actor/find";

             // invoke  service method/operation using exchange(-,-,-) method
             ResponseEntity<String>  resp=template.exchange(serviceUrl,
                                                            HttpMethod.GET,
                                                            null,  // no  body for GET moode request and  we do not want to pass
                                                                   //any header values
                                                            String.class);
             //  display the details
             System.out.println("response body (json output)::"+resp.getBody());
             System.out.println("response status code ::"+resp.getStatusCode());
             System.out.println("response status code value ::"+resp.getStatusCodeValue());
             System.out.println("response header values ::"+resp.getHeaders());
             System.out.println("-----------------------------------------");
             //converting JSON text response(body) to  Java class object /Model class obj/Entity class object using JACKSON api
              String jsonBody=resp.getBody();
              //create ObjectMapper
              ObjectMapper mapper=new ObjectMapper();
              Actor actor=mapper.readValue(jsonBody,  // body
                                           Actor.class);  //requred object type
              System.out.println("response body as the Actor object data:: "+actor);

      }//method
}//class
```

Dec 28 RestTemplate all methods

Converting List of objects Data (1D Array Data) of JSON given By Server/Producer/Provider App into List of Objects in Conumer App using Jackson api

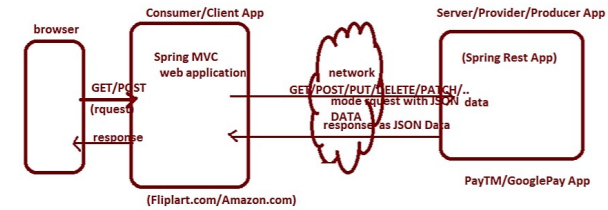
In server /Producer/Provider App

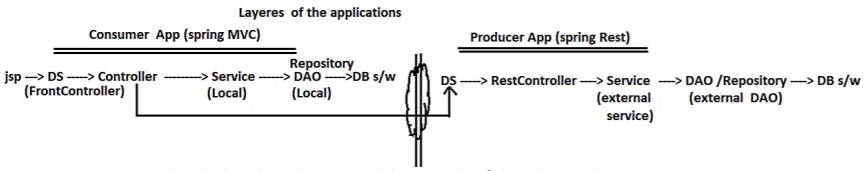
```
@RestController
@RequestMapping("/actor")
public class ActorOperationsController {

   @GetMapping("/findAll")
        public  ResponseEntity<List<Actor>>  fetchAllActors(){
              return new  ResponseEntity<List<Actor>>(List.of(new Actor(101,"salman",55.0f,"hero"),
                                                              new Actor(102,"rajesh",65.0f,"hero"),
                                                              new Actor(103,"ranveer",35.0f,"hero")),
                                                             HttpStatus.OK);
        }
 }
```

In Consumer/Client App

```
@Component
public class ActorSeviceConsuming_GettingJSONData_Runner implements CommandLineRunner {

@Override
public void run(String... args) throws Exception {
 System.out.println("ActorSeviceConsuming_GettingJSONData_Runner.run()");
 //create RestTemplate class object
 RestTemplate  template=new RestTemplate();

 //Define  service url
 String    serviceUrl="http://localhost:3030/SpringBootRestProj14-ProviderApp/actor/findAll";
 //  invoke  service method/operation using exchange(-,-,-) method
       ResponseEntity<String>  resp1=template.exchange(serviceUrl,
                                    HttpMethod.GET,
                                    null,  // no  body for GET moode request and  we do not want to pass
                                           //any header values
                                    String.class);

//  display the details
System.out.println("response body (json output)::"+resp1.getBody());
System.out.println("response status code ::"+resp1.getStatusCode());
System.out.println("response status code value ::"+resp1.getStatusCodeValue());
System.out.println("response header values ::"+resp1.getHeaders());
//converting JSON text response(body) to  Java class object /Model class obj/Entity class object using JACKSON api
 String jsonBody1=resp1.getBody();
 //create ObjectMapper
 ObjectMapper mapper1=new ObjectMapper();
 Actor[] actors=mapper1.readValue(jsonBody1,   // body
                                  Actor[].class);  //required object type
 List<Actor> listActors=Arrays.asList(actors);
 System.out.println("response body as the List<Actor> object's data:: "+listActors);
 System.out.println("-----------------------------");
 listActors.forEach(System.out::println);
 System.out.println("-----------------------------");
 List<Actor> listActors1=mapper1.readValue(jsonBody1,new TypeReference<List<Actor>>() {});
 System.out.println("-----------------------------");
 listActors1.forEach(System.out::println);

    }//method
}//class
```

TypeReference is jackson api supplied
abstract class is used for obtaining full generics type information by sub-classing;





The Distributed comps are also called External comp/External services becoz they will not part of Consumer App/Project .. They always reside outside the of consumer Project/App giving services to Consumer App.

what is the difference b/w put(-) and exchange(-,-,-,-) of RestTemplate?

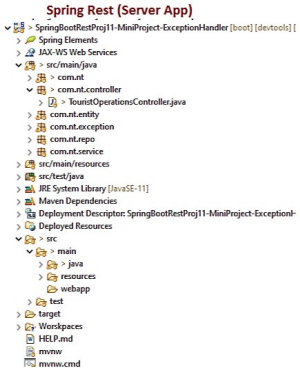
Ans) =>put(-) mehthod can send only PUT mode request producer App where as exchange(...) can send different modes of requests

=>put(-) mehthod return type is void i.e we can not get response body given by producer App where as exchange(...) return type is ResponseEntity<T> i.e we can get result/response given by provider App

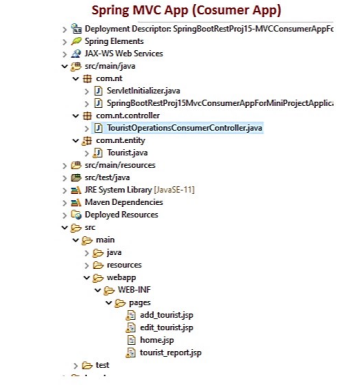
what is the difference b/w delete :(-) and exchange(-,-,-,-) of RestTemplate?

Ans) Similar to above

Spring Rest (Server App)



Spring MVC App (Cosumer App)



Dec 31 Spring Cloud-Need of MicroServices

Spring Cloud (working with MicroServices)
==========

Need of MicroServices
==========

Monolithic Apps Vs SOA Apps Vs MicrorSErvices Apps

In spring Boot the Micro service archicture Apps will be implemented with support of spring Cloud Module and multiple other facilities

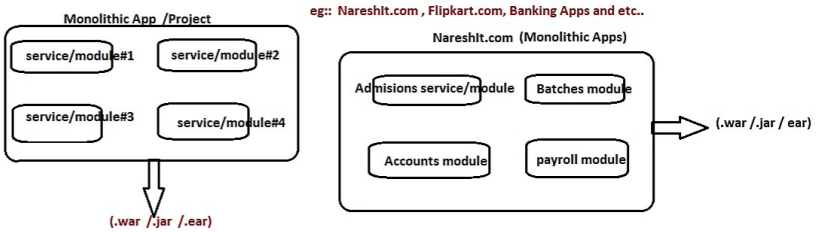
=> Generally we do not prefer using spring framework for MicroService implemenations .. we prefer spring boot a lot in the angel of spring boot cloud to implement MicroServices..

Monolithic Apps
==========

=>The Application the packs multiple services as multiple modules in a single unit (nothing but project) either as war file or jar file or ear file is called Monolithic Application/Project.

1 application = 1 Project contains multiple services as muitiple module inside the project/Application

=>So far the spring MVC Apps, spring data jpa Apps , spring core Apps and etc.. we developed are called Monolithic Apps
=> Even spring MVC with sprign data JPA /spring ORM/spirng JDBC that we have developed so far (except Spring Rest Apps) are called Monolithic Apps..



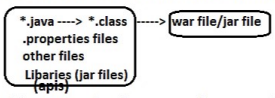
.war ==>web application
.jar ===> standalone App (Non-web applicaiton) / standalone web application (Using Embedded Server)

ear file = jar file + war file +..... (or)
ear file => jar file + jar file +jar file .... (or)
ear file => war file +war file +war file

ear file :: enterprise App archieve (now ear files not popular becoz web services comps also coming as war files .. When EJB was there ear files bit popluar to represent web applicaiton(war) + ejb comp (jar) togather as ear file

Build Process /Building the Application
==========

=>The process keeping the app ready for execution in certain env.. is called build process (or operations)



=> In buid process , developmentn of source code , compilaation of source code , development helper resources like properties files and other files , adding api libaries and etc.. will be there .. At last the war file or jar file will be created reprsenting the whole project.

What is the difference App /Service or API?

=>API --> application Programming Interface .. It is base for programmer to devleop certain language or technology or framework Apps. In Java api comes in the form of pkgs having classes,interfaces, enums and annotations.. These apis will be released /available in the form jar files (libaries/dependencies)

3 types of APIs :: a) pre-defifned apis (given by technology/language/framework vendor)
b) user-defined apis (given by developer)
c) third party apis (given by third party vendor)

=>Application/Service is the outcome of build process either in the form of jar file or war file having certain to task to perform.

note:: In the development App /Services in certain language/technology /framework we take the support APIs.

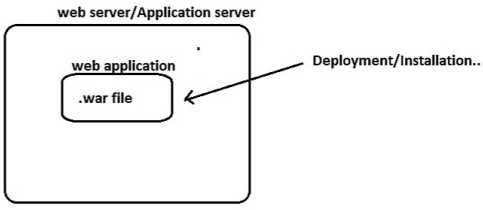
note:: The existing APIS/libraries can be used to create new APIs/Libaries or projects/Apps/sevices

eg: we can use hibernate api directly to develop hibernate persinstece logic /hibernate App
the spring creators have used the same hibernate api to develop spring ORM apis and spring data jpa apis

=>Jar file purpose will change context to context..
jar repsents apis
jar represents jdbc driver
jar represents standalone web application
jar represents ejb comp
jar represents web application (war file)
jar represents standalone App/project
and etc...

Deployment /Installation
==========

=>The process of keeping web application (.war file) in the server (either that is running or later started) is called deployment..



note:: In the devleopment /Testing env.. deployment takes place in Local servers or s/w companies cloud account (aws/ gcp/azure and etc..)
note: In the UAT and Production the deployment takes place in the Client Org Servers or Client org's Cloud Account

How build Process takes place in Realtime companies?

Ans) using Maven/ gradle Tool In combination with Jenkins CI/CD configration

CI :: Continous intergration
CD :: Continous Deployment

How do we pack app/service/project in to single unit having entire env.. for execution?

=>Only coding packing -----> war /jar file
=> Envirormment packing ---> docker image (code(war/jar) + server + DB + OS + ...)

Dec 31.1 Spring Cloud-Need of MicroServices

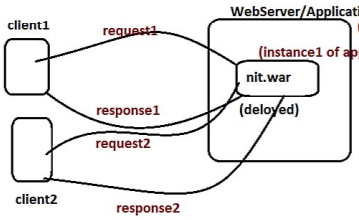
Service Instance / App Instance /instance

=>A runnig Application inside the server giving services to Clients (either for endusers or for other Apps) is callled Service Instance /instance /App instance..

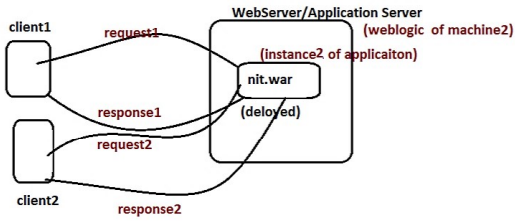
=> Each copy of app(jar or war) that is deployed in server successfully to render services for clients is called is called service Instance..

App/Project ---> is like class
each deployed App/Project giving services to clients is like instance
note:: One class can have multiple instances.. simpliarly one app/project can deployed in multiple servers i.e can have multiple instances/Service instances

nareshIT web application (App /Project) => (complted devleopment/testing)
(nit.war)



we can deploy the
the multiple instances /sevice instances of
one App/Project either in same server or
in different servers of same machine or different meachiens



In Every server we can specify max no.f requests that each service can allow , In most of the server this max request count is 200 by default

In spring boot MVC or spring Rest Apps we can change /control this max request count using the following properties of application.properties

In application.properties
---

server.tomcat.threads.max= 150 (default is 200)
server.iettv.threads.max= 300 (default is 200)

Load Count /Current Load

=> The no.of requests that are currently under proccessing by service instance is called Load Count /Current Load.
=> if the App1 instance/service instance is currently processing 10 requests then the Load count /Current Load is 10.

Load Factor

=> it is current Load/Max Load

=> Load factor = courrent Load/ max Load

=> if App instance with respect server max load is 150 and that app instance is processing only 100 request currently then 100/150 is the Load Factor (0.666)

Load Factory is always >=0 to <=1

maxLoad = max requests that server can take for each instance of the Application at a time
we can specify this using server settings or using application.properties while working with embedded servers of spring boot.

Scaling

=> Increasing the service capacity of App/SErvice /Project is called Scaling..

=> if the App is having facility to increase its service capacity as needed then it called scalable App.. (scalability feture)

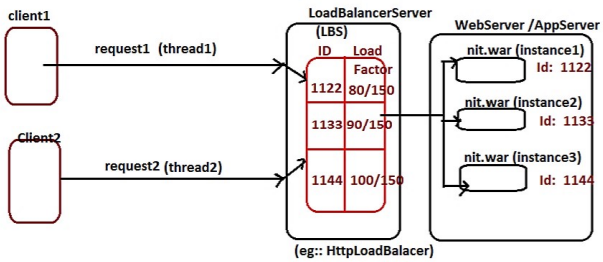
Examples :: eg1:: increasing shop capacity from 1000 square feet to 10000 square feet (vertical scaling)
eg2:: keeping same shop of 1000square feet in 10 places. (Horizontal scaling)
eg3:: increasing no.of ATM machies in different locations (Horizontal scaling)

Two types of Scaling

a) Horizontal Scaling (good)
b) Vertical Scaling

a) Horizontal Scaling

In Horizontal scaling we take the multiple copies same App i.e multiple service Instances to given services to Clients.. and we control these multiple instances with the support of Load Balancer Service (LBS)



These instances
can be there either
in same server different domains
or different copies of same server or different servers
residing in the same machine or different machines,

=> The given request goes to that instance/service instance of the App whose LoadFactor is less (whose value is nearer 0) .. In our example the both req1,req2 will go to instance beoz it is having less load factor (80/150). if all the instances are having same load factor then the LBS will pick up the instance randomly..

Examples for different LoadBalancer

- HAProxy – A TCP load balancer.
- NGINX – A http load balancer with SSL termination support. ...
- mod_athena – Apache based http load balancer.
- Varnish – A reverse proxy based load balancer.
- Balance – Open source TCP load balancer.
- LVS – Linux virtual server offering layer 4 load balancing.

and etc..

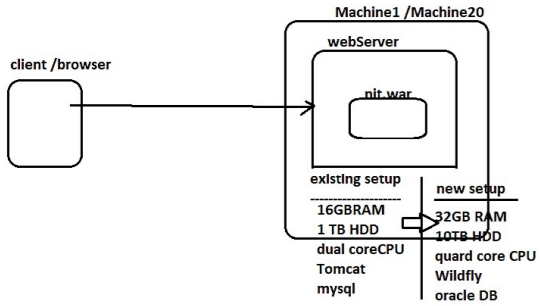
Keeping gmail App in different zones of world map falls under horizontal scaling.

Either DevOPs team or Infrastucture team like Linux admin and etc.. will take care of this kind Load Balancer..

Jan 01 Spring Cloud-Need of MicroServices

b) Vertical Scaling

=> Here we add more software or hardware infrastructure for the existing App env..
to make it ready for taking more requests from more clients.



In Cloud env.. like AWS
we get max of 720 GB RAM
and unlimited HDD suport
can be taken..

Pros and cons Monolithic architecture based Applicaiton Development
==========

pros (advantages)
==========

=> Provides simple and easy env.. to develop Projects/Apps as the Layered Apps having layers like presnetation layer , controller layer , service Layer , Persistence Layer(DAO), Integration layer and etc..

note: Most of the current maintainance projects are Monolithic projects
=> Easy to deploy (all togather single jar/war file) , easy to manage (single war/jar file) , easy to scale (increasting or decresing instances)
=> Performing unit Testing is very easy becoz all layers and all services are avaiable togaher
=> Less possiblity of getting network related issues becoz all services /layers mostly there togather.
=> Performing logging and dubegging operations is very easy.. (Even Adutiing activities are easy)

cons (DisAvatages)
====

=> For small or tiny changes in one or two services /modules /layers we need redeploy the application by alterting the endusers when App in the production. (App will not for these many hours) work

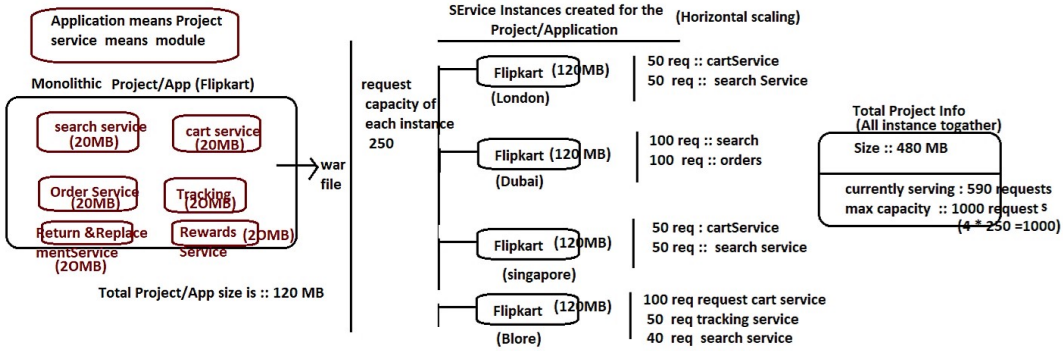
=> one bug or issue in one module/service of Project/App may effect other services /modules ..this indicates these Apps are not reliable. (Sometimes Apps will shutdown providing no services to clients)

=> Adding new technologies /concepts/ frameworks in the existing as part of enhacement is very complex sometime we will be forced to redesign the project..

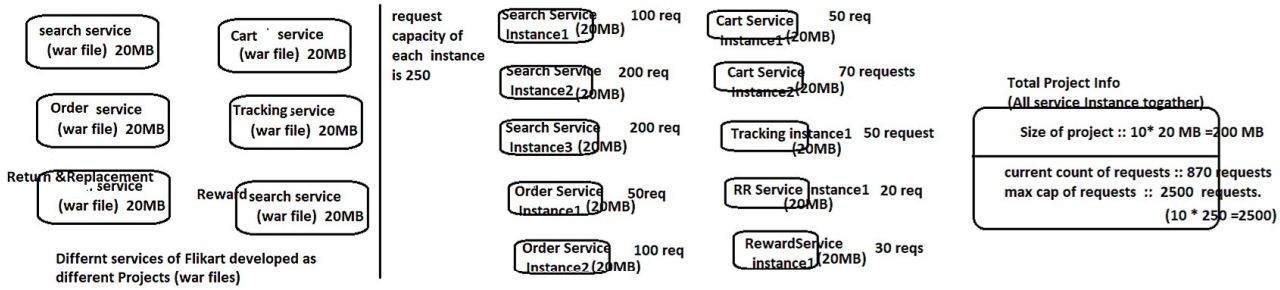
=> if no.of modules/services are incresed in the Project then their maintainace towards bug fixing and redeployment also takes lots of time while working with websErver and application server .. (This may cause increasing App/Project downtime)

=> we can not sell certain services /modules of one Project to clients as they need...

=> Since we can create service instances only for entire App .. not for certain services/modules of the Project lots of memory and CPU time wastage will be there.
and etc..



Let us assume we are able to create seperate project/App for every service/Module (Horizontal scaling by taking modules/sericces as the Projects/Apps)
==========



=>This kind of Application development in Monolithic architecture is possible but very complex to implement and manage .. To overcome this problem we have got MicroServices Archicture..

SOAP based web service fall under SOA
Restfull webservices fall under Producer- Consumer mechanism

note:: MicrorServices Archicture is enhacment Producer-consumer mechanism where we develop each microservice as restfull App

What is the link between webServices and MicroServices?
Microservices are extension of webServices (Restfull webservices) In fact each micro service will be developed as one Restfull webservice by adding other facilities.

Jan 02 Spring Cloud-Need of MicroServices

Monothilic vs SOA vs MicroServices archicture

SOA (Service Oriented Architecture)

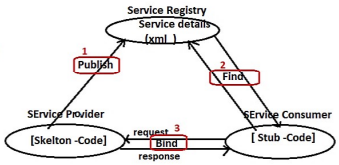
=> This architecture given to get communication between two applications that are developed in two different technologies and running from two different machines using Register and Discovery Concepts..

=>SOAP based webServices (like Jax-ws, axis ,apche cfx and etc..), CGI-Links are given as the implmenentation models of SOA Design..

=>SOA is design that provides certain archicture with principles to get communication between two incompitable Apps..

3 comps SOA

a) Service Provider
b) Service Consumer
c) Service Registry



1.Service Provide develops the prodiver App and exposes it details to serviceRegistry in the form of xml docs (Publish Operation)

2. TheService consumer contacts the ServiceRegistry and get the details about provider or service (Find Operation)

3. The service consumer prepare stub code to consumer services of the Provider using request-response model (Bind Operation)

Advatntages of SOA

1) Interoperability :: we can develop the Provier and consumer Apps either in same technology or in different technologies

2) Easy Maintainace :: Any change in provider App , we just need to update to Service registry and need not inform to all consumers..becoz Service cosumer can collect the changes from SErvice Registry

3) Quality Code is guaranteed :: Since we can develop provider App in our choice technology So we can given quality provider App..

4) Scalable :: we can add more parellel servers or instanstaces for provider App as service consumers and their no.of requests are incresing

4) realiable :: Since the SOA machanism is involving the Service Registry .. we can maintain our Distributed App in reliable and stable state..

DisAdvantages ::

HighCost :: It needs lots of man power and technology /infrastrucure support for developing SOA applications

Xml utilization :: Supports only XML based Data exchange which is fading out day to day

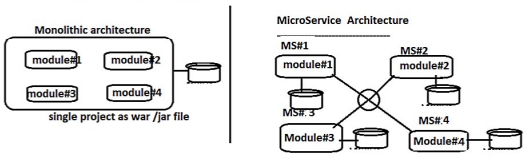
HighBandwidth of internet required :: Since service registry ,Service Providers will be placed in different locations we definelty need good bandwith of internet connections.

MicroServices Architecture (Extesion Restufull web service model)

=>In this architecture every service /module will be developed as seperate project/App having connection with other service/module..

=> It is Decoupled Archicture of a single Project becoz multiple services/modules will be devleoped as multiple projects and they will be registered/Integrated in a common place.

=> Micro SErvice is small service or small application
micro = small
service = project/ Application..



Every Micro service will be developed as Restfull App /RestController and will be place in common registry/server to make it avaiable for other micro services or Client Apps.

Advatnatages of MicroSErvice Architecture

=> Need not stop other services /modules (MicroServices) if problem comes in certain module/service [if Rewards or Feebback sevrice failed it does not effect other services like sales , orders, cart and etc..]

=> Need not stop all other the services .. while updating /patching certain service for betterment

=> We can use different Technologies to develop different services of the Project/application..

=> We can do horizontal scaling for each service as needed i.e if certain service like searching, cart and etc.. are having more demand then we can create more instances for them by enabling Load Balancing

=> Updgrading/Migrating to new tehchnologies in each service is no complex .. Quite easy compare to monolithin architecture

=> Service Down time (while performing reloading or restart or redployment activities) less becoz we need to perform these actitvies on one service at time.

=> Allows to place common things of multiple Services (MicorServices) in a common place like GIT hub env.. instead of placing in every service/ module

note:: Developing MicroServices using spring boot is very easy becoz most of the common things like DB setup , Schema initializations and etc.. can be cfg to occur through auto Configuration of spring boot

Limitations of MicroServices

=>Maintainance of the Project is very complex becoz it generates multiple log files , contains multiple communication chains and debugging the code across multiple services is too difficult..

=> Needs high end intrastruture to maintain multiple micro services , if going for Cloud we need to purchase /rent muliple things to deploy and manage these projects.

=> Knowlege on More Tools , technlogies and processes is required to work with MicroServices Projects

=> Migrating Monothic Project into MicroServices project is very complex

=> Testing is bit difficult if one microservice is having dependency with other micro service.

note:: we generally uses Cloud env.. to deploy and execute the modren monolithic , SOA and Micro services projects.. Working with with Cloud is nothinb but taking things on Rental basis and using them for our requirements..
eg:: AWS , Azure , Goolge clould, PCF (Pivotal Cloud Foundary) , RackSpace and etc..

Clould env.. (AWS/Azure ...)

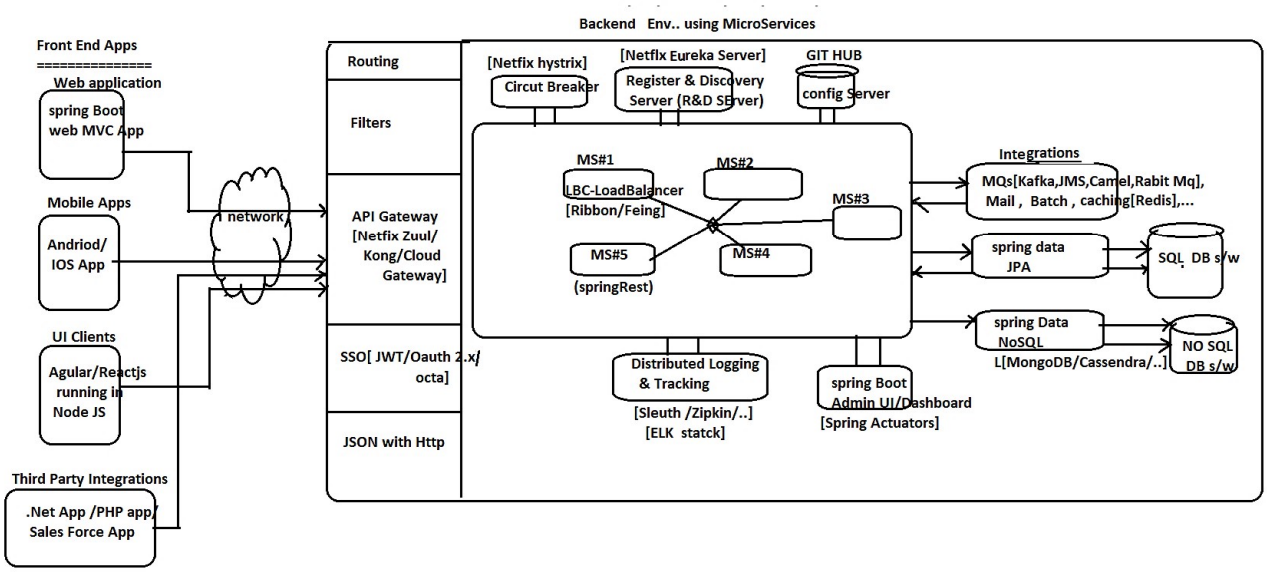
IAAS	PAAS	SAAS
RAM	Linux	=>web server/App servers
HDD	windows	=>DB s/w
network	Ubentu	=>Mail servers
CPU	MAC	=JDBC drivers
RACKs	Device Drivers	=>MQ Services
...		=>Jdk s/w
		=> Jenkins
		=> Docker
		....

Cloud services
a) PAAS :: Platform As A service
b) IAAS :: Infrastrutur As A service
c) SAAS : Software As A service

=> MicroServices architeture is Design and whose Apps/Projects can be implemented/ developed in spring env.. with the support of spring boot + spring clould + tools

Jan 05 MicroServices overview

Overview of MicroService Archicteture



Micro service Architecture is Desinging or specification which provides set of rules and guidelines to develop Project as set of loosely coupled /De coupled Services .. and this can implemented using spring boot +spring cloud + Netflix and lots of other tools.

=> Every module in the project will be develop as seperate microservices in the form of spring RestController with the support of spring boot env..

=> Once the Microservic architecture project is ready .. It will be deployed in cloud env..like AWS/AZure/Google Cloud and etc.. with DevOPs tools Jenkins with CI/CD + Docker + Ansible + Kubernate +....

=>After developing each Micro SErvice as seperate Spring Rest App/Project it must be published in a common place called R & D SErver (Register And Discovery Server) like Netflix Eureka server

=> One MicroSErvice can find another MicroService in R & D SErver and can be used for Communication Through the same old R & D Server..

=> One MicroService can find and communicate with another microservice only when it is published in R & D server ..

=> The common properties with same values of Multiple Micro services can be placed outside the Micro services in place called Config server like GIT Hub

=> if any exception is raised in the execution of one microService then it has to be informed to Admin UI /Dashboard with the support of Circuit Breker like Netflix hystrix.

=> if any MicroService is having more demand then we allows to create multiple instances dynamically.. In that situation to pick up right istance with less Load factor from other MicroServices we take the support of Load Balancer Clients (LBC) like Ribbon ,Feign ,Http LoadBalancer and etc..

=> since we are devleoping every MicrService as Spring Rest App in spring boot env.. So we can make these Micro services Apps integrating with lots of other facilities like MQs (Message Queries), Mail, Caching, Batch PRocessing and etc..

=> MicroServices can interact with SQL Db s/ws using spring data jpa

=> MicroServices can interact with No SQL Db s/ws using spring data Nosql modules like spring data mongoDB ,spring data cassendra and etc..

=> To Monitor and Manage all the MircorSErvices of the Project .. we try to spring boot Admin UI/Dashboard that is created using support of spring boot Actuators which are also useful providign non-funcational features like Health metrics on the projects..

=> Since Project contains multiple microSErvices interacting with each other..So we need perform logging and tracing activity across the multiple micro services as needed with the support Distributed Logging and tracking tools like slueth and zipkin

=> The MicroServices of the Project can have different types of Clients (Front end Apps) mobile Apps , web mvc Apps, Third party Apps , UI Technologies App and etc..

=> To use all these microServices and tools from different types of Clients /Front end Apps we need one common entry and exit point concept nothing but API Gateway like netflix zuul/Kong and etc.. providing facilities to apply Filters , Routers , SEcurity like SSO (Single Sign On) and etc..

Jan 06 Eureka Server

Spring Cloud - Netflix Eureka Server

=>Every MicroService must be registered or published with R & D (Register and Disocovery) server in order to make it discoverable/communicatable from other MicroServices

=> As of now netflix eureka server (best), apache zookeeper are two popular R & D Servers.

=> Every R & D server is spring boot Project having R &D Server dependencies (Do not except seperate installation like Tomcat, wildfly and etc..)

=> The Process of keeping /publishing MicroService details in R &D Server is called registration activity

=> The process of discovering/fetching MicroService details from R & D Server to esstablish communication/interaction from another MicroService is called Discovery Operation.

=> When we publish different instances of different micro services to R & D server like Eureka server .. then the details will look like this...

Service Id	InstanceId	HostName/IPAddrs	LoadFactor [CurrentLoad/ MaxLoad]
Search-Service	SS:566-a367	192.6.8.7 7878	0/200
Search-Service	SS:536-a461	192.6.8.7 7171 (IPaddrs / port number)	0/200
Order-Service	OS: 455-a356	192.6.8.7 8989	0/200
Order-Service	OS: 415-a256	192.6.8.7 8182	0/200

=>Eureka Server is no documents server .. i.e it does not contain any xml files or Json files inside the server .

=> Service Id is always Project name (Service Id is called as Service Name)

=> Instance id is the unique Id .. For every Instance one unique instance Id will be generated..

=>Providing isntance Id when single instance is there optional .. In that situation it takes ServiceId as the Instance Id

=> The HostName and port details will be auto detected by Eureka server durigng the process Register/publishment

=> LoadFactor = current Load /Max Load.

Procedure to create Spring Boot project acting as Eureka server

==========

step1) create Spring Boot Project adding Eureka Server as depedencies

step2) add @EnableEurekaServer on the top of main class

