SPRING BOOT

MICRO SERVICES & CLOUD

BY

Mr.RAGHU SIR

06/02/2019

Spring Boot :-

⇒ spring boot is spring based Flw which is open source and developed by pivotal Team.

⇒ Available versions

Spring Boot 1.x

Spring Boot 2.x

⇒ Spring Boot provides AutoConfiguration which mean reduce Common lines of code in Application by programmer and handles Jars with version management.

⇒ (ie providing configuration code [xml/java] and maintaining All JARs required for project: parent Jars + child Jars...)

⇒ spring boot is a abstract maven project also called as parent maven project [A project with partial code and Jars]

⇒ Here programmer will not write configuration code but need to give input data using.

ⓐ Properties file

(application.properties

ⓑ YAML File (application.yml)

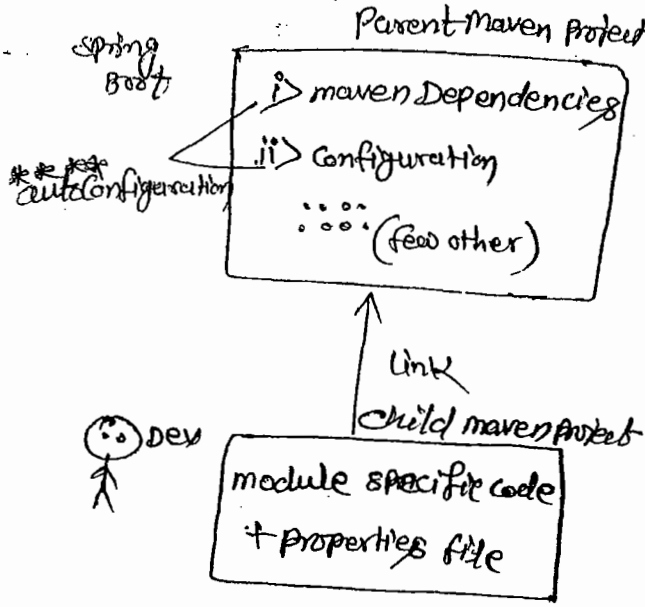
Spring

DEV

i> maven Dependencies (JARS)

ii) Configuration [xml / java]

iii) module specific code



Spring JDBC and Spring Boot JDBC [Sample code comparing]

① Spring JDBC-

ⓐ xml config:-

```
<beans ..>
   <bean name="dsObj" class="org.sf..DriverManagerDataSource">
    <property name="driverClassName"
                value="oracle.jdbc.driver.Oracle"/>
    <property name="username" value="system" />
    <property name="password" value="manager"/>
</bean>
</beans>
```

(b) maven Dependencies with version (JAR)

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>5.1.3.RELEASE</version>
</dependency>
```

(2) Spring Boot:-

(a) Starter dependency
(which gives config code and JARS)

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

(b) application.properties (or application.yml)