```
@SpringBootApplication
@EnableEurekaServer
public class SpringBootMsProj01EurekaServerApplication {

	public static void main(String[] args) {
		SpringApplication.run(SpringBootMsProj01EurekaServerApplication.class, args);
	}

}
```

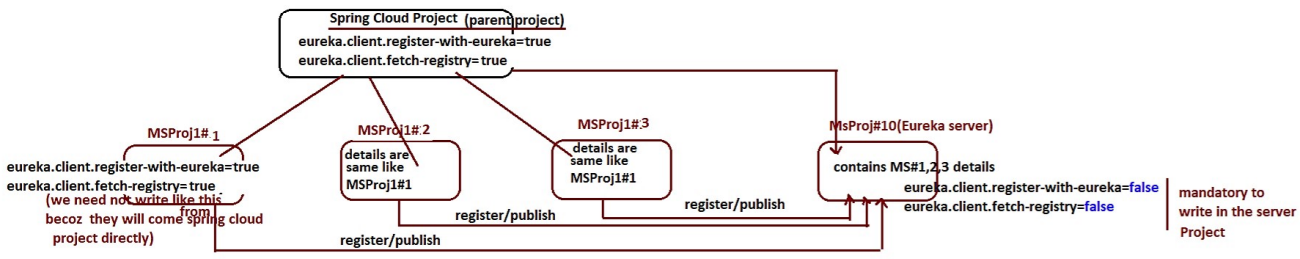
step3) Add the following entries in application.properties

To disable the process of making this project as MicroService and to present this project as R & D Server

appplication.properties

```
server.port=8900
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
```

=>spring boot project we get Eureka server as the Embedded Server..



=> Every Ms Project should kept ready to register with Eureka Server .. Since we can not register Eureke server Project itself with Eureka server , So we need to make the default value "true" Inherited from the spring cloud project for "eureka.client.register-with-eureka" property as "false" in Eureka server Project.

```
eureka.client.register-with-eureka=false
```

=>Every MS project should be kept ready for discovery/fetching for communincation while registering with Eureka server .. Since Eureka Server Project can not kept ready for fetching/discovery so we need make the default "true" inhreited from the spring cloud project for "eureka.client.fetch-registry" as "false" in Eureka server Project

```
eureka.client.fetch-registry=false
```

The default port number for eureka server is :: 8761

step4) Run the application...

Right click on Project -->run as --> java app/spring boot app

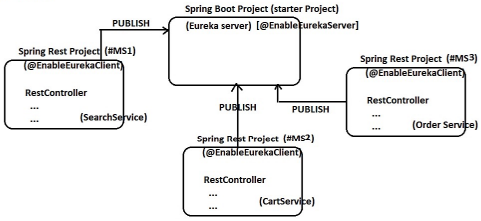
step5) Test the SErver App

http://localhost:8900/

Jan 07 Publishing MS to Eureka Server

Publishing MicroService to Eureka server (R & D Server)

=> Every Ms must be published/registered with Eureka Server (R&D Server) by becoming Eureka client
=> we need to develop MicroService as Spring RestController adding the the support of @EnableEurekaClient



Procedure for MS Development and Publishing

step1) Make sure One Spring boot Project alreay developed and running as Eureka Server (Previous class) *(make sure that it is running the default port : 8761)*

step2) Create Spring Boot Starter Project adding spring web, EurekaClient Dependencies (MicroService Development)
Discovery

step3) Place @EnableEurekaClient Annotation on top of main class.

```
@SpringBootApplication
@EnableEurekaClient
public class SpringBootMsProj01SearchServiceApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringBootMsProj01SearchServiceApplication.class, args);
    }

}
```

step4) add the following entries in application.properties file
in applicaiton.properties

```
# MS service port number
server.port=7171

# Service Id
spring.application.name=Search-Service

# Specify the Eureka server URL to publish the MS
eureka.client.service-url.default-zone=http://localhost:8761/eureka
```

fixed path for publishing

step5) Develop RestController representing the MicroService

```
package com.nt.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/search")
public class SearchServiceController {

    @GetMapping("/display")
    public String displayMessage() {
        return "Welcome to Flipkart Search Service";
    }
}
```

step6) Run the MicroService Project as spring boot App
(This process automatically publishes MS to Eureka server)

step7) Refresh the home page of eureka server (http://localhost:8761) and observe the instance section

Instances currently registered with Eureka

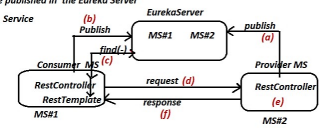
Application	AMIs	Availability Zones	Status
SEARCH-SERVICE	n/a (1)	(1)	UP (1) - DESKTOP-IUDAAVL:Search-Service:7171

step8) Collect url from status section and modify it to generate the request to MS

http://desktop-iudaavl:7171/search/display
(or)
http://localhost:7171/search/display

Inter communication between MicroServices

=> To see communication b/w two micro services .. both micro services must be published in the Eureka Server
=> The MS that provides services is called Provider/Server /Producer /Parent Service
=> The MS that cosumes services is called Cosumer/Client /Child Service



CartService <----------> PaymentService
(consumer/client) (Procudure/Server)

Employee <----------> Department
(consumer/Client) (Procudure/Server)

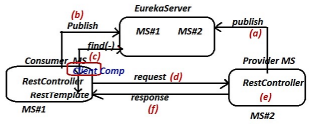
CartService <----------> Billing Service
(consumer/Client) (Procudure/Server)

Generally the the Cosumer and Producer Apps will be represented as show below
(parent and child)

Payment service --------◇ cart service
(parent) (child)

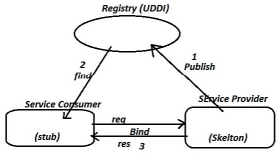
Doctor ◇---------- Patient
(parent) (child)

MicroService style intra communication



Here both Consumer and Provider services will be published to Eureka server before starting interaction

SOA style Intra communication

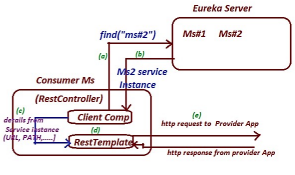


Here only provider Service will be published.

=>In the intra communication of MicroService through Eureka Server the Consumer /Client/ Child MS should find and get the details of Provider/Producer/Server MS by submitting its service Id .. For this the Cosumer MS must use one special comp "Client Comp/ Client Type comp".

=>The work of "Client comp" in Consumer Ms is
a) getting Producer MS service Instance from Eureka SErver by submitting its Service Id
b) Gathers Producer MS details like URL/URI ,method type, PATH and etc.. from Service Instance
c) Passing the above details to RestTemplate of Consumer MS to make RestTemplate to send http request to Producer Ms

As of now In Eureka server env.. 3 types of "Client Comps/Client type comps" are possible
a) Discovery Client Comp (Very Basic Client -- Legacy)
b) LoadBalance Client comp (good)
c) Feign Client Comp (Abstract Client -- Good)



jan 09 Ms-Intra Communication

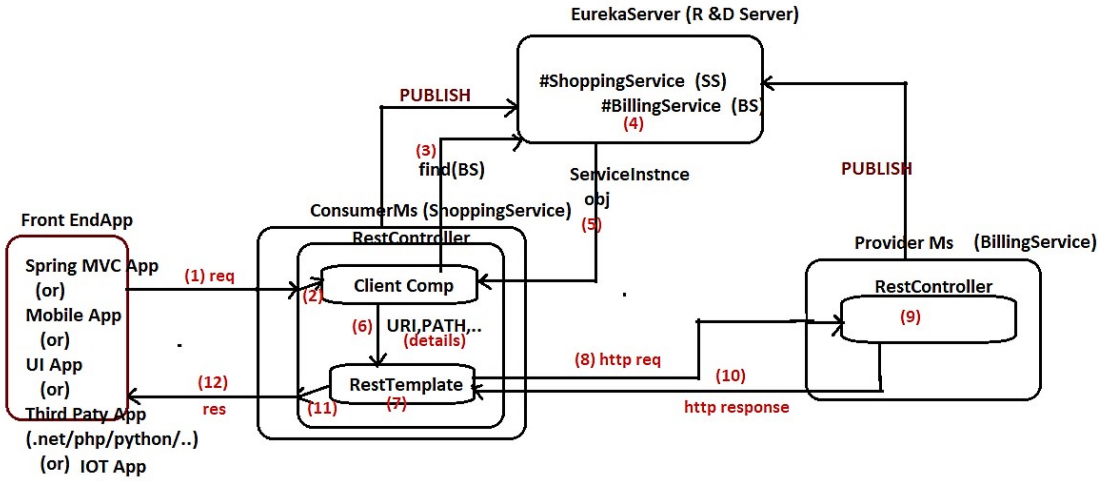
MicroServices Intra Communication
==========

=> We can perform intra communication between micro services by publishing them in the Eureka Server ...

=> The Cusumer Ms must use one or another "Client Comp" or "Client Type comp" to get multiple details about Provider MS by passing service Id .. These multiple details are like URI, PATH, method type,path variables and etc.. will be passed RestTemplate obj to make Http Call to connect with Provider Ms

=> The Consumer MS can work with 3 types of "Client Comps" or "Client Type comps"

> DiscoveryClient (Legacy)
> LoadBalancerClient (for Load balacing) (good)
> FeignClient (Abstract Client) (good)



Example App development (as POC) ==========

(Using DiscoveryClient as the Client Comp)

step1) Develop Project as Eureka Server

=>dependencies :: Eureka service
=> add @EnableEurekaServer on the top of main class
=> application.properties

```
server.port=8761
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
```

step2) Develop Project as Producer MicroService

=>dependencies :: web , EurekaDiscoveryClient ,devtools

=> add @EnableEurekaclient on the top of main class

=> application.properties

```
#Ms Properties
#Port number
server.port=9900
#Service id
spring.application.name=Billing-Service
#Eureka server publishing info
eureka.client.service-url.default-zone=http://localhost:8761/eureka
```

=> Develop RestController having producer b.methods

```
//RestController (provider)
package com.nt.controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
@RestController
@RequestMapping("/billing")
public class BillingServiceController {

    @GetMapping("/info")
    public String getBillingInfo() {
        return "we accept Card Payment , UPI Payment, NetBaking Payment, COD";
    }
}
```

step3) Develop the Cosumer MicroService (Shopping Service)

=>dependencies :: web , EurekaDiscoveryClient ,devtools
=> add @EnableEurekaclient on the top of main class

=>application.properties

```
#Ms Properties
#Port number
server.port=6600
#Service id
spring.application.name=Shopping-Service
#Eureka server publishing info
eureka.client.service-url.default-zone=http://localhost:8761/eureka
```

-> Develop spring bean (@Componet) as helper class having logic to use DiscoveryClient as the ClientComp to find and get Producer/provider Ms Instance and Details

```
http://localhost:9900   (URI)
http://localhost:9900/billing/info (URL)

        URL=URI +PATH
```

jan 09.1 Ms-Intra Communication

```
//RestConsumer
==============
package com.nt.client;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.cloud.client.ServiceInstance;
import org.springframework.cloud.client.discovery.DiscoveryClient;
import org.springframework.stereotype.Component;
import org.springframework.web.client.RestTemplate;
@Component
public class BillingServiceRestConsumer {
      @Autowired
      private  DiscoveryClient  client;  //comes through  AutoConfiguration
                                 //client Comp

      public  String  getBillingInfo() {
            //find and get SErviceInstances  of Producer by using Service Id
            List<ServiceInstance>  listSI=client.getInstances("Billing-Service");
            // use  first ServiceInstance from the List of Instances
            ServiceInstance  SI=listSI.get(0);
            //get Producer MS URI  and make it  as URL
            String url=SI.getUri()+"/billing/info";
            // create RestTemplate object
            RestTemplate  template=new RestTemplate();
            // invoke producer MS service method or operation by generating Http call
            String  resp=template.getForObject(url,String.class);
             return resp;
      }//method
}//class
```

note:: All producer and consumer Ms must be annotated with @EnableEurekaClient

=> Develop RestController in consumer Application by taking the support of above helper class..

```
  //Rest Controller
  package com.nt.controller;
  import java.util.function.Consumer;
  import org.springframework.beans.factory.annotation.Autowired;
  import org.springframework.web.bind.annotation.GetMapping;
  import org.springframework.web.bind.annotation.RequestMapping;
  import org.springframework.web.bind.annotation.RestController;
  import com.nt.client.BillingServiceRestConsumer;
  @RestController
  @RequestMapping("/shopping")
  public class ShoppingController {
        @Autowired
        private BillingServiceRestConsumer consumer;

        @GetMapping("/info")
        public  String   displayShoppingDetails() {
              return  "Pongal Shopping for Family ...."+consumer.getBillingInfo();

        }                    One MS  RestController is  using another MS RestController
  }
```

step4) Execute the Projects/Apps in the following order

8761 -server
9900 -producer
6600 -consumer

=>run the Eureka Server
=>Run the Produder App (BillingService)
=> Run the Cosumer App (ShoppingService)
=> Go to Eureka Server Console to modify Consumer Service Url for testing..

original url ::
http://desktop-iudaavl:6600/actuator/info
modified url :: http://desktop-iudaavl:6600/shopping/info

Pongal Shopping for Family ....we accept Card Payment , UPI Payment, NetBaking Payment, COD

Limitations of DiscoveryClient type ClientComp
==========

(a) We get list of target MS instances and we need to instance manually .. But actually want
one instance of target MS(Producer Ms) which having less Load Factor (Load balacing is not possible)

(b) This Baisc Client Comp or Client Type which very much Legacy .. i.e it is not indurstry standard..

(c) Collecting one instance from the list of instance is pure responsibity of Cosumer App that to manual process.. So other instancces of target/produce Ms may sit idle..

note:: To overcome these problems take the support of LoadBalancerClient type Client Comp.

LoadBalancerClient
==========

=> It is another Client type Comp or Client Comp .. which choose the less load factor instance though they are multiple instances for Target /Procuder Ms i.e we never get List of instances though they are muliple instances for MS.. we always get one instance of MS which is having Less LoadFactor

=>LoadBalnaceClient is an interface and implementation is given by Spring Clould NetFlix peopled in the form of RibbonLoadBalanceClient class which can be injected to Cosumer App through AutoConfiguration..

=> The method choose(-) with instance Id called on the LoadBalanceClient obj will bring the Less LoadFactor Instance of target /producer Ms.

note:: To create multiple instances for any MS run that Ms App for multiple times with different Port numbers becoz Two Servers or Two Ms of same computer can not take same port number .. but possible across the multiple machines.

jan 09.2Ms-Intra Communication.png

Example App using LoadBalanceClient Type Client Comp (MS intra communication using LoadBalanceClient)

step1) Develop Eureka Server App same as previous App

step2) Develop Producer/Provider/Target MS (This time provide random number as the instance Id)

=>Since we are planning take multiple instances of producer App by running the Producer App for multiple times it is better to give seperate name for every instance generally it is serviceId:<randomvalue> using

```
eureka.instance.instance-id=${spring.application.name}:${random.value}
```

App name as the ServiceId — System property giving one psuedo random number for every execution

(do not add dev tools)

=> create project having sprigng web , CurekaDisoveryClient, dependencies

=> add @EnableEurekaClient on the top main class

=> add the following entries in application.properties

application.properties

```
#Ms Properties
#Port number
server.port=9900
#Service id
spring.application.name=Billing-Service
#Eureka server publishing info
eureka.client.service-url.default-zone=http://localhost:8761/eureka
# provid applicaiton + radom vlaue as Instance Id
eureka.instance.instance-id=${spring.application.name}:${random.value}
```

LoadBalancerClient

=> It is another Client type Comp or Client Comp .. which choose the less load factor instance though they are multiple instances for Target /Procuder Ms i.e we never get List of instances though they are muliple instances for MS.. we always get one instance of MS which is having Less LoadFactor

=>LoadBalnaceClient is an interface and implementation is given by Spring Clould NetFlix peopled in the form of RibbonLoadBalanceClient class which can be injected to Cosumer App through AutoConfiguration..

=> The method choose(-) with instance Id called on the LoadBalanceClient obj will bring the Less LoadFactor Instance of target /producer Ms.

note:: To create multiple instances for any MS run that Ms App for multiple times with different Port numbers becoz Two Servers or Two Ms of same computer can not take same port number .. but possible across the multiple machines.

Example App using LoadBalanceClient Type Client Comp (MS intra communication using LoadBalanceClient)

step1) Develop Eureka Server App same as previous App

step2) Develop Producer/Provider/Target MS

(This time provide random number as the instance Id)

=>Since we are planning take multiple instances of producer App by running the Producer App for multiple times it is better to give seperate name for every instance generally it is serviceId:<randomvalue> using