```
spring.datasource.driver-class-name = oracle.jdbc.driver.
                                      OracleDriver
spring.datasource.url = jdbc:oracle:thin:...
spring.datasource.username = system
spring.datasource.password = manager
```

07/02/2019

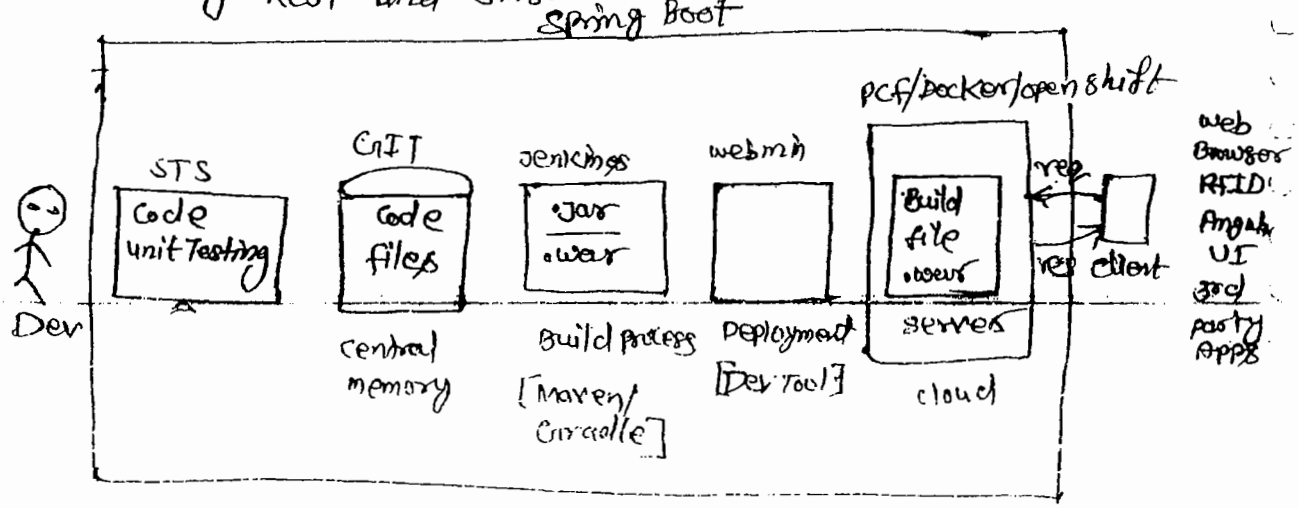
Spring Boot Supports end-to-end application process that is called coding ⇒ unit Testing ⇒ version control ⇒ Build ⇒ Deploy ⇒ client App.

ⓐ GIT (github.com) is used to store our files. It is called as central Repository or version control Tool.

ⓑ .java is converted to .class (Compile) ~~.class~~
.class (+other file .xml, .html...) converted to .jar/.war finally Build Process).

ⓒ Place .jar/.war in server and start server is called as Deployment.

ⓓ Spring Boot App is a service provider app which can be integrated with any UI client like Android, Angular UI, RFID (swiping machine), Any 3rd party APPs, web APPs,.. using Rest and JMS.



NOTE:-

(a) Spring Boot support two build tools maven and or Gradle.

(b) Spring Boot support 3 embeded server and 3 embedded databases These are not required to download and install.

Embedded Servers:-

i. Apache Tomcat
ii. JBoss Jetty
iii. Undertow

Embedded Data bases:-

i. H2
ii. HSQL
iii. Apache Derby

(c) Spring Boot supports cloud apps with microservices pattern. ["Both coding and deployment"]

[i] coding is done using Netflix Tools

[ii] Deployment is done using PCF Tools (or its equal...)

(d) Spring Boot supports basic operations:-

i. WebMvc and Webservices (REST)
ii. JDBC and ORM (Hibernate with JPA)
iii. Email, Scheduling, JMS, Security
iv. Cache and Connection Pooling
v. DevTools, Swagger UI, Actuator, and Profiles.

VI. UI Design using

Html, JSP, Thymeleaf ... etc.

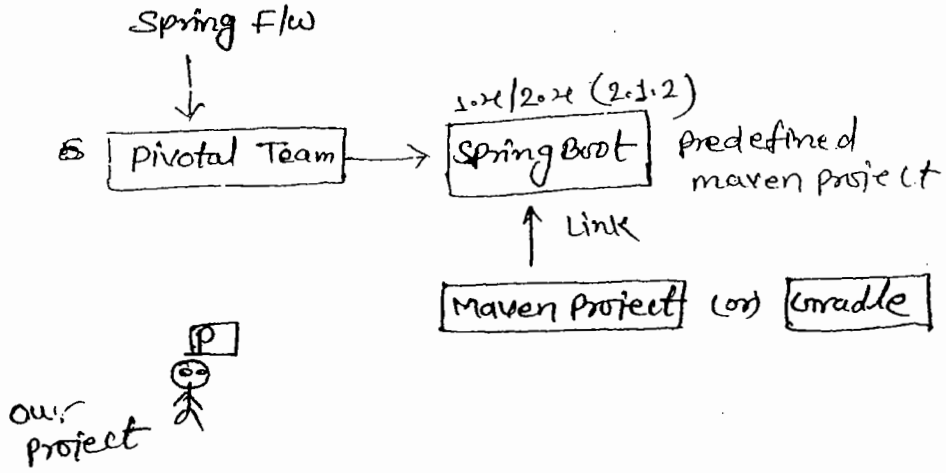
@ Supports Input Data (key = val) using (for AutoConfiguration code) properties file or YAML.

09/02/2019

Spring Boot Application Folder System

⇒ we can write Spring Boot application either using maven or using Gradle (one of build tool)

⇒ our project contains one parent project of spring boot which is internally maven project (holds version of parent)



⇒ Application should contain 3 major and required files.

Those are;-

1. SpringBootStarter class
2. application.properties / application.yml
3. pom.xml / build.gradle

1. SpringBootStarter Class :—

It is a main method class used to start ~~up At~~ our App.
It is entry point in execution. Even for both stand alone and web this file is used.

2. application.properties / application.yml :—

This is Input file for Spring Boot (Spring Container).
It holds data in Key = Value format.