```
eureka.instance.instance-id=${spring.application.name}:${random.value}
```

App name as the ServiceId — System property giving one psuedo random number for every execution

(do not add dev tools)

=> create project having sprigng web , CurekaDisoveryClient, dependencies

=> add @EnableEurekaClient on the top main class

=> add the following entries in application.properties

application.properties

```
#Ms Properties
#Port number
server.port=9900
#Service id
spring.application.name=Billing-Service
#Eureka server publishing info
eureka.client.service-url.default-zone=http://localhost:8761/eureka
# provid applicaiton + radom vlaue as Instance Id
eureka.instance.instance-id=${spring.application.name}:${random.value}
```

=> Develop Restcontroller actiging Producer MS

```
package com.nt.controller;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/billing")
public class BillingServiceController {
 @Value("${server.port}")
 private int port;
 @Value("${eureka.instance.instance-id}")
 private String instanceId;

@GetMapping("/info")
public String getBillingInfo() {
      return "we accept Card Payment , UPI Payment, NetBaking Payment, COD---> port::" +port+"----InstanceId::"+instanceId;
}
}
```

Injected to know which instance is picked by LoadBalanceClient to cosume the Producer services (purely optional)

jan 09.3 Ms-Intra Communication

step3) Devlop the Consumer App with support of LoadBalanceClient

(do not add dev tools)

=> create project having sprigng web , CurekaDisoveryClien dependencies (Ribbon comes automatically)
=> add @EnableEurekaClient on the top main class
=> add the following entries in application.properties

```
#Ms Properties
#Port number
server.port=6600
#Service id
spring.application.name=Shopping-Service
#Eureka server publishing info
eureka.client.service-url.default-zone=http://localhost:8761/eureka
```

=> develop helper having LoadBalanceClient comp Injection

```
package com.nt.client;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.cloud.client.ServiceInstance;
import org.springframework.cloud.client.loadbalancer.LoadBalancerClient;
import org.springframework.stereotype.Component;
import org.springframework.web.client.RestTemplate;

@Component
public class BillingServiceRestConsumer {

    @Autowired
    private LoadBalancerClient client; //comes through AutoConfiguration

    public String getBillingInfo() {
        //get Less LoadFatory Service Insntance
        ServiceInstance SI=client.choose("Billing-Service");
        //get Producer MS URI and make it as URL
        String url=SI.getUri()+"/billing/info";
        // create RestTemplate object
        RestTemplate template=new RestTemplate();
        // invoke producer MS service method or operation by generating Http call
        String resp=template.getForObject(url,String.class);
        return resp;
    }//method
}//class
```

=>Develoe the RestController

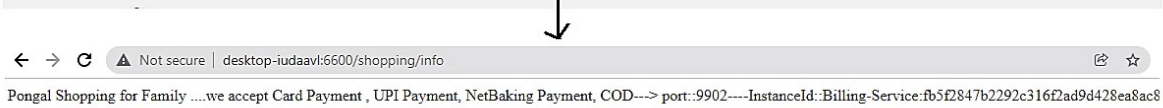
```
package com.nt.controller;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.nt.client.BillingServiceRestConsumer;

@RestController
@RequestMapping("/shopping")
public class ShoppingController {
    @Autowired
    private BillingServiceRestConsumer consumer;

    @GetMapping("/info")
    public String displayShoppingDetails() {
        return "Pongal Shopping for Family ...."+consumer.getBillingInfo();
    }
}
```

step4) run Apps in the following order

=>run Eureka server App
=> Run Producer Apps multiple times but change port number in application.properties each time (at least 2 times)
=> Run the Cosumer App

=>Go to Eureka server Consle modify the Consumer App url

Application	AMIs	Availability Zones	Status
BILLING-SERVICE	n/a (2)	(2)	UP (2) - Billing-Service:cc78dd2418223af081691cb6054a459f , Billing-Service:8fa7b22275cccff553b00ffe7d335591
SHOPPING-SERVICE	n/a (1)	(1)	UP (1) - DESKTOP-IUDAAVL:Shopping-Service:6600



Pongal Shopping for Family ....we accept Card Payment , UPI Payment, NetBaking Payment, COD---> port::9902----InstanceId::Billing-Service:fb5f2847b2292c316f2ad9d428ea8ac8

Jan 10 Ms-Intra Communication-Feign Clent

MicroServices Intra Communication

==========

Limitation with Discovery Client , LoadBalancerClient

=> They can find and get producer MS ServiceInstance and other details from Eureka Server .. But they can not give http calls to interact with target/Producer MS For that we need to use RestTemplate seperately

=> we need to coding manually to find and get Target MS Service Instance

To overcome this problems use Feign Client as Client Comp/Client Type comp

Feign Client /Open Feign

==========

=> It is called abstract client becoz we just provide interface with method declaration and adding annotation .. But entire logic will be generated in the InMemory dynamic proxy class (Proxy pattern)

=>It is Combination Client i.e it takes getting taget/producer Ms service Instance from Eureka server and also takes care of interacting target/Producer Ms by generating http calls that to with out wrting code.

=> It is internally LoadBalancer Client i.e it gets Target/Producer Ms service Instance from the List of seerviceInstances which having Less Load Factor.

=> This mode of Client Comp development improves the productivity of the App.

=> Here we do not develop helper RestConsumer for Consumer RestController rather we develop Interface having @FeignClient("ServiceId") and the generated InMeory DyamicProxy class object will be injected to ConsumerRestController.

=> While worokign with FeignClient we need to add 2 annotation special annotations along with regualar annotations in the Consumer App

a) @EnableFeignClients on the top main class along with @EnableEurekaClient

b) @FeingClient on the top of interface for which dynamic Inmemory proxy class will be generated.

must match with taget/producer MS service Id

```
@FeingClient("Billing-Service")
public interface IBillingServiceRestConsumer{
@GetMapping("/billing/info")          // target/producer MS method /operation path
public String getBillingInfo();
}
```

singnature should mattch with target Ms method

method name need not to match taret MS method name

Example App Using Feign Client

==========

step1) Develop Project as Eureka Server

=>dependencies :: Eureka service of

=> add @EnableEurekaServer on the top main class

=> application.properties

```
server.port=8761
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
```

step2) Develop Project as Producer MicroService

=>dependencies :: web , EurekaDiscoveryClient ,devtools

=> add @EnableEurekaclient on the top of main class

=> application.properties

```
#Ms Properties
#Port number
server.port=9900
#Service id
spring.application.name=Billing-Service
#Eureka server publishing info
eureka.client.service-url.default-zone=http://localhost:8761/eureka
```

=> Develop RestController having producer methods

```
package com.nt.controller;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/billing")
public class BillingServiceController {
 @Value("${server.port}")
 private int port;
 @Value("${eureka.instance.instance-id}")
 private String instanceId;

 @GetMapping("/info")
 public String getBillingInfo() {
       return "we accept Card Payment , UPI Payment, NetBaking Payment, COD---> port::"+port+"----InstanceId::"+instanceId;


   }
 }
```

Jan 10.1 Ms-Intra Communication-Feign Clent

step3) Develop Cosumer MS Project adding spring web , EurekaDiscoveryClient ,open Feign

=> Add @EnableEurekaClient , @enableFeignClients annotations on top of main class

```
@SpringBootApplication
@EnableEurekaClient
@EnableFeignClients
public class SpringBootMsProj04ShoppingSErviceConsumerApplication {

public static void main(String[] args) {
        SpringApplication.run(SpringBootMsProj04ShoppingSErviceConsumerApplication.class, args);
}
}
```

=> add the following entries in application.properties

```
#Ms Properties
#Port number
server.port=6600
#Service id
spring.application.name=Shopping-Service
#Eureka server publishing info
eureka.client.service-url.default-zone=http://localhost:8761/eureka
```

=> Take an interface supporting FeignClient code as InMemory Dynamic Proxy class

```
package com.nt.client;

import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;

@FeignClient("Billing-Service")
public interface IBillingServiceRestConsumer {

    @GetMapping("/billing/info")
    public String fetchBillDetails();

}
```

=>Develop the Cosumer RestrController Injecting FeignCleint related Proxy object to consume the target /Producer Ms services.

```
//RestController
package com.nt.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.nt.client.IBillingServiceRestConsumer;

@RestController
@RequestMapping("/shopping")
public class ShoppingController {

@Autowired
private IBillingServiceRestConsumer consumer;

@GetMapping("/info")
public String displayShoppingDetails() {
System.out.println("ShoppingController:: client comp class name::"+consumer.getClass());
return "Pongal Shopping for Family ...."+consumer.fetchBillDetails();

}

}
```

step5:
Execute the application
==========

=>Run Eureka server app
=> Run the Producer MS application for 2 or 3 times with different port numbers changed in the application.properties file
=> Run the Cosumer Ms application
=> Go to Eureka Server Home page and modify the Cosumer SErvice url to as shown below

Pongal Shopping for Family ....we accept Card Payment , UPI Payment, NetBaking Payment, COD---> port::9902----InstanceId::Billing-Service:43c6cbdef49aebccda0ab868294114f4

Different pratices to develop Feign Client Interfaces
==========

Producer Ms / Target Ms
==========

a) ServiceId /applicaiton -name:: Vendor-Service

```
@RestController
@RequestMapping("/vendor")
public class VendorServiceController{
   @GetMapping("/all")
   public ResponseEntity<List<Product>>
                        getAllProducts(){
      ...
   }
}
```

Feign Client Interface at Cosumer MS
==========

a)

```
@FeginClient("Vendor-Service")
public interface IVendorServiceConsuner{
    @GetMapping("/vendor/all")
    public List<Product> fetchAllProducts();
          (or)

    @GetMapping("/vendor/all")
    public ResponseEntity<List<Product>> fetchAllProducts();

}
```

---

b) Service Id /app name :: Payment-Service

```
@RestController
@RequstMapping("/payment")
public class PaymentServiceController{

  @PostMapping("/save")
  public String saveCard(@RequestBody CarDetails details){
    ...
    ..
  }

  @DeleteMapping("/delete/{cardNo}")
  public String removeCard
          (@PathVariable Integer cardNo){
    ...
    ..
  }
```

```
@FeignClient("Payment-Service")
public interface IPaymentServiceRestConsumer{

  @PostMapping("/payment/save")
  public String addCard(@RequestBody CardDetails details);

  @DeleteMapping("/payment/delete/{cardNo}")
  public String deleteCard( Integer cardNo) ;

}
```

Assuming the front end app of Consumer MS sig giving JSON inputs

Assume Front End app is not giving json data.. to this Cosumer MS

Jan 17 Ms-Intra Communication (Feign Clent)

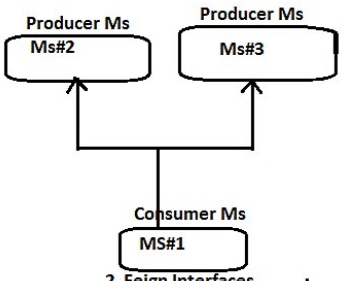
In MicroServices intra communication we need Client Type comps to find the SErvice Instance Details of Target Ms and to perform Http calls based communucation with taget MS/Producer MS from Consumer Ms.

Client Type comps are
a) Discovery Client
b) LoadBalacerClient (LBC)
c) Fiegn Client

=> if we have 1 producer - 1 consumer type of MicroServices using Feing client then we need to take 1 Feign Interface (@FeignClient) at Cosumer Ms side... if the the Conumer Ms is consuming the multiple Producer Ms services then we need to take multiple feign Interfaces.

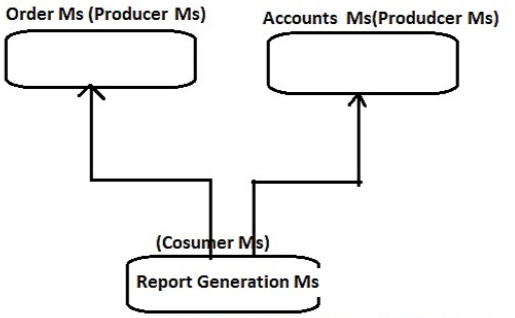
Feign Interface count = no.of producer Ms services consumed by Cosumer Ms

=> For example if MS#1 is consuming Ms#2 and Ms#3 producer services then Ms#1 consumer Ms should have 2 feign Interfaces as shown below



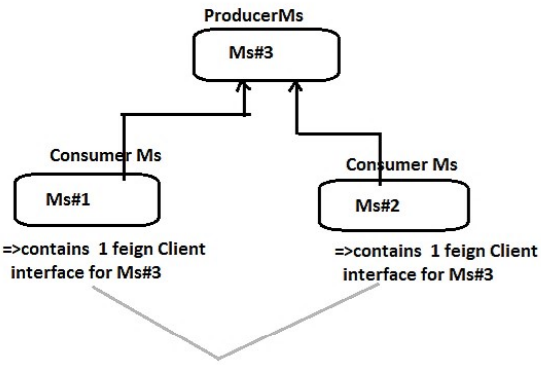
2 Feign Interfaces
=> 1 for Ms#2 service
=> 1 for Ms#3 service

In this Cosumer Ms we need to place 2 @FeignClient("....") annotations on two different feign interfaces having the two different service ids of two different producer Ms. But we need to place @EnableFeignCleints only for 1 time on Main class.



=>Contains 1 Feign Interface for Order Ms
=>Contains 1 Feign Interface for Accounts Ms

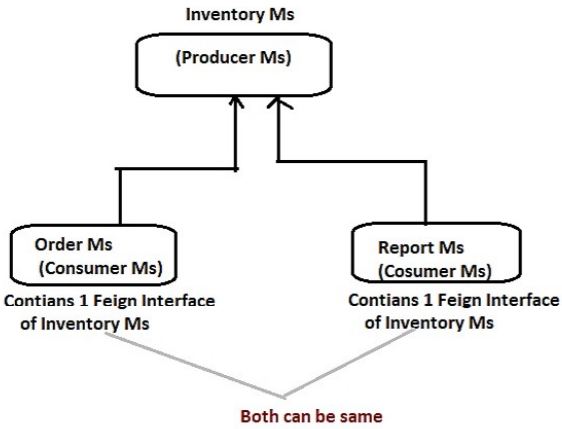
=> if one Producer Ms services are used by multiple consumer Ms then we may need to place same feign Client Interface in multiple consumer Ms.



=>contains 1 feign Client interface for Ms#3

=>contains 1 feign Client interface for Ms#3

(Both may be same interfaes)



Contians 1 Feign Interface of Inventory Ms

Contians 1 Feign Interface of Inventory Ms

Both can be same

Summary on ClientType comps

Client Type comp	support for Load Balancing	Required Annotation	Is Abstract Client or Concrete Client	industry standard or not	required dependnecy	operations
DiscoveryClient	no	@EnableEurekaClient (main class)	concrete client	not	Discovery Cleint	=>call getinstances(-) to get SService Instance and use RestTemplate to make http calls
Load Balancer Client	yes	@EnableEurekaClient (main class)	concrete client	not	Discovery Client	=>same as above but method is choose(-)
FeignCleint	yes	@EnableEurekaClient, @EnableFeignClents (main class) @FeignClient(---) (interface level)	abstract client (Intenrally InMemory proxy will be generated)	yes	Open Feign	=>The Inmemory Proxy class for @FeignClient interface will take of getting service Instance and making http calls

Spring Cloud Config Server
==========

jan 19 Spring Cloud Config SErver

=> It is also called Configuration server(CS) and this is useful to get common key=value pairs required for the multiple micro services from the common place.. i.e instead of placing same key=value pairs in multiple micro services .. we can place them in common place and we can get them through configuration server for multiple micro services

=> These common key=value pairs are generally DB conneection properties , email properties , security properties and etc.. which are required as same properties in multiple in MicroServices..

=> We place these common key=value pairs seperate properties file outside of all MicroServices Projects and we take seperate project for Configuiration Server having one application.properties and this file will be linked with that common/seperate properties file.

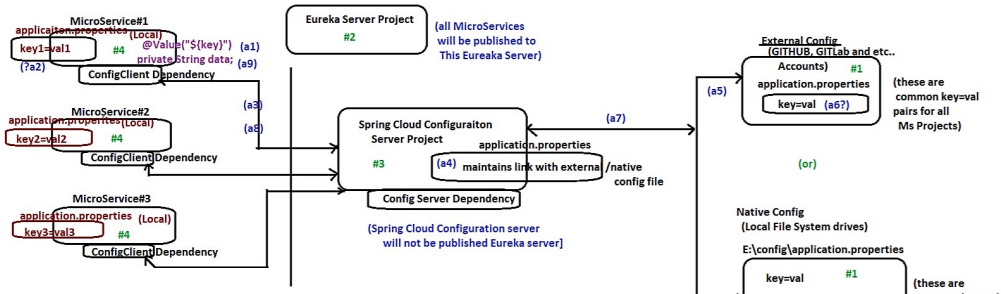
=> WE need to create Configuration server Project(Also a Ms Project) adding "ConfigurationServer" dependendency and this Configuration server by default runs on the Port number 8888.

=> We can make Configuration server Project getting common key=value pairs in two ways

(best) a) Using External Server Configuration (i.e we can place common properties file in GIT accounts like github, gitlab(best), bitbucket,..)
(Good for all env.. dev, test, uat, prod)

b) Using Native Server Configuration (i.e we can place common properties file in Local File System Drives like E:,D: drives and etc..
(Generally used in dev env.. bit)

=> The real micro service projects that want to use Configuration server managed common properties file content (key=value pairs) must be added with "Configuration Client " dependency.



==========

Example Application on using GitLab Account maintain External Config file through spring colud Config server

==========

step1) create application.proeprties (External config file) in Git Lab account

-> go gitlab.com
-> register/singup account , verify through email address

-> singin/Login to git lab account by submitting username,password that were created above

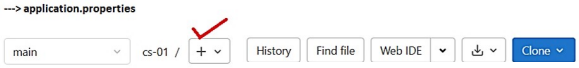
-> create new Project giving Project name
Menu bar ---> Projects ---> create new Project ----> Blank Project-->
project name :: CsProj1 --->select public ---> create project.

->add applicaiton.properties file in that Project
Go to home page CsProj1 ---> + ---> new file ---> application.properties

add --> key=value pairs

application.properties
---
dbuser=system
dbpwd=manager

commit .. chanages

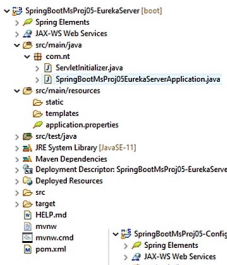


-> Gather GitLab account Project url
Go to CsProj1 home page ---> clone ---> gather url ::
https://gitlab.com/nataraz/csproj1.git
protocol domain gitlab username project name

step2) create Eureka server Project in Eclipse IDE
(add :: Eureka server as dependency ,spring web)

--> place @EnableEurekaServer on the top of main class
--> add the following entries in application.properties

```
# server port
server.port=8761

#disable registration and fetching
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
```



step3) create Spring Cloud Configuration server Project (ConfingSeever Project)
(add Config server Dependency ,spring web)
(select from spring cloud config section)

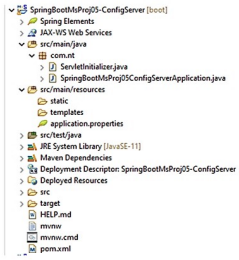
->add @EnableConfigServer on main class
-> add the following entries in application.properties file

```
# server port
server.port=8888
```

default server port no

```
# Provide Link to GitLib user account
spring.cloud.config.server.git.uri=https://gitlab.com/nataraz/csproj1.git
```

Link Url



step4) create Mutliple MicroService Projects having the following dependencies
a) spring web b) Discovery Client , c) Config Client (new)
(select from spring Cloud Config section)

Eureka

Ms#1

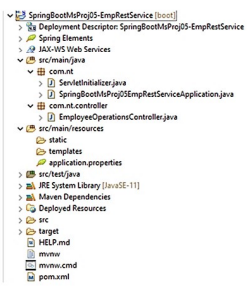
-> add @EnableEurekaClient on the main class
-> add the following entries in application.properites file

```
# server port (MS Port)
server.port=9900
# service name or applicaiton name
spring.application.name=EMP-SERVICE
#provide Eureka server Url to register EUreka server
eureka.client.service-url.default-zone=http://localhost:8761/eureka

#to make MS connecting to 8888 port no configurationServer(required from sprig boot 2.4 onwards)
spring.config.import=optional:configserver:
```

->Develop one RestConntroller conusuming the vlaues ot extenal config file (GibLab account application.properties file)

//controller class

```
package com.nt.controller;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/emp")
public class EmployeeOperationsController {

    @Value("${dbuser}")
    private String user;
    @Value("${dbpwd}")
    private String pass;

   @GetMapping("/show")
   public String showData() {
        return "Data Collected throgh Config Server ::"+user+"----"+pass;
    }
  }
```



jan 19.1 Spring Cloud Config SErver

#Ms2 application.properties

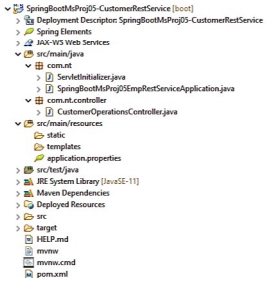
same as MS#1

add dependencies:-
a) spring web
b)config client
c)Eureka discovery client

(+) add @EnableEurekaClient on main class

```
# server port (MS Port)
server.port=9901

# service name or applicaiton name
spring.application.name=CUST-SERVICE

#provide  Eureka server Url to register  EUreka server
eureka.client.service-url.default-zone=http://localhost:8761/eureka

# To  make  Ms Connecting to 8888 port number ConfigurationSErver (required from spring boot 2.4 onwords)
spring.config.import=optional:configserver:
```

-> add @EnableEurekaClient on the main class




```
//CustomerOperationsController.java
package com.nt.controller;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/cust")
public class CustomerOperationsController {

    @Value("${dbuser}")
    private  String  user;
    @Value("${dbpwd}")
    private String pass;

@GetMapping("/display")
public  String   displayData() {
        return  "(Customer)Data Collected throgh Config Server ::"+user+"----"+pass;
}

}
```

We can use GITHUB , BitBucket also
External Configuration for Configuration
server using the same process.
note:: while working with GIT hub
there is no need of passing .git
in the link url of application.properties

step5) Run the Applications in the following order

->Run Eureka server Project
-> Run Config SErver Project
-> Run All Ms Projects

-> Go to EurekaSever Home page and modify the URL both
Emp, Cust Ms services

http://desktop-iudaavl:9901/cust/display ---> To generate request Cust Ms

http://desktop-iudaavl:9900/emp/show ---> To generate request Employee Ms

---

Making Multiple MicroSErvices getting common data from Native Config file (Local system drivers) With the support spring Cloud Configuration server

---

=> It is suitable only in Dev , Test env.. but not in UAT , production env..
=>Use this in dev, test env.. if u r not ready with GIT Accounts
=> Generally we place this Native Configuration related application.properties file
in the spring Cloud Configuration project itself (which also uses Local system Drives)

Example App
==========

step1) Develop Eureka Server App

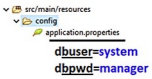
(same as previous App)

step2) Develop Confiauration SErver

(add Config server Dependency)
(select from spring cloud config section)

->add @EnableConfigServer on main class

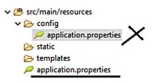
-> add new applicaiton.properties by creating a folder like "config" in src/main/resources folder
to keep common key=value pairs required for all Ms projects.



dbuser=system
dbpwd=manager

src/main/java , src/main/resources folders
of maven project will be placed in classpath
by default.

->add the following entries in application.properties file of src/main/resources folder




```
# server port
server.port=8888
#active native profile of its Parent Project
spring.profiles.active=native
# specify  "config"  folder of  classpath as NativeConfig Location for  Configuration Server
spring.cloud.config.server.native.search-locations=classpath:/config
```

Native Cofig Location

The Parent project of Configuration server is having profiles .. the default
profile is designed to get Linked with Git Accounts i.e taking Extenal Configuration
where as "native" profile is designed to get linked with Native Config (local drives)
So we are activating native profile.

step3) Develop Multiple MicroServices (MS#1 and MS#2)
(same as previous)
add dependencies===>

step4) Execute the Applications/Projects in the following order.

->Run Eureka server Project
-> Run Config SErver Project
-> Run All Ms Projects

-> Go to EurekaSever Home page and modify the URL both
Emp, Cust Ms services

http://desktop-iudaavl:9901/cust/display ---> To generate request Cust Ms

http://desktop-iudaavl:9900/emp/show ---> To generate request Employee Ms

Q:- how ConfigServer gets data of MS Projects properties file in the eureka server ..??

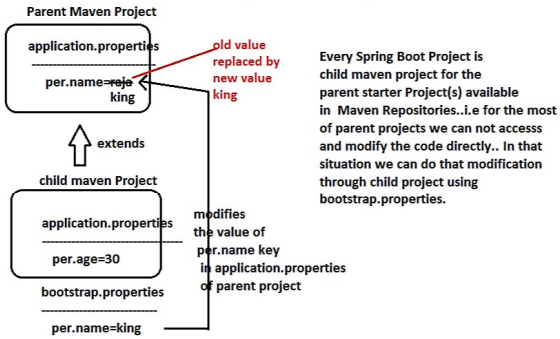
configServer is capable of getting the datails nothing but properties ke value pairs from Repository of external configServer project & we create that as public mode we can access & get details & can use in the required MS project

bootstarp.properties

jan 20 Spring Cloud Config SErver with Bootstrap Properties

if u want change the value of certain key belonging to parent project being from child project then the child project can use bootstrap.properties placed in the classpath like in src/main/resoruces folder.

=>Being from Child Maven Project.. if u r looking to change the parent Project's application.properties file values then use bootstrap.properties in the child project.



Q) What is default port number of spring Clould ConfigServer
ans) 8888

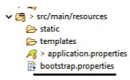
Q) What is spring Clould ConfigServer url/uri
ans) http://localhost:8888

Q) Can we change the Port the number of Config Server?
Ans) yes .. use the application.properties file of Config server project
server.port=8811
accordingly config server uri will change http://localhost:8811

Q) How can we connect to ConfigServer from Ms App if the Config Server is not running on the default port number?

step1) make sure Config SErver Port number is changed(server.port=8811)

step2) add bootstrap.properties in src/main/resources folder of Ms Project having config server uri as shown below.



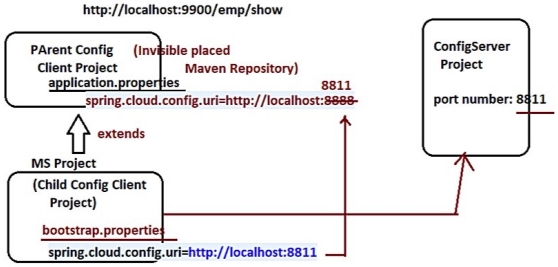
spring.cloud.config.uri=http://localhost:8811

step3) make sure that spring-cloud-starter-bootstrap dependency jar file is added to build path in the MS Project

```
<!-- https://mvnrepository.com/artifact/org.springframework.cloud/spring-
 cloud-starter-bootstrap -->
 <dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-starter-bootstrap</artifactId>
 </dependency>
```

(collect from mvnrepository.com)

step4) Run the PRojects
=> Run EurekaServer
=> Run ConfigServer
=> Run Ms Project → after running this in log messages you'll see some message like `Fetching config from server at : http://localhost:8811`
=> Test Ms Project
http://localhost:9900/emp/show



=> While executing any Ms Project the bootstrap.proeprties will execute first and modifies parent project settings

---

Q) How can we connect to ConfigServer from Ms App if the Config Server is not running on the default port number?
(How can we do this work with out bootstrap.properties)

Ans) Possible by adding following entries in the application.properties file of MS Project reflecting the Changed port number /uri of Config Server.

application.properties

```
# To make Ms Connecting to certain port number ConfigurationSErver (required from spring boot 2.4 onwords)
spring.config.import=optional:configserver:http://localhost:8811
```

note:: Here no need of adding bootstarp.properties
adding spring-cloud-starter-bootstrap dependency → is mandatory or not is in confusion state

---

In Ms Project if we specify two different uris with two different port numbers of Config Server using both application.properties and bootstrap.properties then which will be taken?

Ans) Both uris will be taken but the application.properites file supplied uri will override the uri of bootstrap.properties file

Conclusion:-

Old Approach:- creating bootstrap.properties file and adding the line
`spring.cloud.config.uri=http://localhost:8866`

Modern Approach:- adding the line in already available properties file
`spring.config.import=optional:configserver:http://localhost:8811`

Jan 22 RefreshScope and Actuactors

Working with RefreshScope using @RefreshScope Annotation

=>The modifications GitLab /GITHUB Extenal Config file content will reflect to all the MicroServcies olnly when we restart the Config serrver and all Ms .. But Restrating Configserver and all Ms every time for each modification done in External Config is a not recomanded process.

=>To overcome that problem we can use RefreshScope (@RefreshScope) with support of spring boot Actuators (readymade endpoints /ready made Microservices given by spring boot)

Procedure to work @RefreshScope

step1) spring boot actuator dependency to our MS Project (EmpRestSErvice kind of Project) (Config Client)

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

right click on project ---> properties ---> spring ---> add starter ---> search for actuator and select spring boot actuator.

step2) Activate the readymade actuator "refresh" in our MS Project (Config Client) adding the entries in application.properties

In application.properties

```
# Activate all actuators (*) or only refresh actuator
management.endpoints.web.exposure.include=*  (additional entry)
```

step3) place @RefreshScope on the top RestController of In Ms Project

```
@RestController
@RequestMapping("/emp")
@RefreshScope
public class EmployeeOperationsController {
    ...
    ....
    ....
}
```

step4) start Eureka Server ,Config Server , MsProject in regularfashion

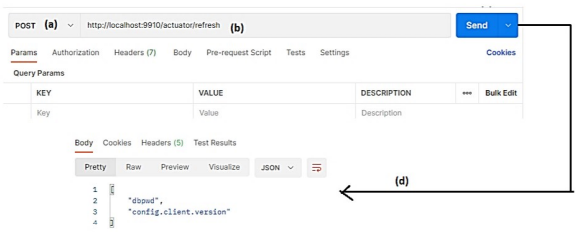
step5) Test the Ms using the regular url http://desktop-iudaavl:9910/emp/show

step6) Modify entries in application.properties file of GitLab account (External Config file) open properties file in git lib --> edit web editor ----> ....

step7) Test the MS using regular url http://desktop-iudaavl:9910/emp/show (Modifications does not reflect suprisingly)

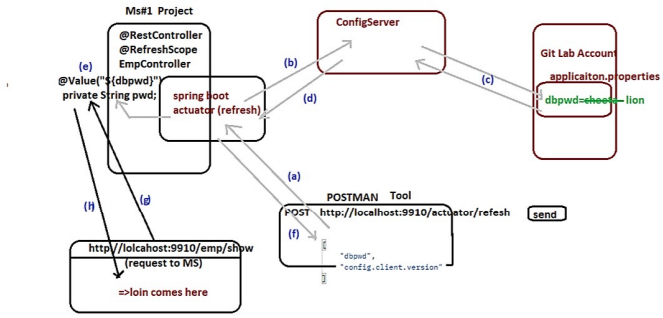
step8) Gather EndPoint details of "refresh" spring boot actuator and given POST mode request to it using POSTMAN tool.

"refresh" actuactor url is http://localhost:9910/actuator/refresh



step9) Test the MS using regular url

http://desktop-iudaavl:9910/emp/show
(chanages will reflect)



spring Boot Actuactors

Spring Boot Actuators

Production Sever :: The Server in which Application is deployed to give services to endusers for commercial use is called Production Server. This Server manages Live Code and Live DB..

Old days :: Dedicated high end System for Producttiion Server

recent days :: using Cloud env (rental basis infrastructure) like AWS , Google Cloud and etc..)

Production Enviroment /Setup :: The s/w's and tools that are used in production server to deploy and run the App is called Production env/setup

eg:: jdk s/w , wildfly server/Tomcat server , jenkins, Docker and etc..

EndPoint :: The Rest apis /Rest Service /some other service that is ready to use is called Endpoint.. In order take the services from any EndPoint we need gather End point details like url , method type , path variables and etc..

=>Each RestController we develop spring boot App is called one Endpoint and to use that endpoint we need to the above said details

eg : http://localhost:9910/emp/show (URL) , GET (method type) , {id} , {name} are path variables, Input type : JSON, output type: String and etc..

Spring Boot Actuator :: It is production ready endpoint (buit-in Rest Service) given by spring boot people to perform non-functional operations on the production env.. spring boot project.

eg :: health , info , configprops ,beans ,logging, refresh and etc.. (generally we get 13+ ready made actuators)

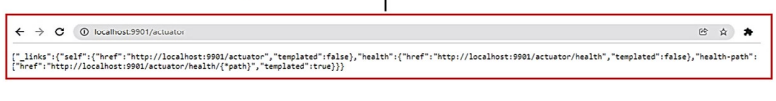
=> Upto spring boot 2.5 we used to see two actuators as activated/enabled actuators by default they are health, info
=> From spring boot 2.6 they are giving only health as defaultly activated actuators.

=> In any Ms Project add "spring boot actuators" dependency in order use spring actuator services.

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

Right click spring boot project --> spring --> add starter --> select spring boot actuator

URL to see all the defaulty activated actuators
http://localhost:9901/actuator
Current App port number



Jan 22.1 RefreshScope and Actuactors

Example app

==========

=> create Spring Boot Starter Project (As RestService / Not As MicroService) adding web , actuator dependencies
(No need of adding DiscoryClient, Config Client)

=> Developе RestController as shown below

```
@RestController
@RequestMapping("/cust")
public class CustomerOperationsController {
    @GetMapping("/display")
    public String displayData() {
        return "(Customer)Customer Info will be displayed here .. welcome Call Center ";
    }
}
```

=> add following entries in application.properties

```
# server port (MS Port)
server.port=9901
# service name or applicaiton name
spring.application.name=CUST-SERVICE
```

=> Run the App as spring boot App

=> get the defaultly activated list of actuactors.

type the url http://locahost:9901/actuator


```
{"_links":{
"self":{"href":"http://localhost:9900/actuator","templated":false},
"health":{"href":"http://localhost:9900/actuator/health","templated":false},
"health-
path":{"href":"http://localhost:9900/actuator/health/{*path}","templated":true}
}}
```


this is again copied and simplified from webpage of that url to readable and as it is given before this example code

(the data is coming in the form of JSON format)

---

health

=> This actuator gives health metrics info like Currenet App is up or down

=>Using this actuator we can also get more detais related memory like Diskspace , free space used space and etc..

=> It is defaulty activated spring boot actuator.

=>These actuators services are very useful for Project maintainace team generally that is DevOps Team

=>To get memory details about the application we need to add one more supporting entry in application.properties.

In application.properties

```
#To get Memory details default using health actuator
management.endpoint.health.show-details=always
```

(the possible values are always,never(default), when-authorized)


```
{"status":"UP","components":{"diskSpace":{"status":"UP","details":{"total":185532936192,"free":145165168640,"threshold":10485760,"exists":true}},"ping":{"status":"UP"}}}
```


```
management.endpoint.health.show-details=always
    (always gives health metrics(Memory details) though u have not logged to the App)
management.endpoint.health.show-details= never
    (Does not give health metrics)
management.endpoint.health.show-details= when-authroized
    ( gives health metrics(Memory details) only for the logged in user of the App i.e authentication and authorizations are completed.)
```

To disable any atcuator

In application.properties

```
management.endpoint.<endpoint-id>.enabled=false
                                    (default is true)
```

these are like health, beans,info and etc..

To enable certain actutor which is not activated by default

In application.properties

```
management.endpoints.web.exposure.include=info
    (Activates only info actuator)

management.endpoints.web.exposure.include=info,health
    (Activates only info,health actuators)

management.endpoints.web.exposure.include=*
    (Activates all the actuators)
```

when all actuators are activated the list looks like this


```
{"_links":{"self":{"href":"http://localhost:9901/actuator","templated":false},"beans":{"href":"http://localhost:9901/actuator/beans","templated":false},"caches":
{"href":"http://localhost:9901/actuator/caches","templated":false},"caches-cache":{"href":"http://localhost:9901/actuator/caches/{cache}","templated":true},"health":
{"href":"http://localhost:9901/actuator/health","templated":false},"health-path":{"href":"http://localhost:9901/actuator/health/{*path}","templated":true},"info":
{"href":"http://localhost:9901/actuator/info","templated":false},"conditions":{"href":"http://localhost:9901/actuator/conditions","templated":false},"configprops-prefix":
{"href":"http://localhost:9901/actuator/configprops/{prefix}","templated":true},"configprops":{"href":"http://localhost:9901/actuator/configprops","templated":false},"env-
toMatch":{"href":"http://localhost:9901/actuator/env/{toMatch}","templated":true},"env":{"href":"http://localhost:9901/actuator/env","templated":false},"loggers-name":
{"href":"http://localhost:9901/actuator/loggers/{name}","templated":true},"loggers":{"href":"http://localhost:9901/actuator/loggers","templated":false},"heapdump":
{"href":"http://localhost:9901/actuator/heapdump","templated":false},"threaddump":{"href":"http://localhost:9901/actuator/threaddump","templated":false},"metrics-
requiredMetricName":{"href":"http://localhost:9901/actuator/metrics/{requiredMetricName}","templated":true},"metrics":
{"href":"http://localhost:9901/actuator/metrics","templated":false},"scheduledtasks":{"href":"http://localhost:9901/actuator/scheduledtasks","templated":false},"mappings":
{"href":"http://localhost:9901/actuator/mappings","templated":false}}}
```


Info

=>Gives info about current application in the form JSON content when user request to this "info" atuator gives

=>For that we need to maintain information about application in application.proeprties file having fixed prefixes in keys (the is info.app)

example

=> keep spring rest app ready (same as previous)

=> activate "info" or all actuators

In application.properites

---

management.endpoints.web.exposure.include=info (or) management.endpoints.web.exposure.include=*

=> add more info about the application. in application.properties having fixed prefix in keys (info.app)

In application.properties

```
# info about application
info.app.name=CUST-Service
info.app.id=45467
info.app.size=10modules
info.app.vendor=Naresh IT
info.app.createdBy=Team-S
```

To disable info actuator management.endpoint.info.enabled=false

=> enable info enviorment to read about current app

In application.properties

```
management.info.env.enabled=true
```

=>Test the info acuator Application


```
{"app":{"name":"CUST-Service","id":"45467","size":"10modules","vendor":"Naresh IT","createdBy":"Team-S"}}
```


---

jan 24 Spring Boot Actuactors

What is the benifit of spring boot actuators ?

=>These are non-functional features that required in any project's deployment and maintainance .. Earlier teams used dependent various tools given by jdk and thirdparty to get these non-functional features..

=>For example when bug is raised we need logging info and threaddump info
Earlier people have used seperate logging tools like log4j , slf4j and etc..
also seperate thread dump tools like jconsole , JMC , jstack ,Visual Vm and etc..
After the arrival of spring boot .. we can use actuators support like logging , threaddumps for same.

url To get all actuators:- http://localhost:9901/actuator

List of imp actuators

health ,info , configprops , mappings, threaddump, heapdump, env, beans, refresh loggers , scheduledtasks,cache,metrics

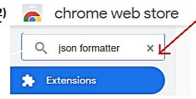
beans :: Gives info about all spring beans of current spring boot App (Both AutoConfigured and manually configured)

URL :: http://localhost:9901/actuator/beans

How to add Json Formatter Extension to chrome?

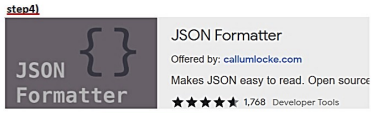
step1) in chrome search for chrome web store and open the home page of chrome web

step2)



step3) search for json formatter

step4)



step5) click on Add to Chrome

---

configprops :: Gives all the @ConfigurationProperties based key=value loadings done both user-defined and pre-fined spring bean classes.

http://localhost:9901/actuator/configprops

env :: Gives the current env.. of the Project in the form of key=value pairs like jars files added classpath , cpu details , os details and etc..

http://localhost:9901/actuator/env

loggers :: Gives all log messages related details along withe their logging levels.

http://localhost:9901/actuator/loggers

scheduledtasks : Gives all @Scheduled methods info like initialDelay , fixedDelay,fixedRated, cron and etc.. details

threaddump :: Gives info about all the daemon threads(backgrounds) that are created running continusoly

http://localhost:9901/actuator/threaddump

mappings :: gives info all handler methods , RestController methods mappings info like request path , method name, method signature , method type and etc..

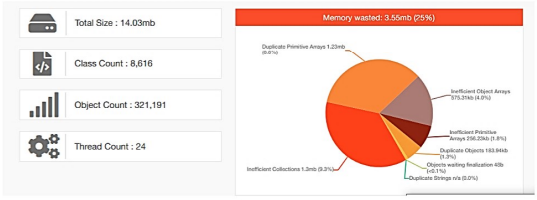
http://localhost:9901/actuator/mappings

heapdump :: Gives gives heapdump (memory details) in the form downloade byte code info..
So we need addtional tools to analyze that content like Eclipse Memeory Analyzer , IBMheapMemoryAnalyzer , heap hero and etc...

http://localhost:9901/actuator/heapdump ⟶ gives "heapdump" file in downloads folder havig byte /machine code

To analyze the heap dump

go to https://heaphero.io/ ⟶ select the above heapdump file ⟶ get the analyzation report



caches :; gives info all CacheManagers that are configured in different parts of the application.

http://localhost:9901/actuator/caches

metrics :: gives Info about properties list related to current project.

http://localhost:9901/actuator/metrics

What is the base path for actuators?

nothing but url to see actuators:- http://localhost:9901/actuator

Can we change the basepath of actuator?

=>Like changing context path of web application.. we can also change basepath or context path of actuators

in application.properties

```
# To change basepath of actuctors
management.endpoints.web.base-path=/nitinfo
```

To test the change

http://localhost:9901/nitinfo
http://localhost:9901/nitinfo/health
http://localhost:9901/nitinfo/beans

note:: we can not take "/" as the basepath for actuators becoz it is already assigned to DispatcherServlet.

Q) While working with "refresh" actuator if Extenal Config file (GitLab account file) and local application.proeprties file of the MS project contains same keys with different values Can u tell me which value will be injected.

GitLab's application.properties

```
dbuser=system
```

local application.properties

```
dbuser=ramesh
```

In Ms App

```
@Value("${dbuser}")
private String user; -- gets what value :: system
```

(takes from External ConfigServer)

jan 25 CircutBreaker In Spring Boot Ms

Circut Breaker in spring Boot MicroSErvices

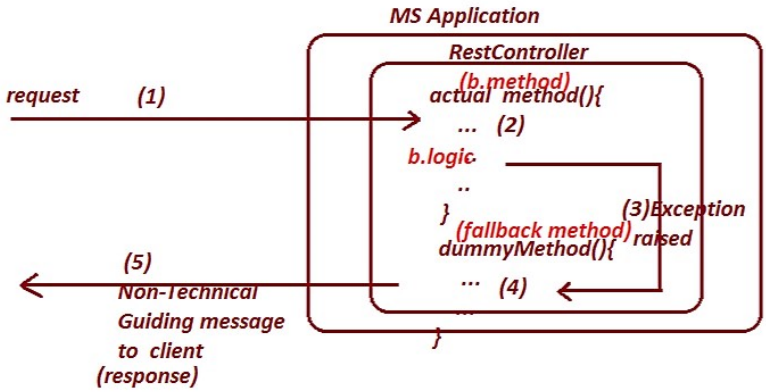
==========

=> if the method b.method of Application is continuosly throwing exception then avoid execution of actual b.method logic and provide dummy response or non-technical guiding response to Client is called working with fallback method with circut breaker.

What is fallback method?

=> The method the executes automatically when the exception raised in the actual b.method to provide non-technical guiding messages to enduser is called fallback method.

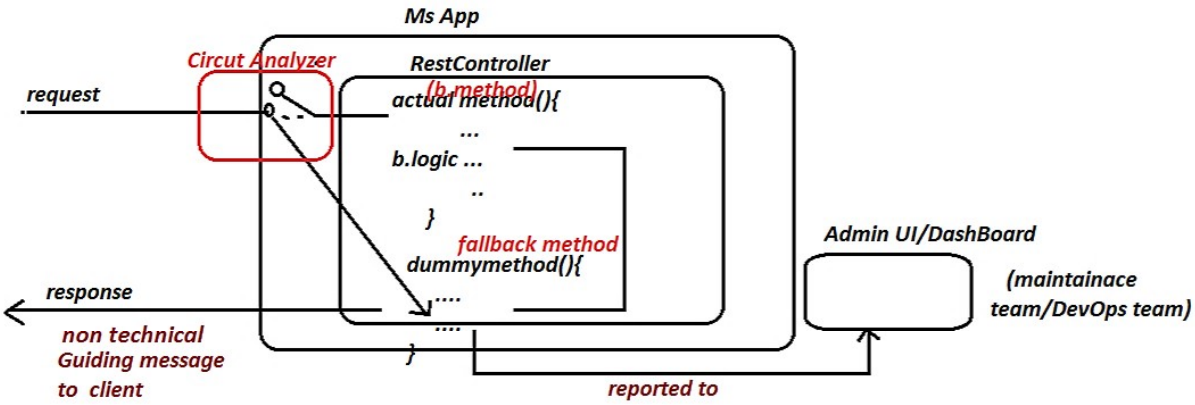
The non-technical guiding messages are " Service not found " , "please try after some time" "Inconvience regratted" , " service down due technical issues" , " site on maitainace" and etc..



What is fallack method with Circuit breaker support?

if the actul b.method of the application is continuosly throwing exception.. then avoid sending request b.method for some amount of time and make DispatcherSErvlet kind of FrontController redirecting request of actual b.method to dummy fallback method again for certain amount of time is called fallback method with circut breaker.

=>let us assume the actual b.method has thrown exception for 20 times already and the fallack method is executed .. if circut breaker concept is added the next few requests given actual method directly goes to dummy method for certain amount of time . this is called fallback method with circuit breaker.



case1:: actual method has got 20 requests and all of thrown exception so fallback methid is executed (only fallack method)

case2: after 20 times exeception raising continuosly(back to back) in actual method the next requests given to actual method goes to fallaback directly to deliver non-technical guidling messages for some amount time .. after that case1 repeates (fallback with circuit breaker)

jan 25.1 CircutBreaker In Spring Boot MS

Two types of Circuits on Circuit Breaker with Fallback method mechansim

==========

(a) Open Circuit :: It indicates that the b.method had continuosly throws exception (back to back) So circut to b.method is broken (open) for client requests.. So client request will redirect to dummy fallback method as shown in the diagram for certain amount of time

(b) Closed Circuit :: It indicates the request given by client goes to b.method directly

Realtime use-cases for Circuit breaker impl ::

a) card payment b) Online flash sale c) online exam results d) online ticket booking and etc..

=>To implement Circut breaker concept in Spring Boot Ms we need to use spring Boot hystrix implementation given by nextfilx For this we need two annotation

a)**@EnableHystrix** on the top of Main class

b)**@HystrixCommand** on the top of actual b.method specifying the dummy method name

Example App on fallback method

==========

step1) create spring boot Project adding web , hystric starters.
(Choose spring boot version as <=2.3)

note : Latest versions of spring boot not support netflix hystrix .. so use resilliance4j as the alternate

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

```
<!-- https://mvnrepository.com/artifact/org.springframework.cloud/spring-cloud-starter-
netflix-hystrix -->
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
  <version>2.2.10.RELEASE</version>
</dependency>
```

step2) add @EnableHystrix on the main class

step3) Develop RestController as shown below having b.method and dummy method

```
@RestController
@RequestMapping("/ticket")
public class TicketBookingRestController {

    @GetMapping("/book")
    @HystrixCommand(fallbackMethod = "dummyBookTicket")
    public String bookTicket() {
        System.out.println("TicketBookingRestController.bookTicket()");
        if(new Random().nextInt(10)<5)
            throw new RuntimeException("Problem is b.logic");

        return "output from b.logic";
    }

    public String dummyBookTicket() {
        System.out.println("TicketBookingRestController.dummyBookTicket()");
        return "Place Try later -- Inconvience is regratted";
    }

}
```

This dummy fallback method must be there is same class that to not having any parameters.

step3) Run the App and Test the Application.

http://localhost:9901/ticket/book

jan 27 CircutBreaker -Hystrix DashBoard

=>Spring Boot circuit breaker is the concept and implementations netflix hystrix and resilience4j

=> with respect to hystrix implementation

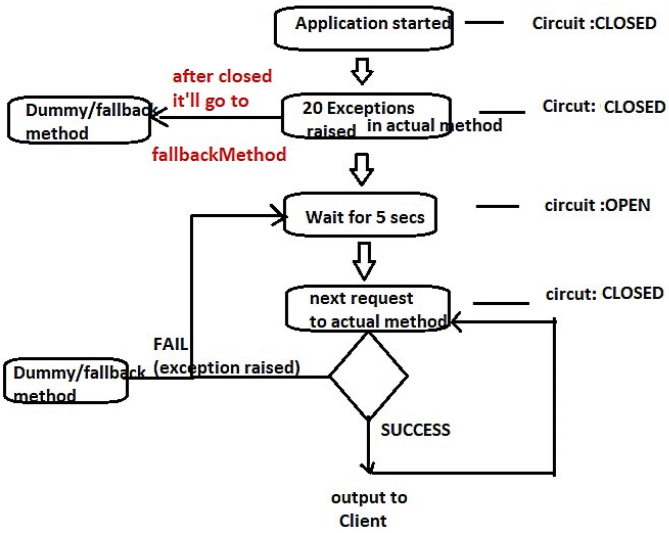
=>When the Applicaiton is started the Circuit will remain closed i.e intially request goes to atual method

=>The default exception count is : 20 (After raising exception for 20 times by actual method the curcuit will open)

=> The default retry-time is :: 5 sec (after getting 20 exceptions the circuit will open next 5sec i.e request will go to fallback method directly in these 5 secs and circuit will be closed after this 5 sec)

request

=> Once Circuit is ReClosed .. the next goes to actual method .. if exception is raised the circuit will be again opened for 5 secs .. and so on...



Example1 (fallbackMethod with Circuit Breaker)

step1) create spring boot project taking version as 2.3.6.RELEASE and adding web , netflix hystrix dependencies

step2) add @EnableHystrix on the main class

step3) Develop RestController having @HystrixCommand to specify dummy fallback and to enable circuit breaker

```
package com.nt.controller;

import java.util.Random;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.netflix.hystrix.contrib.javanica.annotation.HystrixCommand;
import com.netflix.hystrix.contrib.javanica.annotation.HystrixProperty;

@RestController
@RequestMapping("/ticket")
public class TicketBookingRestController {
      int count=0;

     @GetMapping("/book")
     @HystrixCommand(fallbackMethod = "dummyBookTicket",
         commandProperties = {
                        @HystrixProperty(name="circuitBreaker.enabled", value="true")
                    })
      public  String   bookTicket() {
            System.out.println("TicketBookingRestController.bookTicket()");
             if(new Random().nextInt(10)<10)
                   throw new RuntimeException("Problem is  b.logic");
              return "output from  b.logic";

      }

      public  String dummyBookTicket() {
            count++;
            System.out.println("TicketBookingRestController.dummyBookTicket()::"+count);
            return "Place Try later -- Inconvience is regratted";
      }

}
```

enables the circuit breaker

jan 27.1 CircutBreaker -Hystrix DashBoard

step4) give 20 to 3 contionuos requests to RestController from brrowser observe server consoe

http://localhost:9901/ticket/book

```
TicketBookingRestController.bookTicket()
TicketBookingRestController.dummyBookTicket()::1
TicketBookingRestController.bookTicket()
TicketBookingRestController.dummyBookTicket()::2
TicketBookingRestController.bookTicket()
TicketBookingRestController.dummyBookTicket()::3
TicketBookingRestController.bookTicket()
TicketBookingRestController.dummyBookTicket()::4
TicketBookingRestController.bookTicket()
TicketBookingRestController.dummyBookTicket()::5
TicketBookingRestController.bookTicket()
TicketBookingRestController.dummyBookTicket()::6
TicketBookingRestController.bookTicket()
TicketBookingRestController.dummyBookTicket()::7
TicketBookingRestController.bookTicket()
TicketBookingRestController.dummyBookTicket()::8
TicketBookingRestController.bookTicket()
TicketBookingRestController.dummyBookTicket()::9
TicketBookingRestController.bookTicket()
TicketBookingRestController.dummyBookTicket()::10
TicketBookingRestController.bookTicket()
TicketBookingRestController.dummyBookTicket()::11
TicketBookingRestController.bookTicket()
TicketBookingRestController.dummyBookTicket()::12
TicketBookingRestController.bookTicket()
TicketBookingRestController.dummyBookTicket()::13
TicketBookingRestController.bookTicket()
TicketBookingRestController.dummyBookTicket()::14
TicketBookingRestController.bookTicket()
TicketBookingRestController.dummyBookTicket()::15
TicketBookingRestController.bookTicket()
TicketBookingRestController.dummyBookTicket()::16
TicketBookingRestController.bookTicket()
TicketBookingRestController.dummyBookTicket()::17
TicketBookingRestController.bookTicket()
TicketBookingRestController.dummyBookTicket()::18
TicketBookingRestController.bookTicket()
TicketBookingRestController.dummyBookTicket()::19
TicketBookingRestController.bookTicket()
TicketBookingRestController.dummyBookTicket()::20
TicketBookingRestController.dummyBookTicket()::21
TicketBookingRestController.dummyBookTicket()::22
TicketBookingRestController.dummyBookTicket()::23
```

Circuit is closed for 20 exceptions So acutal is executing becoz of exception that is riased control is going to fallback method.

for 5 sec only fallback executes becoz the circuit open

Hystrix Command properties

For hystrik command properties
---
https://www.logicbig.com/tutorials/spring-framework/spring-cloud/hystrix-configuration-properties.html

```
@RestController
@RequestMapping("/ticket")
public class TicketBookingRestController {
	int count=0;

	@GetMapping("/book")
	@HystrixCommand(fallbackMethod = "dummyBookTicket",
		commandProperties = {
				@HystrixProperty(name="circuitBreaker.requestVolumeThreshold", value="5"),
				@HystrixProperty(name = "circuitBreaker.sleepWindowInMilliseconds", value = "10000"),
				@HystrixProperty(name = "circuitBreaker.enabled", value = "true")
					}
		)
	public String bookTicket() {
		System.out.println("TicketBookingRestController.bookTicket()");
		if(new Random().nextInt(10)<8)
			throw new RuntimeException("Problem is b.logic");

		System.out.println("End of Ticket Booking Opeation");
		return "output from b.logic(Success)";
	}
	public String dummyBookTicket() {
		count++;
		System.out.println("TicketBookingRestController.dummyBookTicket()::"+count);
		return "Place Try later -- Inconvience is regratted";
	}
}
```

circuitBreaker.requestVolumeThreshold :: *allows us to specify exception count (default is 20)*

circuitBreaker.sleepWindowInMilliseconds :: *allows to sepcify retry sleep time (default is 5 sec)*

circuitBreaker.enabled :: *allows us to specify wheather circuit beaker should be enabled or not (default is false)*

step4) give 5 to 6 contionuos requests to RestController from brrowser observe server consoe

http://localhost:9901/ticket/book

```
TicketBookingRestController.bookTicket()
TicketBookingRestController.dummyBookTicket()::1
TicketBookingRestController.bookTicket()
TicketBookingRestController.dummyBookTicket()::2
TicketBookingRestController.bookTicket()
TicketBookingRestController.dummyBookTicket()::3
TicketBookingRestController.bookTicket()
TicketBookingRestController.dummyBookTicket()::4
TicketBookingRestController.bookTicket()
TicketBookingRestController.dummyBookTicket()::5
TicketBookingRestController.bookTicket()
End of Ticket Booking Opeation
TicketBookingRestController.dummyBookTicket()::6
TicketBookingRestController.dummyBookTicket()::7
TicketBookingRestController.dummyBookTicket()::8
TicketBookingRestController.dummyBookTicket()::9
TicketBookingRestController.dummyBookTicket()::10
```

becoz of circut is closed

becoz of circuit is open

jan 28 Hystrix DashBoard and Resiliance4J

Hystrix DashBoard

=>It is given to provide GUI env.. to know current running MS details and its Circuit Breaker Details like checking wheather Circuit is open or close.

=>Since hystrix is kept in maintainace mode or deprecated mode in latest spring boot version , we need to use bit old spring boot versions like <2.4

=> For this we need to add new dependencies like hystrix dashboard and we need to place @EnableHystrixDashboard on the main class /starter class..

=>CircuitBreaker concept Desing Pattern to resovle the issues related to fault tolerance.

=>Circuit breaker concept says when there is possibility of getting error while using certains microservice locally or remotely .. then make request going certain fallback method after getting certain count of continuos exceptions/errors from the actual method... This avoid wastage of resources like CPU utilization or network intrastructure and etc..

Example

step1) create spring boot project adding web , hystrix , hystrix dashboard,actuators as dependencies..

from mvnrepository,com

```
<!-- https://mvnrepository.com/artifact/org.springframework.cloud/spring-cloud-starter-netflix-hystrix -->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
    <version>2.2.10.RELEASE</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework.cloud/spring-cloud-starter-netflix-hystrix-dashboard -->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-hystrix-dashboard</artifactId>
    <version>2.2.10.RELEASE</version>
</dependency>
```

from list of starters

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

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

step2) modify or add spring cloud version in pom.xml file and perform maven update

```
<properties>
    <java.version>11</java.version>
    <spring.cloud.version>Hoxton.SR4</spring.cloud.version>
</properties>
```

To make the project compitable with hystrix dashBoard.

step3) add @EnableHystrix , @EnableHystrixDashBoard on main class.

step4) Develop MS as RestController having @HystrixCommand to specify fallback method

```
@RestController
@RequestMapping("/ticket")
public class TicketBookingRestController {
 int count=0;
@GetMapping("/book")
@HystrixCommand(fallbackMethod = "dummyBookTicket",
                commandProperties = {
                @HystrixProperty(name="circuitBreaker.requestVolumeThreshold", value="5"),
                @HystrixProperty(name = "circuitBreaker.sleepWindowInMilliseconds", value = "10000"),
                @HystrixProperty(name = "circuitBreaker.enabled", value = "true")
                                    }
                )
public String bookTicket() {                                        //same as previous class
      System.out.println("TicketBookingRestController.bookTicket()");
       if(new Random().nextInt(10)<8)
            throw new RuntimeException("Problem is b.logic");

       System.out.println("End of Ticket Booking Opeation");
       return "output from b.logic(Success)";

}

public String dummyBookTicket() {
      count++;
      System.out.println("TicketBookingRestController.dummyBookTicket()::"+count);
      return "Place Try later -- Inconvience is regratted";
}

}
```

jan 28.1 Hystrix DashBoard and Resiliance4J

step4) Add addtional properties in application.properties fil;e

application.properties

```
server.port=9901

#enable actuators
management.endpoints.web.exposure.include=*

# enable hystrix streaming list
hystrix.dashboard.proxyStreamAllowList=*
```

step5) Run the Application.. (Test through DashBoard)

a) give reqeust to MS

http://localhost:9901/ticket/book

b) open dashboard of hystrix

http://localhost:9901/hystrix

enter following url related to our app below the image of dashboard

1

Hystrix Dashboard

http://localhost:9901/actuator/hystrix.stream

Cluster via Turbine (default cluster): https://turbine-hostname:port/turbine.stream
Cluster via Turbine (custom cluster): https://turbine-hostname:port/turbine.stream?cluster=[clusterName]
Single Hystrix App: https://hystrix-app:port/actuator/hystrix.stream

Delay: 2000 ms Title: Example Hystrix App

Monitor Stream ***2***

c) gives multiple requests our MS application using browser (refresh button) and observe the hystrix dash board messages

http://localhost:9901/ticket/book

---

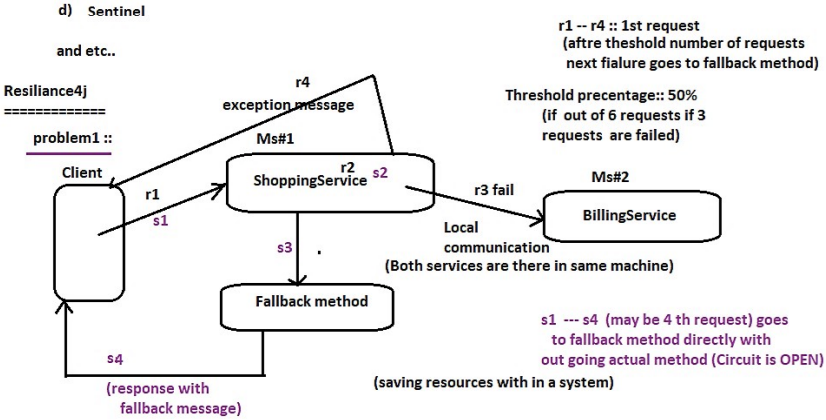
spring cloud Circuit Breaker Impl using resilience4J

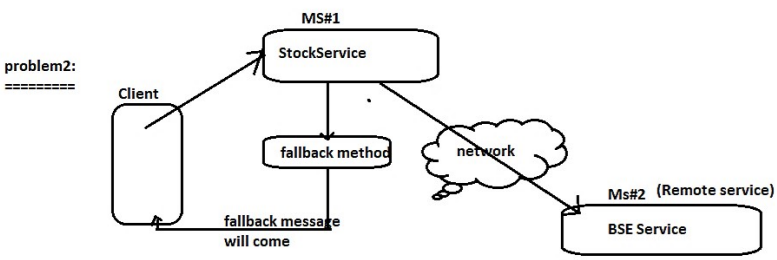
=>spring cloud circuit breaker concept is given as design pattern for fault tolerance settings and that is implemented by different vendors

a) Netflix hystrix (upto spring boot 2.3)
(deprecated in 2.4 and later removed)
b) Resiliance4j (from 2.4)
(Latest)
c) spring retry

d) Sentinel

and etc..





Saving resources of Ms System and alos network resources.

To solve the problem problems .. we can take the support resiliance4j which keeps circut in 3 states

=>Closed state :: success ---> dest Ms is executing through actual method soruce Ms
=> Open state :: After Threshold percentage is statisfied The circut will open and does not attempt to talk with dest comp
=>Half open state :: after staying open state for certain amount time it comes to Half open state to check actual Service is ready by generating internal implicit requests.. if ready .. the goes closed state.. if not ready goes to again open state.

feb 01 Messaging Intro - Feb1st 2022

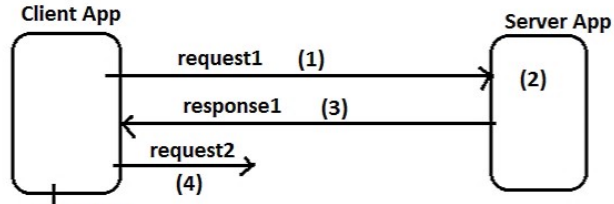
Messaging (Communication using Messages)
==========

The Client - Server Communication is of two types

a) Synchronous communication
b) Ashynchronous Communication

a) Synchronous communication

In this communication, the client App can send next request to Server App only after getting the response for the already given request i.e The Client App should wait for given request related response from server App in order to make it next request to server App or to perform some client side operations.



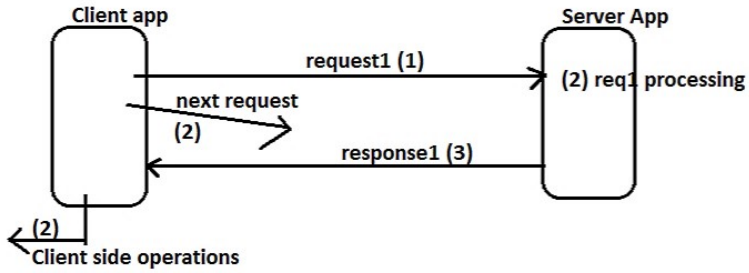
in synchrnous communication .. both Client App and server App must be active at time

Here the Client App is blocked to generate next request or to perform client side operations until given request related response comes back from server App.

note:: By default all Client - server communications are synchronous communincations

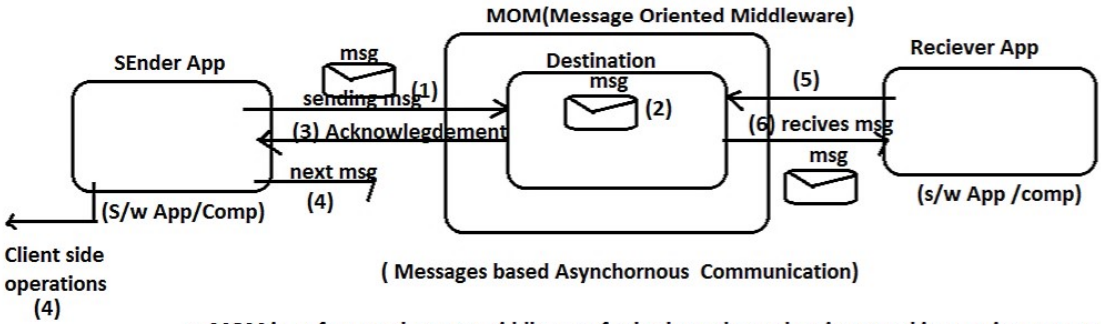
b) Ashynchronous Communication

=>Here Client App is free (not blocked) to generate the next request or to perform client side operations with out waiting for the given request related response from server App..



=>In web applications we can use AJAX (Asynchronous Java Script and Xml) to asynchrnonous communication b/w browser(Client) and web application(server) (now ajax is part of UI Technologies)

=> we can use "Messaging concept" (Messages based communication) to get Asynchronous communication b/w two software Apps /comps.



(Messages based Asynchornous Communication)

=> MOM is software that acts middle man for both sender and reciever and it contains memory called Destination to hold messages sent by sender App and to deliver the same messages to the Reciver App

note :: In Messaging the messages will go from one App to another App in continuos flow like stream.

note :: SMS messaging, whatup messaging , mailing does not come under messages based communication becoz that deals with person to person communication.. the actual messaging takes place b/w two software Apps or comps.

feb 01.1 Messaging Intro - Feb1st 2022

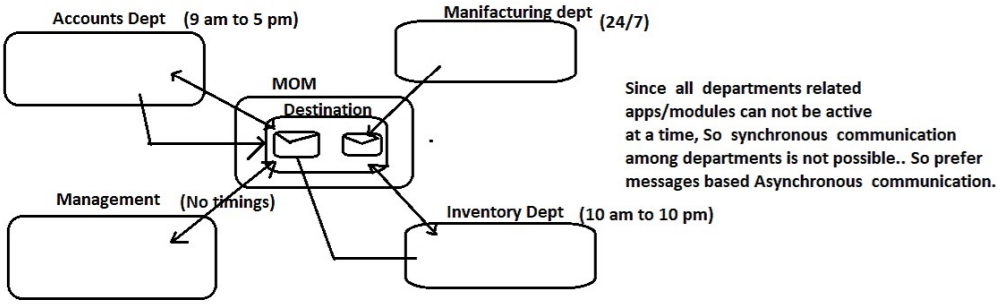
use-cases for Messaging ::
==========

a) Data streaming based on Scheduled /continous Flow
(eg:: uber cab availability status , Goods delivery App store availabiity status , Live Train Running status , Live cricket score status and and etc..)

b) Web activities and serach results
(eg: The advertisement agecy App gathers google search queries continuosly to Aggregate search queries)

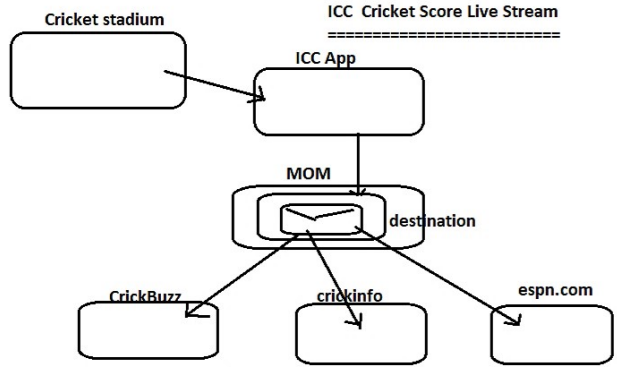
c) Log Aggregation
(eg:: Collecting log messages generated by Production env. App and aggregrating their those messages)

Car manifacturing company
==========



Since all departments related apps/modules can not be active at a time, So synchronous communication among departments is not possible.. So prefer messages based Asynchronous communication.

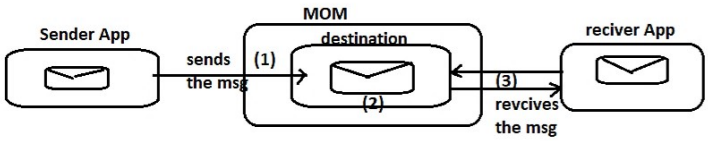
ICC Cricket Score Live Stream
==========



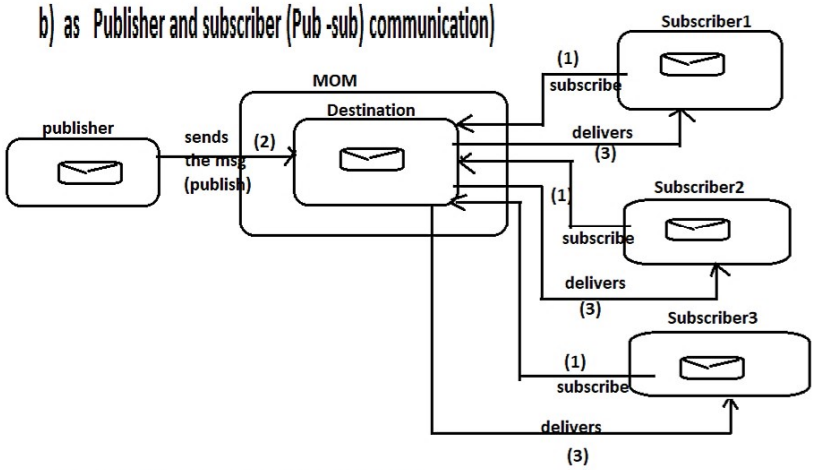
This Messaging based communication can be done in two ways
a) as Point to Point (PTP) communication
b) as Publisher and subscriber (Pub -sub) communication)

a) as Point to Point (PTP) communication

=>Here each message send to destination by sender will have only one consumer/Reciver i.e once consumer consumes the messages the message will be removed from the destination.



b) as Publisher and subscriber (Pub -sub) communication)



=> Here each messages sent by the publisher will go multiple subscribers who have done their subscription to destination before publisher publishes the message.
(each message can have 0 or more subscribers)

feb 02 Messaging using JMS

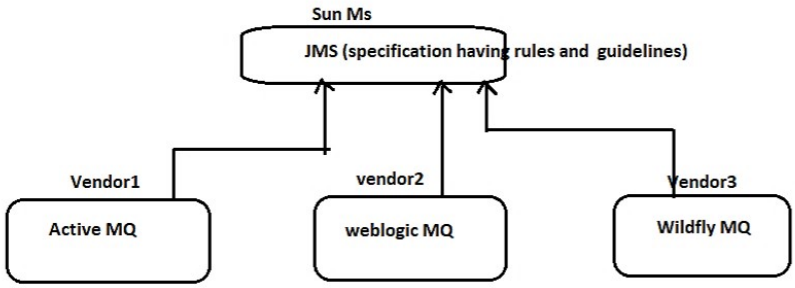
=> By Default all Client app -server app communication is synchronous communication
eg:: browser to web application , Rest Client to Rest Service , Ms Client to Ms , MS to MS.
=> In web application to get asynchronous communication b/w browser and web application take the support of AJAX .
=> Between two Java Apps or two MS or Rest Client And RestService if u r looking for Messages based asynchronous communication then for Messaging/Message Quees with the support of MOM software.

=> Messaging /Message Queue can be implemented using two types of porotocols
a) Using Basic MQ Protocol (BMQP) :: For this we need to use JMS
b) Using Adavanced MQ protocol (AMQP) :: For this we need to use RabitMq, Apache kafka and etc..

==========
Messaging /Message Queue using JMS
==========

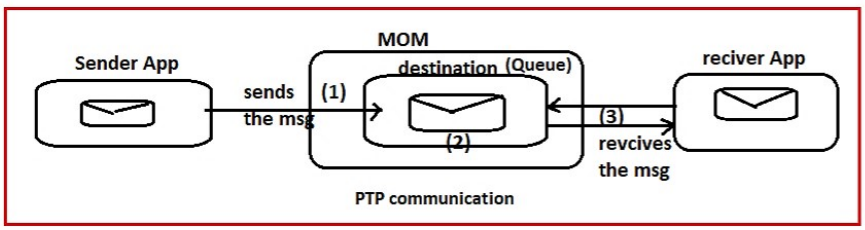
jakarta
(JMS :: ~~Java~~ Messsaging Service)
Java Mail Api is different from JMS

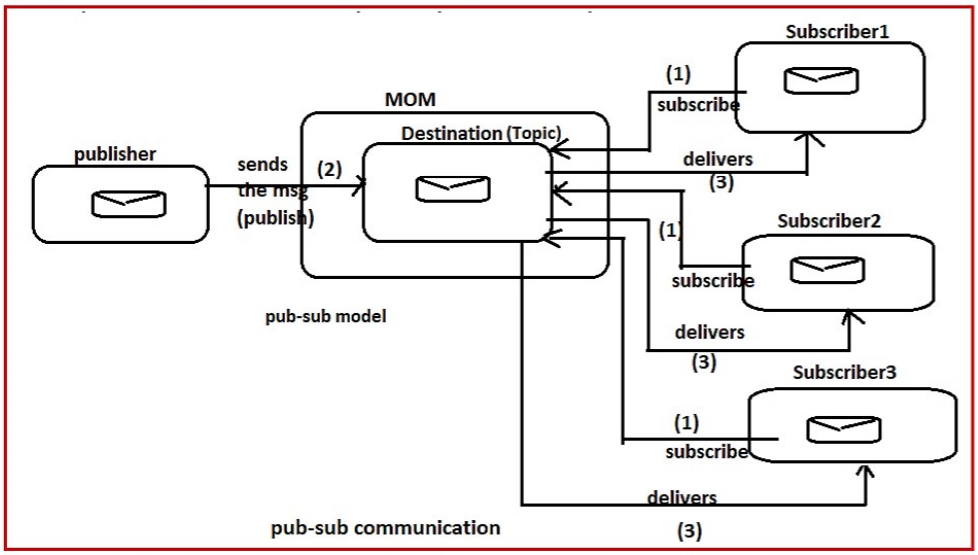
=> JMS is the software specification given by Sun Ms in JEE module having set of rules and guidelines to provide messsages based communication b/w java comps or Apps.



=> Each Implementation software of JMS given by different vendors is called one MOM software
=> Every Application server s/w like weblogic , wildfy and etc.. given one built-in MOM software
=> Tomcat is not providing any MOM software..so if u r application is using tomcat server then prefere working with Active MQ as the MOM software.

=> JMS supports both PTP and pub-sub Models of Messaging
=> The MiddleMan software between send and reciver/consumer who takes messages given by sender and holds them for Reciver to come and cosume is called MOM
=> MOM contains destinations as the logical mememories to recived and hold the messages..
In PTP model this destination is called "Queue" and in pub-sub model this destination is called "Topic"





=> JMS provides api representing rules and guidelines in the form of interfaces,classes placed in javax.jms or jakarta.jms and its subpkgs. (It is like JDBC API)

=> Active MQ, weblogic MQ, WildlfyMQ and etc.. are MOM softwares which are internally providing impl classes for JMS API intefaces (These are like JDBC DRIVERS)

=>Programmers takes impl classes through JMS interrfaces .. to create objects and to consume services [The Sender and reciver Apps will be developed using JMS Api referring one or another MOM Impl sofwares) (These are like Java Apps using JDBC drivers through JDBC API)

feb 02.1 Messaging using JMS

sample reference code to understand API , Impl software and App development using API

```
interface Operation1{
  public void process();
}
interface Operation2{
  public String process(String msg);
}
```

It is like JMS API given by Sun Ms (JMS specification)

```
public class Operation1Impl implements Operation1{
  public void Process(){
    ....
  }
}
public class Operation2Impl implements Operation2{
  public String Process(Stirng msg){
    ....
  }
}
```

=> impl classes provided by the Vendor

(It is like active Mq , weblogic mq and etc.. MOM softwares give by different Vendors

```
Operation1 op1=new Operation1Impl();
op1.process();
Operation1 op2=new Operation2Impl();
String result= op2.process("hello");
```

It Like JMS application devleoped the Programmer

=>The interface that contains only one abstract method direclty or indirectly is called functional Interface.

```
@FunctionalInterface
interface Operation1{
  public void process(String msg);
}
```

Functional interface

Impl1 ::

```
public class Operation1Impl implements Operation1{
  public void proess(String msg){
    ...
    ..
  }
}
```

Normal Impl class

Impl2 :: Anyonomous inner class (inline impl)

```
Operation1 op1= new Operation1(){
  public void process(Stirng msg){
    ...
    ...
  }
};
```

Here Anonymous inner class is created implementing Operation1 interface and process(-) is implemented in that class.

Impl3 :: Lamda based anonymous inner class (anonymous inner clss Impl class obj + method impl) (InLine impl)

```
Operation op1= (msg)->{ ....
                      }
```

spring

The JMS api is providing one Funcational Interface called "MessageCreator" as shown below

```
@FunctionalInterface
public interface MessageCreator {
    Message createMessage(Session session);
}
```

Impl1 (Using anynomous innner class)

```
MessageCreator mc=new MessageCreator(){
    public Message createMessage(Sesssion ses){
      .... // logic
      ....
      return message obj;
    }
}
```

Impl2 (Using Lamda anonymous inner class)

```
MessageCreator mc=(ses)->{ .....
                           ...
                           return message obj;
                         }
```

(or)

```
MessageCreator mc=ses-> message obj;
```

SpringBoot JMS/spring JMS provides abstraction on plain JMS to simplify the messages based communication with the support of "JMSTemplate" (template method DP)

Procedure to keep ActiveMq as MOM software

step1) Dowload ActiveMQ software as zip file from Internet

https://activemq.apache.org/components/classic/download/

ActiveMQ 5.16.3 (Aug 17th, 2021)

Release Notes | Release Page | Documentation

Windows	apache-activemq-5.16.3-bin.zip	SHA512	GPG Signature
Unix/Linux/Cygwin	apache-activemq-5.16.3-bin.tar.gz	SHA512	GPG Signature
Source Code Distribution:	activemq-parent-5.16.3-source-release.zip	SHA512	GPG Signature

step2) Extract the zip file to a folder.

step3) start the ActiveMQ software

E:\ActiveMQSoft\apache-activemq-5.16.3\bin\win64\activemq.bat file

step4) open admin console page of Active MQ software

http://localhost:8161

username : admin
password : admin

step5) Observer Topic and Queue sections in admin console page

home page ---> manage active mq --->

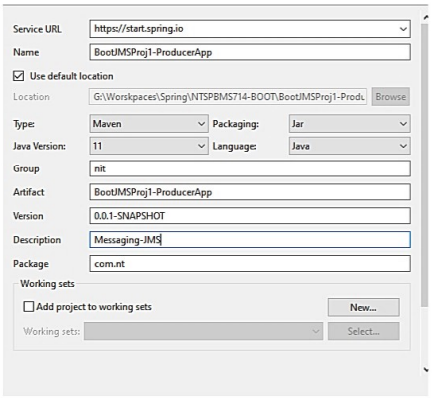
feb 03 Messaging using JMS (PTP Example)

Procedure to develop JMS PTP (Queue) application using active MQ

=>In any JMS App (Producer /Sender or Reciver /Subscriber/consumer) once we spring -boot-starter-activemq starter as dependency we get JMSTemplate class object through AutoConfiguration that can be injected to Sender / Reciver App through Autowiring.

For Producer App

step1) create spring boot project adding active mq starter..




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

step2) add the following properties in application.properties file

```
# MOM connectivity Details
spring.activemq.broker-url=tcp://localhost:61616
spring.activemq.user=admin
spring.activemq.password=admin

#enable PTP communication
# true enables pub-sub model and false enables ptp model
spring.jms.pub-sub-domain=false
```

#8161 for admin console
61616 for actual MOM service

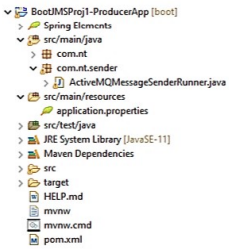
step3) Develop runner class as the Message sender

=>JmsTemplate class is having send(-,-) method taking destiona (queue/topic) logical name(generally new name) and MessageCreator(I) (Functional interface).

```
public void send(Destination destination,   MessageCreator messageCreator)   throws JmsException
```

For this we can pass either anonymous inner class obj or Lamda style inner class obj

Runner class having sender logic




```
package com.nt.sender;

import java.util.Date;

import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.Session;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.jms.core.JmsTemplate;
import org.springframework.jms.core.MessageCreator;
import org.springframework.stereotype.Component;

@Component
public class ActiveMQMessageSenderRunner implements CommandLineRunner {
    @Autowired
    private JmsTemplate template;

    @Override
    public void run(String... args) throws Exception {
  /*      //using anonymous inner class logics
        template.send("testmq1", new MessageCreator() {
            @Override
            public Message createMessage(Session ses) throws JMSException {
                Message message=ses.createTextMessage("From Sender at ::"+new Date());
                return message;
            }
        }); */

        /* using LAMDA style anonyomous inner class
         template.send("testmq1",ses->{
            return ses.createTextMessage("From sender at"+new Date());
        });*/

            //using LAMDA style anonyomous inner class
            template.send("testmq1",ses-> ses.createTextMessage("From sender at"+new Date()));
            System.out.println("Message sent");
    }//run
}//class
```

step4) Make sure that Active MQ started

E:\ActiveMQSoft\apache-activemq-5.16.3\bin\win64\activemq.bat

step5) Run the Sender App and ActiveMQ Server console

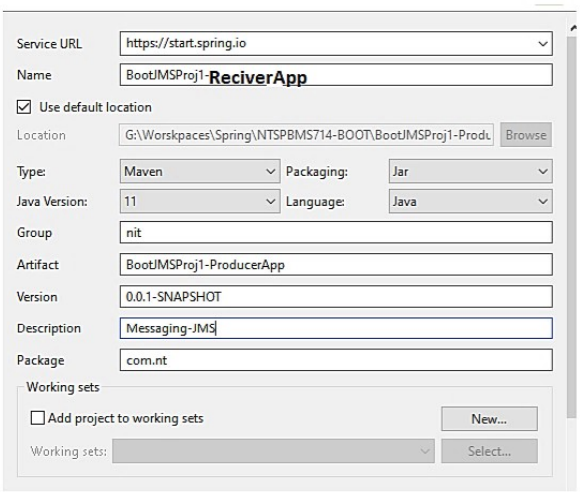
Queues:

Name	Number Of Pending Messages	Number Of Consumers	Messages Enqueued	Messages Dequeued	Views	Operations
testmq1	1	0	2	1	Browse Active Consumers Active Producers atom rss	Send To Purge Delete Pause

feb 03.1 Messaging Using JMS (PTP Example)

Procecedure to develop Consumer/Reciever App of PTP model using ActiveMQ MOM software

step1) create spring boot project adding active mq starter..

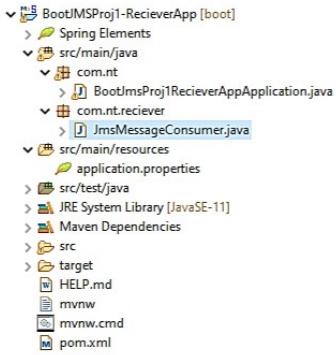



```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-activemq</artifactId>
    </dependency>
```

step2) add the following properties in application.properties file

```
# MOM connectivity Details      #8161 for admin console , 61616 for actual MOM service
spring.activemq.broker-url=tcp://localhost:61616
spring.activemq.user=admin
spring.activemq.password=admin

#enable PTP communication
# true enables pub-sub model and false enables ptp model
spring.jms.pub-sub-domain=false
```

step3) Develope the java class having @JMsListener method as shown below.




```
package com.nt.reciever;

import org.springframework.jms.annotation.JmsListener;
import org.springframework.stereotype.Component;

@Component
public class JmsMessageConsumer {

    @JmsListener(destination = "testmq1")
    public   void  readMessage(String text) {
            System.out.println("Recived Message::"+text);
    }

}
```

These messages executes automactically to read message from queue destination

step4) Run the App Consumer App

(reads the message from Quue destinaiuon)

step5) Observe the console

Queues:

Name	Number Of Pending Messages	Number Of Consumers	Messages Enqueued	Messages Dequeued	Views
testmq1	0	0	2	2	Browse Active Consumers Active Producers atom rss

Keypoints on PTP model messaging

(a) The destination name is "Queue". (FIFO rule)

(b) Both Sender and reciver need not be active at a time

(c) One Message will have only one Reciever/Consumer

(d) The Queue Destination can send message to reciever App who connected to the Destination before Sender sends the message will also the recieve the message when the sender sends the messsage.

(e) if multiple consumer are waiting for a message sent by the SEnder then only first reciver the recieves the message.

(f) The queue destination delivers the message to Reciever App and deletes the message.

usecase:: our car factory usecase needs this model (PTP) communication

feb 04 Messaging using JMS (PUB SUB Example)

How to make Sender of App PTP Model sending message continuosly to MOM software

=> we can take the support of scheduling concept.. For that we need to add @Scheduling annotation on the b.method of Sender App.

note:: @Scheduling can not be applied on the method with args.. So make sure that u r b.method is designed having no args/params.

```
@Component
public class ActiveMQMessageSender    {
@Autowired
private JmsTemplate template;

@Scheduled(cron="*/10 * * * * *")
public void sendMessage() {
//using LAMDA style anonyomous inner class
 template.send("testmq1",ses-> ses.createTextMessage("From sender at"+new Date()));
 System.out.println("Message sent");
 }//run
}//class
```

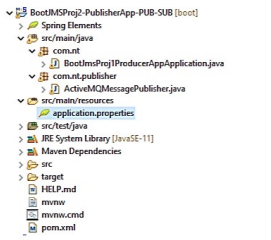
note:: while developing sender and Reciver Apps .. placing @EnableJms on the top of main class is optional.

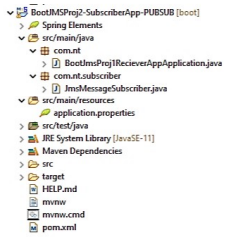
Developing ActiveMQ Pub-sub model App

=>All are same in both Sender/Publisher App and also in Reciever/Subscriber App but change

in application.properties

```
spring.jms.pub-sub-domain=true
```





Order of execution

=>Run subscriber App for multiple times (more than 1 time) (To give the feel more subscribers)
=>Run publisher App
=> Check the console windows of subscribers application

(web console screen /page of active MQ)

Keypoints on pub sub model

(a) One message published by publisher can be consumed by more than one subscriber
(b) The subscribers must done their subscription before publisher publishes the message
(c) The message published by publisher will be duplicated and will be delivered to multuiple subscribers
(d) The Publisher and Subscribers all must be in active mode at a time.
(e) In this model the Destination name is Topic.
(f) Once the subscriber consums the message the message from Topic destination of MOM s/w will not be deleted.

Can we send java object data over the network ?

a) yes , by making the object as Serializable object.

note: Serializing object means the converting object data into stream of bits -bytes that are required to send data over the network or to write to destination file.

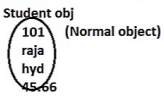
How does normal object become Serilizable object?

=> By making the class of the object as the Serializable class .. (class must implement java.io.Serializable(I))

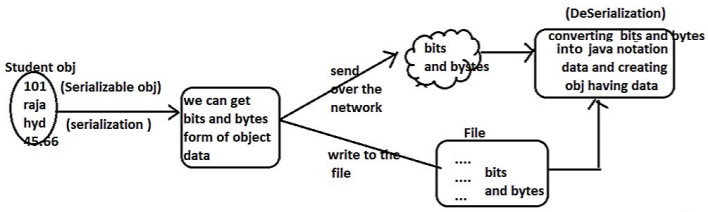
java.io.Serializable(I) is empty interface (marker interface) then how does it makes its impl class object as the Serializable object?

Ans) Markers Interface does nothing directly .. By seeing the marker interface implementation the underlying JVM /Server /Framework /Container provides special runtime capabilities to implementation class objs they create.

=> if JVM is create object for normal class .. that object data is in java format and can not be converted to stream of bits and bytes i.e we can not send/write normal object over the network or to a file.



=> if JVM is creating object for Serializable class (class implementing java.io.Serializable(I)) then the JVM provides capability to the object to convert its data as stream of bits and bytes when needed to send the object's data over the network or to write to a file.



=>Most of marker interfaces are empty interface.. but we can not say every empty interface is marker interface.
=> We can develop our own custom marker interface .. but we need to create custom Container to provider runtime capabilities to the impl class objs of custom marker interface.

other marker interfaces are
=========================
java.io.Serializable(I)
java.lang.Cloneable(I)
java.rmi.Remote(I)
Repository(I) of spring data jpa
java.lang.Runnable (not empty)
and etc..

To decide wheather interface is marker or not ...
do not take emptiness as the criteria.. Take wheather the underlying JVM/container/server/Framework/... providing special runtime capabilities to the impl class object or not.

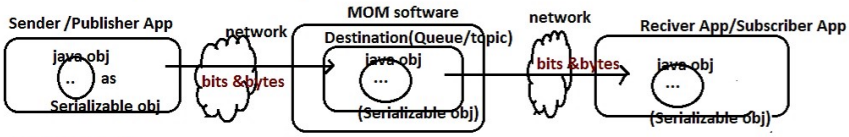
feb 05 Messaging using JMS for sendign Object as Message

==========

Developing ActiveMq PTP application to send Object as message

==========

=> since Sender and Reciever Apps can be there in two different machines of a network or there is possibility of running MOM software on different machine so we need to take the object as Serializable object ..



Sender App code

==========

step1) create spring boot project adding activemq , lombok dependencies

step2) Develop the Java bean class as the Model class.

```
//Model class
============
package com.nt.model;
import java.io.Serializable;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class ActorInfo implements Serializable {
        private  Integer actorId;
        private String actorName;
        private String actorAddrs;

}
```

step3) Develop the Sender App by Enabling Scheduling

```
//Sender class
----------------------

package com.nt.sender;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jms.core.JmsTemplate;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;

import com.nt.model.ActorInfo;

@Component
public class ObjectMessageSender {
    @Autowired
    private JmsTemplate template;

    @Scheduled(cron = "0/20 * * * * *")
    public  void  sendObjectDataAsMessage() {
        //prepare object
         ActorInfo actor=new ActorInfo(1001, "ranveer", "mumbai");
        //send object as the message
         template.convertAndSend("obj_mq1", actor);
         System.out.println("Object is send  as Message ");
    }

}
```

The SimpleMEssageConvertor class contains logic to convert given Serializable object into JMS style Message object.

main class

```
----------
 @SpringBootApplication
 @EnableScheduling
 @EnableJms
 public class BootJmsProj3SendingObjectSenderAppApplication {

      public static void main(String[] args) {
           SpringApplication.run(BootJmsProj3SendingObjectSenderAppApplication.class, args);
      }
 }
```

step4) Add properties in application.properties file

```
 MOM connectivity Details
spring.activemq.broker-url=tcp://localhost:61616
spring.activemq.user=admin
spring.activemq.password=admin

#enable  PTP  communication
# true enables  pub-sub model and false enables  ptp model
spring.jms.pub-sub-domain=false

# make  all packages as the trusted packages (especically model class pkg)
# to send/recieve model class object data as message to MOM software
spring.activemq.packages.trust-all=true
 # or
#spring.activemq.packages.trusted=com.nt.model
```

if we do not place this entry them there is a possibility of getting the following error

This class is not trusted to be serialized as ObjectMessage payload. Please take a look at activemq.apache.org/objectmessage.html for more information on how to configure trusted classes.

step5) Makesure Activem MQ software (MOM software) is in running mode

use E:\ActiveMQSoft\apache-activemq-5.16.3\bin\win64\activemq.bat file

step6) Run the Send App ...

feb 05.1 Messaging using JMS for sendign Object as Message

Reciver App Code

==========

step1) create spring boot project adding activemq , lombok dependencies

step2) Develop the Java bean class as the Model class.

(copy from sender App)

```
//Model class
============
package com.nt.model;
import java.io.Serializable;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
@Data
@NoArgsConstructor
@AllArgsConstructor
public class ActorInfo implements Serializable {
        private Integer actorId;
        private String actorName;
        private String actorAddrs;
}
```

step3) Add properties in application.properties file

```
MOM connectivity Details
spring.activemq.broker-url=tcp://localhost:61616
spring.activemq.user=admin
spring.activemq.password=admin

#enable PTP communication
# true enables pub-sub model and false enables ptp model
spring.jms.pub-sub-domain=false

# make all packages as the trusted packages (especically model class pkg)
# to send/recieve model class object data as message to MOM software
spring.activemq.packages.trust-all=true
 # or
#spring.activemq.packages.trusted=com.nt.model
```

step4) Deveop the ReciverApp as JMSListener...

```
package com.nt.reciever;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jms.annotation.JmsListener;
import org.springframework.jms.core.JmsTemplate;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;
import com.nt.model.ActorInfo;
@Component
public class ObjectMessageReciever {
        @JmsListener(destination = "obj_mq1")
        public void consumeObjectDataAsMessage(ActorInfo actor) {
                System.out.println("Recived Object Data ::"+actor);
        }
}
```

step5) Run the Reciver App ...

Limitation of JMS

---

a) JMS is java language dependent technology i.e we need to develop both sender and reciver App in same java language .. (JMS can not be used outside of Java Domain)

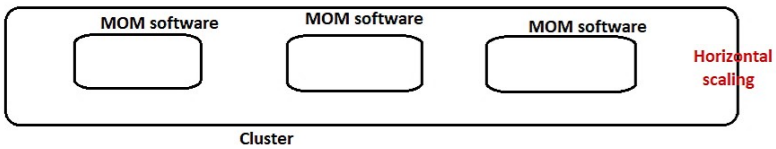
b) JMS based MOM softwares can recieve and send messages only by using protocol TCP i.e we can not use other than protocol TCP like http , smtp and etc..

c) There is a possibilty of loosing data /message if the MOM software is down or MOM software is not responding while publisher or sender is sending the messages

d) if the Message is very big / large scale then MOM software behaves very slow (i.e gives the performance issue)

e) There is no ability of creating multiple instances of single MOM software .. if multiple senders /publishers are sending messages simultaenously the Performance of single copy MOM software may not suitable for industry needs.

(It indirectly say JMS style MOM software does not horizontal scaling.. it supports only vertical scaling)



Conclusion:: use JMS style messaging only when no.of message are less and no.of senders/publishers are less towards sending messages.

feb 06 Apache kafka Installation & working with Flow

Apache Kafka
==========

=> It is java based messaging technique .. which can be implemented in different languges

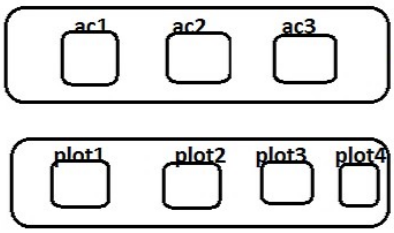
=> It supports to use different technologies and concepts for messaging activity like hadoop, spark, scala and etc..

(kafka integration is possible with multiple technologies of java and non-java env..)

=> Apache kafka is all about Store + Process + Integrate (Transfer)

=>Kafka is basically used to transfer (integrate) data /messsages between multiple complex Apps/systems using Cluster design.

cluster :: set of similar items is called cluster.



Ex:-like connection pool

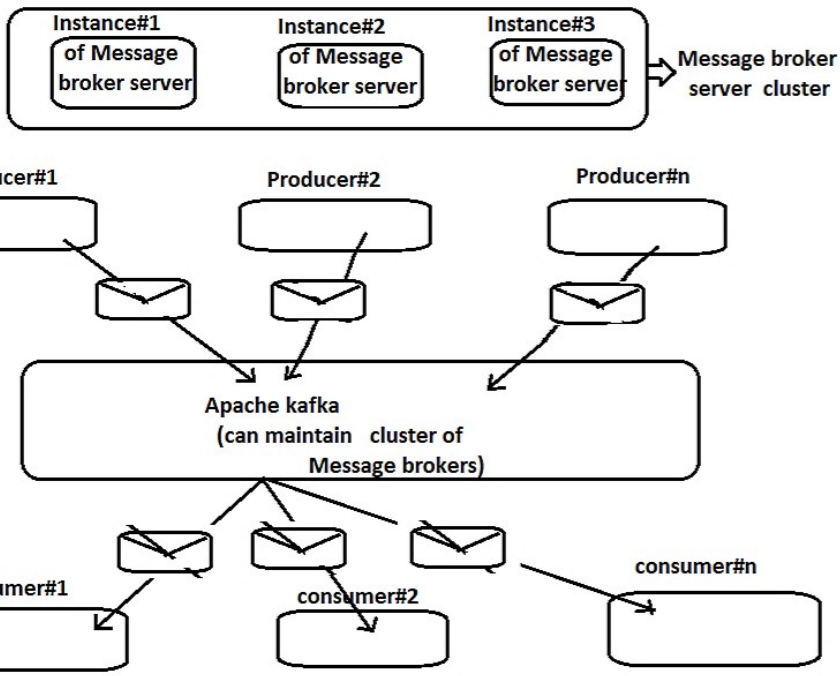
=>Kafka intergration /interaction with non-java applications is possible with the support of REST calls.

=> Karfka is protocol Independent ... we can write code to send messages among the applications using http ,tcp ,ftp and etc .. protocols.

=> Kafka allows to take multiple message broker s/ws (MOM s/ws) at a time as cluster having support for Horizontal scaling giving the following advantages..

a) Fast data transfer for larget data set /messagess
b) No data loss even one messsage broker s/w or MOM s/w is down becoz other broker s/w or MOM s/w takes care of messages.
c) It takes the support of apache zookeeper to handle load balancing among the multiple instances of message broker s/w or MOM s/w

apache zookeepr is like netflix eureka server..



-

feb 06.1 kafka Intro

Kafka Message Broker / Broker server / Mom Server (MOM server is not official word)

==========

It behaves like MOM/Broker to recieve messages/data from sender, to hold them as needed and to deliver to Consumer ..

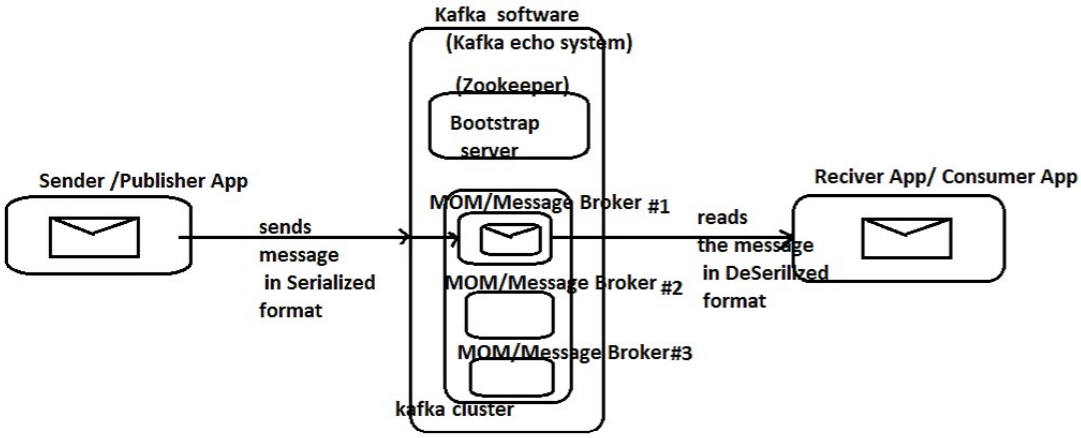
=>When kakfa s/w is started one instance of Message broker will be created automatically.. and it can be increased as needed.

=> collection message broker instances togather is called kafka message broker cluster .. and we can add any no.of instances in one cluster .. i.e there is no limit to add instances.

=>Apache Zookeepr is resposible to manage message broker instances of cluster by applying Load Balance support and it is also called Bootstrap server becoz it is responsbile to create cluster having single instance and increasing the instances as needed.

=>Apache kafka Eco System = (Zookeeper) Boostrap server + message broker cluster.

=> The apache kafka Message broker gets Message from Sender /Publisher App in Serialized format and delivers to consumer /subscriber in DeSerialized format.



=> kafka do not support PTP model messaging i.e the message broker/MOM can not have Queue as the Destination

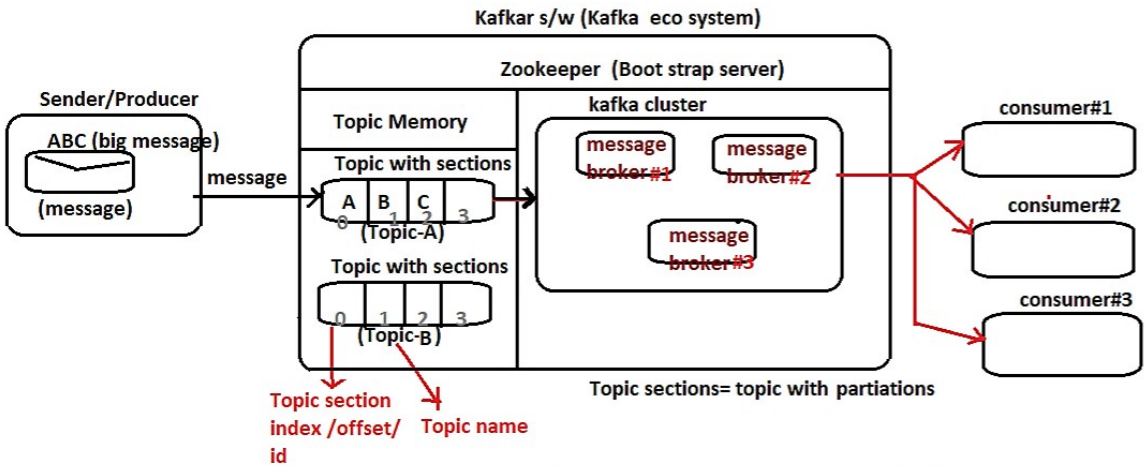
=> kafka supports only Pubsub mode messaging i.e the message broker /MOM can have only Topic as the destination.

=> To send message only to one consumer .. we take the support of topic Destination.

=>For one message one topic destination (Memory) is created/allocated ..that stores given data /message in partiations (huge messsage will be divied into parts/packets). Every paritiation of the message is indentified with an index like 0,1 ,2,3 .. These indexes are technically called as offsets.

=> In kafka messaging , For one consimer one message broker instance will be allocated at a time.
So to send 1 message to 5 consumers we need
1 Sender ---> 1 message --> 1 Topic with sections/partiations --> 5 message broker instanaces -- 5 cosumers

=> At a time , the Message broker reads one paritiation data , takes the data , replicates data (cloned data) and sends to consumer i.e each large message/data will be stored in multiple partiations of topic and the messge broker reads data from all partiations and sends to its respective consumer.



=>Each message broker instance is capable of reading message from multiple sections/partiations of Topic destinations... replicates the message parts -- merges the message parts --sends the message to respective consumer.

feb 06.2 kafka intro

Apache Kafka terminologies

Message Broker /MOM :: The mediator /Middle man that will transfer message to consumer by reading message/data from Topic /Topic partiotions

kafka cluster :: collection /group of message broker instances which are created based cosumers count is called kafka cluster.

Topic :: The memory that holds message sent by the Producer in parts

Producer :: The App/comp that sends the message to Topic Memory in Serialized format

Consumer :: The App/comp that reads the message from Topic Memory through Message Broker in DeSerialized format

offset/index :: The id or index given to partiation message/data in Topic

Zookeeper /BootstapServer :: Handles Message Broker Cluster and Topic Memory .. While managing the Message broker cluster it will do Load Balancing.

Replica-Factor :: The no.of cloned/duplicate messages that should be created in order to send the messages to Consumers through message brokers ..

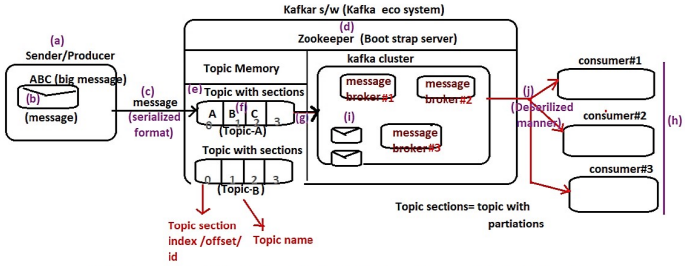
Generally it is decided based consumer count.

=>if there are 10 cosumers then Replica-factor is 1/10 (For 1 message 10 clonnded copies are required)

==========

Apache kafka Work Flow Diagram

==========



=>Each message broker instance is capable of reading message from multiple sections/partiations of Topic destinations... replicates the message parts -- merges the message parts --sends the message to respective consumer.

(a) Define one Producer App

(b) Create either Static message or dynamic message that can be given at runtime

(c) Send message from Producer in the form fo key=value pair format

here "key" is topic name (With the given name topic available then uses it other wise new topic will be created)

"value" is the message/Data

note:: Producer App sends this data/message in Serilized format.

(d) Start zoo keeper (Boot starp server)

(e) Topic Section will be created in Topic Memory to read the message sent by the producer App..

note: New Topic section will be created (if not already available) otherwise it will use the existing Topicsection

(f) Topic section reads message and stores message in different partiations of Topic which will be created as needed.

(note:: These partiations are identified with indexes /offsets)

(g) creates link b/w Topic section/Topic and Message broker

(h) Define one or more consumer Apps as needed by specifying topic name

(note:: Based on the given Topic name in cosumer App(s) the link b/w Topic , message broker and Cosumer App will be created as needed)

(note:: The zookeeper takes the responsbility of creating messagebroker instances as cluster based on the Consumer count and takes care of LB -Load Balancing)

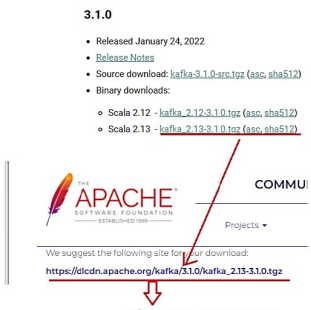
(i) The Message Broker(s) reads the data/messages from Topic Partiations creates cloned/replicate copies of these partitaion messages.

j) The message broker(s) sends the cloned partiation messages to one or more consumer(s) part by part.

Arranging Apache Kafka Software

==========

Go to https://kafka.apache.org/downloads



gives kafka_2.13-3.1.0.tgz

==> Copy this file (E: drive) and extract it

Sending Message using apache kafka infrastructure

==========

step1) start Zookeeper as Boot strap server..

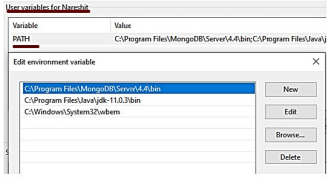
```
E:\kafka_2.13-3.1.0\bin\windows>zookeeper-server-start E:\kafka_2.13-3.1.0\config\zookeeper.properties
```

command as .bat file (input value)

step2) start apache kakfa s/w setup

i) make sure that "C:\Windows\System32\wbem" is added PATH env.. variable (User Variable)

Mycomputer ---> proeprties ----> advanced system settings --->



env.. variables ---> user vairables

-> new --->

variable name :: PATH

value :: C:\Windows\System32\wbem

ok --->ok --->ok

ii) E:\kafka_2.13-3.1.0\bin\windows>kafka-server-start E:\kafka_2.13-3.1.0\config\server.properties

step3) Create new topic specifying reflection factor (no.of copies) , partintiations count (offset count)

```
E:\kafka_2.13-3.1.0\bin\windows>kafka-topics --create --bootstrap-server localhost: 9092
                --replication-factor 1 --partitions 1 --topic nit-tpc
```

step4) Create a producer and link with message broker with support of bootstrap-server

```
E:\kafka_2.13-3.1.0\bin\windows>kafka-console-producer --bootstrap-server localhost:9092 --topic nit-tpc hello
>hai
>123
```

send these messages after starting the consumer

messages

step5) create a consumer and link with message broker

```
E:\kafka_2.13-3.1.0\bin\windows>kafka-console-consumer.bat --bootstrap-server
                localhost:9092 --topic nit-tpc
hello
hai
123
```

feb 09 developing Kafka Clients Using Java APIs

Developing Kafka Client Apps using Java API

=> Both Producer /Sender and reciver/Consumer Apps that are talking to kafka software are called kafka Client Apps.

Producer /Sender App Development (Legacy Style - spring style)

=> First we need to establish a link between Producer App and kafkat setup (mainly bookstrap server) by using the following details /properties

#1 bootstrap-server = localhost:9092 (Kafka Bootstarp server default port is 9092)
key-serializer = pkg.StrngSerializer we generally keep this info in Properties class obj
value-serializer = pkg.StringDeSerializer we develope app in legacy style (not spring boot style)

note1:: any type of data /message given by Producer will be converted to String message/data using StringSerializer class that is specified above. if we give java object as message/ data then it will be converted into json String content .

=> Take the support of "ProducerRecord" class to sepcify Message object details and topic name

#2 ProduderRecord<String,String> record=new ProducerRecord("topicname", message object);

(ProducerRecord class obj indirectly represeing message/data to send from producer App to topic of kafks setup)

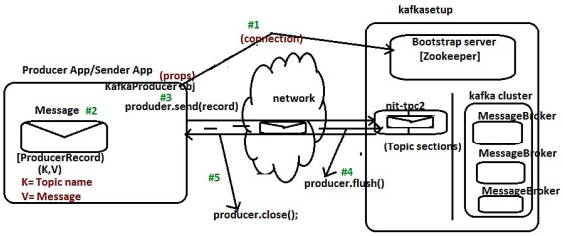
=> To send Message (ProducerRecord) created in the Producer App we need to use KafkaProducer<String,String> class as shown below

#3 KafkaProducer<String,String> producer=new KafkaProducer<>(props);

producer.send(record); //puts data in connection stream in queue

#4 producer.flush(); //sends to data Topic based on the given topic name

#5 producer.close(); // Closes the link b/w producer and kafks setup (Bootstarp server)

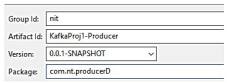


Code development and execution

step1) create maven project (not sprng boot starter Project) using maven-archetype-quickstart

adding the following dependencies (kafkaclients , slf4j-simple , jackson-dataformat-xml)

File menu ---> maven project ---> next --->
select maven-archetype-quickstart --->next -->



In pom.xml

```
<!-- https://mvnrepository.com/artifact/org.apache.kafka/kafka-clients -->
<dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>3.1.0</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.slf4j/slf4j-simple -->
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-simple</artifactId>
    <version>1.7.35</version>
    <scope>test</scope>
</dependency>

<!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.dataformat/jackson-dataformat-xml -->
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
    <version>2.13.1</version>
</dependency>
```

step2) Develop the Producer App

```
package com.nt.producer;
import java.util.Properties;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.common.serialization.StringSerializer;
public class MEssageProducer {

    public static void main(String[] args) {
        // create Connection properties as K=V in java.util.Properties class obj
        Properties props=new Properties();
        props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
        props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,StringSerializer.class.getName());
        props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,StringSerializer.class.getName());
        //create KafkaProducer object
        KafkaProducer<String, String> producer=new KafkaProducer<String, String>(props);
        //create ProducerRecord object representing the message
        String msg="WELcome to Apache kafkas messaging";
        String topicName="nit-tpc-feb";
        ProducerRecord<String, String> record=new ProducerRecord<String, String>(topicName, msg);
        // Send message (record)
        producer.send(record);
        //flush the message
        producer.flush();
        //close the connection with BootStrap server
        producer.close();
        System.out.println("message sent");
    }
}
```

step3) Execute things in the following order

a) start zookeeper b) start kafka setup c) create topic d) Run the above Producer App
e)start Consumer (readmade)

a)start zookeeper

```
E:\kafka_2.13-3.1.0\bin\windows>zookeeper-server-start.bat E:\kafka_2.13-3.1.0\config\zookeeper.properties
```

b) start kafka setup

```
E:\kafka_2.13-3.1.0\bin\windows>kafka-server-start.bat E:\kafka_2.13-3.1.0\config\server.properties
```

c) create Topic and topic sections

```
E:\kafka_2.13-3.1.0\bin\windows>kafka-topics --create --bootstrap-server localhost:9092
                                --topic nit-tpc-feb --replication-factor 1 --partitions 1
Created topic nit-tpc-feb.
```

d) start the consumer App

```
E:\kafka_2.13-3.1.0\bin\windows>kafka-console-consumer --bootstrap-server localhost:9092 --topic nit-tpc-feb
```

WELcome to Apache kafkas messaging

e) run producer App from eclise ide sends message

feb 10 Developing Kafka Clients using Java APIs

Developing kafka consumer as kafka client in legacy style (spring style)

==========

#1] We need to create communication link b/w kafka setup and consumer(s) by supplying multiple details connection details as key=value pairs in the form of Properties object

bootstrap-server = localhost:9092
key-deserializer = StringDeSerializer
value-deserilizer = StringDeSerializer
(given by kafka api)
groupid = grp-listeners [anything can be given here] [optional]

=> if multiple consumers are taken reading same message from topic thorugh message broker then grouping multiple consumers to single group by providing groupid make replicate /cloning operation on messages faster.

#2] create KafkaConsumer class object specifying the above connection properties to get link /connection between consumer and kafka setup (Indirectly with Bootstarp server (zookeeper) that manages Topic memories and message brokers.

```
KafkaConsumer<String,String> consumer=new KakfaConsumer<>(props);
```

java.util.Properties class obj having the above connection properties

#3] Link Consumer with MessageBroker by specifying the topic name
[Here MessageBroker will be created dynamically on 1 per Consumer basis)

```
consumer.subscribe(Arrays.asList("nit-tpc-feb"));   // we given multiple topic names
                                                    // to read messages from muliple topic sections
```

#4] Do polling (pinging the message broker continuosly having certain gap) wth Message broker to check message(s) avaiable or not and read message. (indirectly every second tracking)

Expected process ::

On arrival of message to topic memory , the Message broker reads the messages and sends the message to all the subscribed consumers automatically .. but In legacy style kafka consumer devleopment that is not happening . So polling has to be done

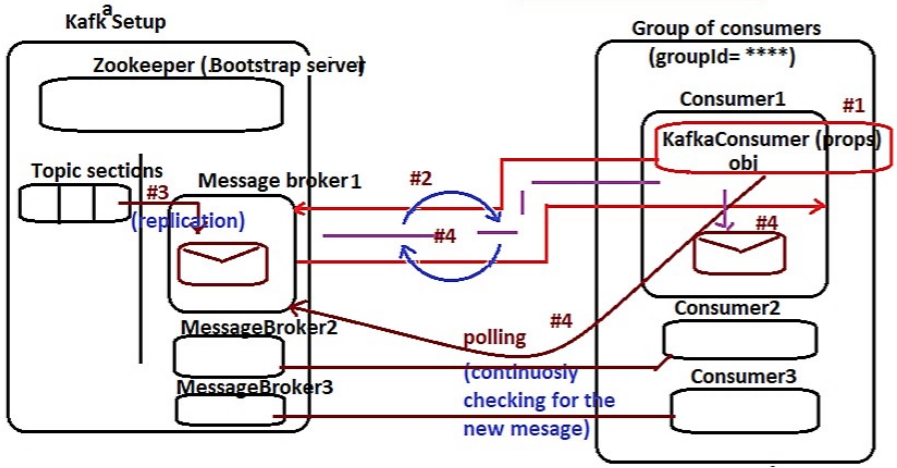
Actual process

==========

=> Consumer pings the Message Broker having scheduling (having certain continuos time gap) for new message .. if came to topic memory then the message broker gives that message to consumer (this process is polling)

```
while(true){
   ConsumerRecords<String,Stirng> records=
                    consumer.poll(Duration.ofMillis(1000));

    for(ConsumerRecord<String,String> record: records){
        System.out.println("message is ::"+record.value());
    }//for
  } //while
```

=>Each consumed message will be represented by ConsumerRecord object
Multiple comsumed messages will be represented by ConsumerRecords object.



=> Each messsage comes as the Consumer Record obj

feb 10.1 Developing Kafka Clients using Java APIs

Code (Kafka Consumer Code)

==========

step1) create maven project adding the following dependencies

kafka clients , slf4j-simple , jacson-data format-xml (same producer App)

```
<!-- https://mvnrepository.com/artifact/org.apache.kafka/kafka-clients -->
<dependency>
  <groupId>org.apache.kafka</groupId>
  <artifactId>kafka-clients</artifactId>
  <version>3.1.0</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.slf4j/slf4j-simple -->
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-simple</artifactId>
  <version>1.7.35</version>
  <scope>test</scope>
</dependency>

<!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.dataformat/jackson-dataformat-xml -->
<dependency>
  <groupId>com.fasterxml.jackson.dataformat</groupId>
  <artifactId>jackson-dataformat-xml</artifactId>
  <version>2.13.1</version>
</dependency>
```

step2) Develop the Consumer App

MessageConsumer.java

```
package com.nt.consumer;

import java.time.Duration;
import java.util.Arrays;
import java.util.Properties;

import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.common.serialization.StringDeserializer;

public class MessageConsumer {

 public static void main(String[] args) {
 // create Connection properties as K=V in java.util.Properties class obj
 Properties props=new Properties();
 props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
 props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,StringDeserializer.class.getName());
 props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,StringDeserializer.class.getName());
 props.put(ConsumerConfig.GROUP_ID_CONFIG,"grp1_consumers");
 //create KafkaConsumer object
 KafkaConsumer<String, String> consumer=new KafkaConsumer<String, String>(props);
 //Subscribe to topic Destination through MEssagebroker
   consumer.subscribe(Arrays.asList("nit-tpc-feb"));
  //Performing polling to check and read the messages
   while(true) {
        //poll and get consumer records (messages)
       ConsumerRecords<String,String> records=consumer.poll(Duration.ofMillis(2000));
        // read and display messages
          for(ConsumerRecord<String,String> record:records) {
                System.out.println("message is ::"+record.value());
          }//for
    }//while
    }
    }
```

step3) **Run The services in the following order**

a) start zookeeper as boostrap server

```
E:\kafka_2.13-3.1.0\bin\windows>zookeeper-server-start.bat
                                        E:\kafka_2.13-3.1.0\config\zookeeper.properties
```

b) start apache kafka setup

```
E:\kafka_2.13-3.1.0\bin\windows>kafka-server-start.bat
                                        E:\kafka_2.13-3.1.0\config\server.properties
```

c) note:: since Topic name "nit-tpc-feb" is already created.. So we need not to create again

d) Run the consumer App using eclipse

e) Run the Producer App using eclipse (previous class App)

feb 12 Developing Kafka Clients using spring boot kafka apis

Spring Boot + Apache Kafka API
==========

=> Spring Boot gives built-in support for apache kafka.. It even given certain objects automatically through AutoConfiguration process

=> if spring -boot- starter-apache-kafka depenecies to app then we get KafkaTemplate<K,V> class object through autoConfiguration..
which internally takes care of creating KafkaProducer, ProducerRecord objects that are required to send messages/data.

In Producer class or comp

```
@Autowired
private KafkaTemplate<String,String> template;
```

=> we can place @KafkaListener(topicName="....." , groupId=".....") annotation on the method Listener class to make Message broker to collect the message recived to topic section i.e it internally takes care of creating KafkaConsumer<String,String> obj and ConsumerRecord object.

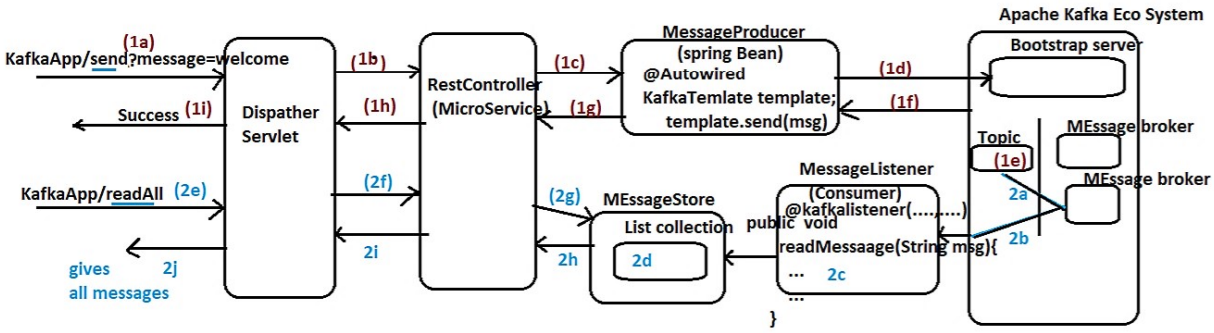
In Listener class

```
@KafkaListener(topicName="nit-tpc-fri", groupId="nit-grp1")
public void readMessage(String msg){
    ....
    .... //logic to read the message.
  }
```

=> We need to add @EnableKafka on the top of starter class (main class)

=> we can get message from endusers as request parameter through RestController/MicorService of spring MVC/Spring Rest App , So we need to make the Producer App taking message from through RestController to send kafka setup. Similarly we need to read the message from kafka setup using Consumer or MEssageListener to send to endusers through RestController.

http://localhost:4041/KafkaApp/send?message=welcome
http://localhost:4041/KafkaApp/send?message=quotation value 2000
http://localhost:4041/KafkaApp/send?message=how are u



(1a) --- 1i :: 1st request -response (message came and stored in topic section of apache kafka server from producer successfully)
(2a) --- 2j :: 2nd request-response (message sent to consumer from topic section through broker of apache kafka server successfully)

order of development
==========

=>MessageProducer having injection KafkaTemplate obj
=> Restcontroller with handler method with "/send" request path having injection of MessageProducer object
=> MessageStore
=>MessageListner injected with MessageStore
=> above Restcontroller with another handler method with "/readAll" request path having injection of MessageStore object

step1) Create spring boot starter Project adding the following dependencies web , kafka , lombok api, devtools

step2) add @Enablekafka on the top of main class/starter class.

```
@SpringBootApplication
@EnableKafka
public class BootKafkaProj2RestWithKafkaApplication {

    public static void main(String[] args) {
        SpringApplication.run(BootKafkaProj2RestWithKafkaApplication.class, args);
    }
}
```

step3) add the following properties in application.properties file

applicaiton.properties

```
#server port
server.port=4041

#context path
server.servlet.context-path=/RestKafkaApp
#topic name
app.topic.name=nit-tpc-sat1

#Producer properites
spring.kafka.producer.bootstrap-servers=localhost:9092
spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer
spring.kafka.producer.value-serializer=org.apache.kafka.common.serialization.StringSerializer
#Consumer properites
spring.kafka.consumer.bootstrap-servers=localhost:9092
spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringDeserializer
spring.kafka.consumer.value-deserializer=org.apache.kafka.common.serialization.StringDeserializer
```

feb 12.1 Developing Kafka Clients using spring boot kafka apis

step4) MessageProducer.java

```
package com.nt.producer;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.stereotype.Component;

@Component("msgProducer")
public class MessageProducer {
        @Autowired
        private KafkaTemplate<String,String>  template;
        @Value("${app.topic.name}")
        private  String topicName;

        public  String   sendMessage(String message) {
                template.send(topicName,message);
                return " message delivered ";
        }
}
```

step5) Restcontroller

```
package com.nt.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import com.nt.consumer.MessageStore;
import com.nt.producer.MessageProducer;

@RestController
public class KafkaMessageHandlingController {
@Autowired
private  MessageProducer producer;
@Autowired
private MessageStore  store;

@GetMapping("/send")
public  String   sendMessage(@RequestParam("message")String message) {
          String status=producer.sendMessage(message);
          return "<h1>"+status+"</h1>";
}
@GetMapping("/readAll")
public   String   fetchAllMessage() {
   return  "<h1>"+store.getAllMessages()+"</h1>";
}
}
```

step6) MessageStore.java

```
package com.nt.consumer;

import java.util.ArrayList;
import java.util.List;
import org.springframework.stereotype.Component;
@Component
public class MessageStore {
        private List<String> listMessages=new ArrayList();
        public void   addMessage(String message) {
                listMessages.add(message);
        }
        public  String   getAllMessages() {
                return  listMessages.toString();
        }
 }
```

step7) MessageConsumer.java

```
package com.nt.consumer;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.stereotype.Component;
@Component
public class MessageConsumer {
        @Autowired
        private MessageStore store;
        @KafkaListener(topics = "${app.topic.name}",groupId = "grp1")
        public void  readMessage(String message) {
                //add  message to store
                store.addMessage(message);
        }
 }
```

Execution order

i) start bootstrap server (zookeeper)

```
E:\kafka_2.13-3.1.0\bin\windows>zookeeper-server-start.bat E:\kafka_2.13-3.1.0\config\zookeeper.properties
```

ii) start kafka server setup

```
E:\kafka_2.13-3.1.0\bin\windows>kafka-server-start.bat E:\kafka_2.13-3.1.0\config\server.properties
```

note:: No need of creating topic seperately .. @EnableKakfa will take care of creating topic dynamically

iii) Run the application as spring boot App or on server
and give requests

```
http://localhost:4041/RestkafkaApp/send?messsage=raja
http://localhost:4041/RestkafkaApp/send?messsage=rani
http://localhost:4041/RestkafkaApp/send?messsage=hello
http://localhost:4041/RestkafkaApp/readAll
 [raja,rani,hello]
```

feb 15 Distributed Tracking&Logging Using slueth -zipkin

Distributed Logging and Tracing

Tracing :: Finding out execution flow /path from request to response is called Tracing

Logging :: Finding out various lines of code and various comps that are involved in the flow of execution having different Logger Levels like DEBUG,INFO,WARN and etc.. is logging

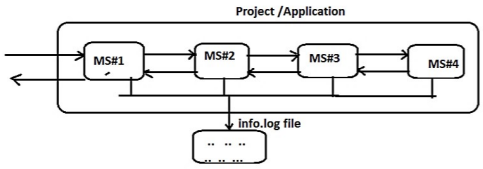
Intra communication of MicroServices

=>It speaks about the communication that happend between multiple microseervices.
=>The request given to one MS taking to another MS is called Intra communication b/w MicroServices.

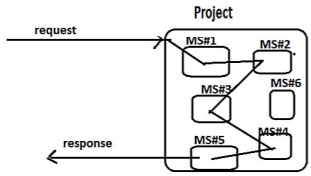
Distributed Logging and Tracing

=>When the MicroServices are in intra communication enabling tracing and logging activities across the multiple MicroServices to findout exection flow of each and every request to response called Distributed Tracing and Logging.

=>This will enabled on multiple MicroServices for one time to help new developers/ testers /debuggers /UAT team to know execution flow from request to response across the multiple MicroServices for ever.



=> Logging is extension of Tracing .. In fact logging a kind of tracing activilty .. The only difference is the logging activity writes its log messages to destination called file/Db/mailserver and etc. having ability to filter the messages based on the Logger Levels we have choosen.



if execution is taking place across the multiple MicroSerivces then we need the support of Distributed Logging and Tracking.

Slueth and Zipkin : These two are components/ tools provided by the spring cloud env.. to enable distributed Logging and tracing on MicroSerivces that are in Intra communication.

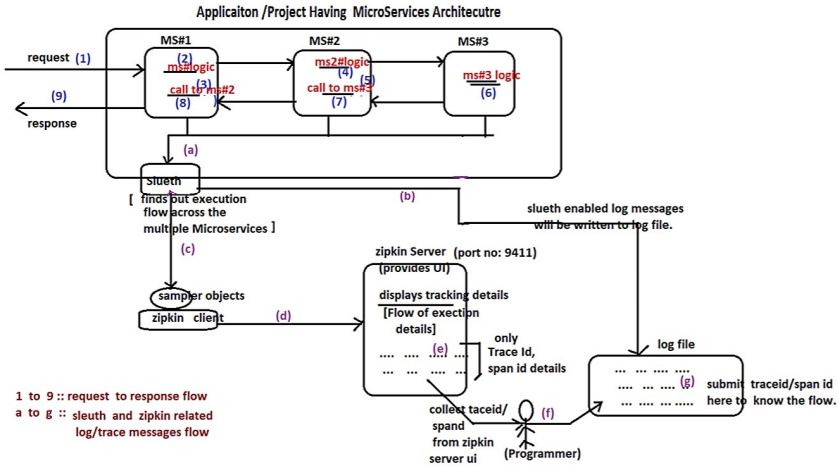
Slueth :: A spring cloud comp providing unique id for each request flow ... The Programmer can use this unique ids to find out the execution flows of requests.

Two types of Ids

traced Id :: unique id for each request flow across the multiple micro services.. if developers gets this traced id he can find out all the microservices that are involved in a request flow.

span id :: Unique id given for each micro service .. if developer gets this span id then he can findout all the executions that happend in a micro service.

Zipkin Client :: We need to add Zipkin dependency to every MS along with sleuth dependency .. It contains sampler obj nothig but the data collected from MicroServices using Slueth tool to Zipkin Server) So it can provide lots of details related trace id , spanid and etc.. with respect to the Distributed loggig and tracking enabled on across the multiple microservices.



Zipkin Server :: provides UI env.. running on the port number (9411 default) The zipkin clients of each Ms collects trace ids/spand ids of distributed tracking, logging from slueth comp and gives to zipkin server to display having UI.

note:: while working with Distribued Logging and tracing each MS contains its own zipclient and slueth comp..but there must be only one zipkin server as centralized server that collects data from all zipkin clients of different MS as sampler objs to display as UI.

note :: The develop /programmer collects trace id/span id from zipkin server ui and seraches in the distributed log file generated by the slueth to find out log messages for flow of execution.

Q) In One Project of MicroServices architecture , can u tell me how many micros serivces will be there?
ans) Mulitple as needed

Q) What is microServices Intra communication
ans) It is the communication b/w microservices (One Ms to another MS communication)

Q) In MicroSerivces Intra Communication , how can we find out which comps of which microservices are executed in the request flow?
Ans) We need enabled distributed logging / tracing on multiple Microservices that are in intr Communication.
(In this process every MS should have slueth and zipkin client support connected to the centralized zipkin server)

Q) When to use log4j and when to use slueth and zipkin?
Ans) log4j is required to write log message in both indepndent Microservices execution and the Intra communication enabled microservices execution .. But we link log4j with slueth and zipkin to write the log4 generted log mesage having trace ids and span ids which helps distributed logging and tracing.

Keeping Zipkin Server ready

step1) download jar file that represents zipkin server

https://zipkin.io/pages/quickstart ----> go to java section --->
click on latestrelease which gives "zipkin-server-2.23.16-exec.jar" representing zipkin server.

step2) start zipkin server ..

copy "zipkin-server-2.23.16-exec.jar" file to u r choice and execute the jar file

```
E:\zipkinserver>java -jar zipkin-server-2.23.16-exec.jar
```

step3) open zinkin server home page

http://localhost:9411/zipkin/

feb 16 Distributed Tracking&Logging Using slueth -zipkin

Example App

==========

=>Keep Zipkin server running mode

=> develop 3 microservices having intra communication + zipkin client and slueth support and execute them

=> Give request to First MicroService

=> perform the following operations in zipkin server console

a) open the home page ::http://localhost:9411/zipkin

b) Click on FindTrace

c) Click on BlueColor Bar

d) Click any One Option (App1 --First MicroSerivce name)

e) collect and copy the TraceId

f) open the log file (App.log) and serach for (ctrl+f) traceId to see all the log messages realated to current request

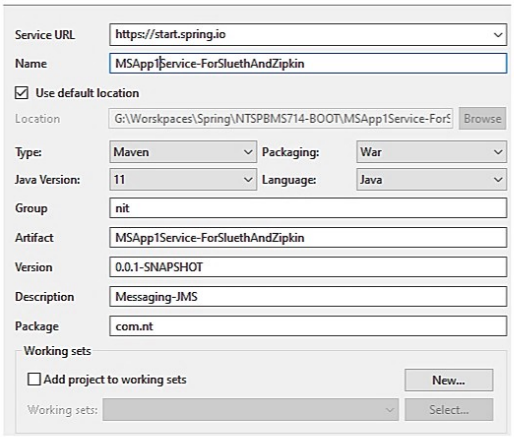
MicrServices Development

==========

First MicroService (App1)

---

a) create spring boot starter project of type war file adding web,lombok , slueth, zipkin client dependencies



for this app 2.5.9 parent starter is used

2.6.x is not working

add these starters

b) Add the following entries in application.properties

application.properties

```
server.port=9091
spring.application.name=App1
logging.file.name=E:/logs/App.log
```

c) create objects to make them as spring beans using @Bean methods in @Configuration class

AppConfig.java

==========

```
package com.nt.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.client.RestTemplate;
import brave.sampler.Sampler;
@Configuration
public class AppConfig {
	@Bean
	public  Sampler  createSampler() {
		return  Sampler.ALWAYS_SAMPLE;
	}
	@Bean
	public  RestTemplate  createRestTemplate() {
		return new  RestTemplate();
	}
}
```

d) Develop RestController as MicroService

```
package com.nt.controller;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;
@RestController
public class ShoppingOperationsController {
	@Autowired
	private  RestTemplate  template;
	Logger logger=LoggerFactory.getLogger(ShoppingOperationsController.class);

	@GetMapping("/shopping")
	public String Shopping() {
		logger.info("Welcome to shopping Module");
		//communicate with  BillingSErvice
		String  resp=template.getForObject("http://localhost:9092/billing", String.class);
		logger.info("Back to  shopping module::"+resp);
		return  resp;
	}
}
```

feb 16.1 Distributed Tracking&Logging Using slueth -zipkin

note: Develop Second MicroServcie and Third MicroSerivce in similar fashion having neccessary chanages
note:: Second MS communicates with Third Ms .. but Third MS does not interact with any Ms
note:: Registering these MicroServices with Eureka is optional .. but recomanded to do..

Second MicrService
==========
=>Project creation :: same as first mS
=>AppConfig.java :: same as first MS
=>application.properties

```
server.port=9092
spring.application.name=App2
logging.file.name=E:/logs/App.log
```

=>Rest Controller class

```
package com.nt.controller;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;
@RestController
public class BillingOperationsController {
        @Autowired
        private RestTemplate template;
        Logger logger=LoggerFactory.getLogger(BillingOperationsController.class);
        @GetMapping("/billing")
        public String doBilling() {
                logger.info("Welcome to Billing Module");
                //communicate with PaymentSErvice
                String resp=template.getForObject("http://localhost:9093/payment", String.class);
                logger.info("Back to Billing module::"+resp);
                return resp;
        }
}
```

Third MicroService Development
==========
=>Project creation :: same as first mS
=>AppConfig.java :: same as first MS
=>application.properties

```
server.port=9093
spring.application.name=App3
logging.file.name=E:/logs/App.log
```

RestController class

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
@RestController
public class PaymentOperationsController {
        Logger logger=LoggerFactory.getLogger(PaymentOperationsController.class);
        @GetMapping("/payment")
        public String doBilling() {
                logger.info("Welcome to payment Module");
                return "payment is done";
        }
}
```

To run the Application
==========

=>Keep Zipkin server running mode

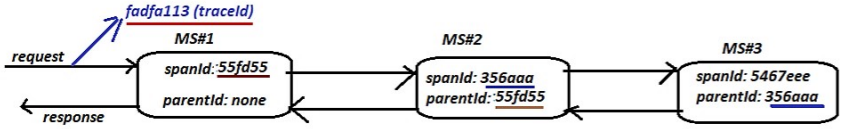
```
E:\zipkinserver>java -jar  zipkin-server-2.23.16-exec.jar
```

=> develop 3 microservices having intra communication + zipkin client and slueth support and execute them
order of execuion :: 3rd , 2nd and 1st

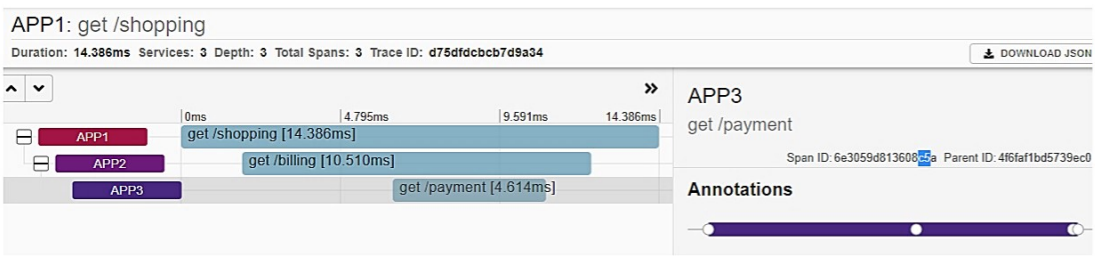
=> Give request to First MicroService

http://localhost:9091/shopping

=> perform the following operations in zipkin server console *+ log file*
a) open the home page ::http://localhost:9411/zipkin
b) Click on FindTrace
c) Click on BlueColor Bar *(Run query)*
d) Click any One Option (App1 --First MicroSerivce name)
e) collect and copy the TraceId
f) open the log file (App.log) and serach for (ctrl+f) traceId to see all the log messages realated to current request



spanId current MS = parent Id of next MS (nothing but current MS spanId will becomes parent id for next MS)



feb 17 API Gateway -zuul

API -Gateway

=> Different MicroServices of project run on different port numbers having different urls .. it is pratically impossible to remember all those port numbers and urls seperately.. So we need single entry and exit point having unique url for all the micro services of the App ..that is API gateway

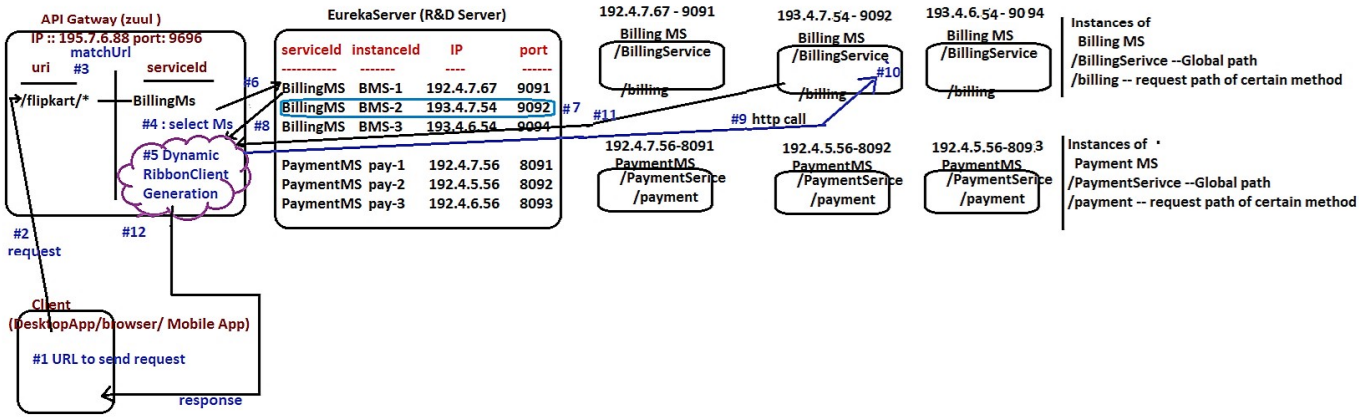
API Gateway provides
==========
=> Acts as single entry and exit point for the application (For all the micro services of the application)
=> Can perform Authentication and Authorization (Security)
=> Provides Filters for Data /request logging (To find out flow related to request/response/error)
=> Dynamic Routing to the instances of MicroServices (Intenrally uses Ribbon Client Code (Proxy Client code)for this)
=> API Gateways also one kind of MicroService Which is able to call all other MicroSErvices of the application or Project using Eureka s erver.

note:: Eureka Server is for registering microService and for finding MicroService ...
Eureka server itself can not communicate with other MicroService in any angel .One MS gets details of other MS from EurekaServer using one or another client code (like LBC, Feign Client and etc..) later uses RestTemplate support to make http call from One MS(source) to another Ms(Destination).

conclusion :: Eureka Server can not communicate with Ms and can not make http calls to interact with any Ms .. It Is just there to register details of One MS, So other MS can use one or another Client Code to find the details about MS(dest) and to interact with that MS (dest)

What is API Gateway ? what is use of it?

=> It provides single entry and exit point all for MicroServices of the Application/ Project.. It itself acts as MicroService having ability to take http requests from Clients and to communicate with other MicroSErvices using http calls through EurekaServer by Generating RibbonClient Code as Proxy code.



URL in the client App
==========
syn :: http://<zuulIP>: <port>/<zuulpath>/<MsGlobalPath>/<Ms Method or req path>
eg:: http:// 195.7. 6.88:9696 / flipkart/BillingService/billing

=> Zuul (API gateway) is also one MS , registering with Eureka
=> Zull Generatoes Client Code to interact with EurekaServer Dyanically as InMemory Proxy class in the form RibbonClient
=> Zuul interacts with Eureka Server using Dynamic Client Code to get Less Load instance (load balance)
=> The dyanamically generated Client code in Zuul makes http request to communicate with MS whose instanace is gathered from EurekaServer .. later it sends the recieved output to Client as response

With respect to the diagram
==========
#1 :: enduser give URL to Client App/Browser to send request
#2 :: The url based generated request goes to Zuul (API gateway)
#3 :: Zuul matches/compares current request url with common paths/zuul paths that are maintained by linking with serviceIds
#4 :: Selects One Service Id based on the matching zuul /common path
#5 :: Generates Dynamic Client Code (as In Memory Proxy class) as Ribbon Client Code
#6 ,#7,#8 :: This Ribbon Client Code contacts the EurekaServer and gets less load balance instanceId of matched MS
#9 : Ribbon Client generates http call to interact with recieved instance id based MS instance
#10,#11 :: Based on the global path , method path of the URL ... the method in MS intance will execute and the generate results comes back Ribbon Client
#12 :: Ribbon Client sends the results to Client App/Browser as response