*** File name must be "application" or its extended type.

** Even .yml (YAML) file finally converted to .properties only using SnakeYaml API

** yml is better approch to write length properties code.

3. pom.xml / build.gradle :—

This file holds all information about

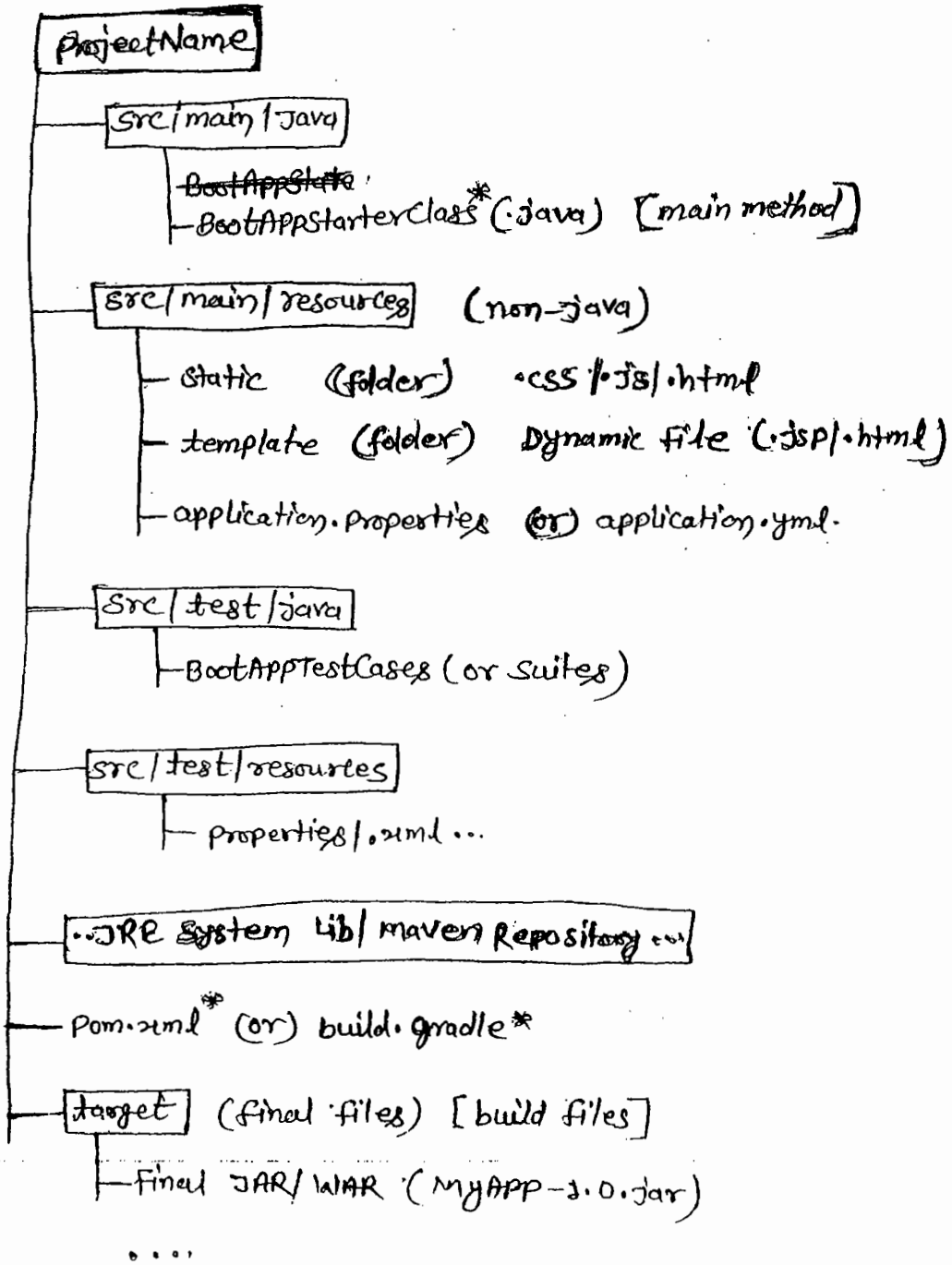
a. parent Boot Project version

b. App Properties (JDK Version / Maven / cloud versions ...)

c. Dependencies (JAR Details)

d. Build plugins (compile / WAR .. etc)

Application Folder System



11/02/2019

Chapter #1 Spring Boot Core

#a# Spring Boot Runners :—

A Runner is a auto-executable component which is called by container on application startup only once.

⇒ In simple this concept is used to execute any logic (code) one time ~~then~~ when application is started.

Types of Runners (2) :—

1. CommandLineRunner :-

This is legacy runner (old one) which is provided in Spring Boot 1.0 version.

*) It has only one abstract method :

"run(String ... args) : void"

*) It is a functional Interface (having only one abstract method)

*) Add one StereoType Annotation over Impl class level (ex: @Component). So that container can detect the class and create object to it.

---Code------

#Setup : JDK1.8 and Eclipse/STS

#1 Create Maven project (simple one)

> File > new > maven project > *** click check box

[V] create simple project

> next > Enter Details (example)

Group Id : com.app

Artifact Id : SpringBootRunnerx

Version : 1.0

> Finish

#2 open pom.xml and add parent, properties, dependencies with plugins

----------pom.xml----------

```
<project ...>
 ....
<!-- a. Parent project details -->
<parent>
<groupId> org.springframework.boot </groupId>
<artifactId> spring-boot-starter-parent </artifactId>
<version> 2.1.2.RELEASE </version>
</Parent>
<!-- b. Versions / properties -->
<Properties>
```

```
    <java.version>1.8</java.version>
  </properties>

  <!-- c. dependencies / JARS -->
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter</artifactId>
    </dependency>
  </dependencies>
  <!-- d. build plugins -->
  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>
</project>
```

#3 create properties file under src/main/resource folder

> Right click on "src/main/resources" > new > other

> Search and choose "File" > next > enter name

eg: application.properties > finish

#4 write Spring Boot Starter class under src/main/java folder

> Right click on "src/main/java" > new > class

> enter details, like:

Package: com.app

Name: MyAppStarter > finish

---- code ----

```
package com.app;
//ctrl+shift+o (imports)
@SpringBootApplication
public class MyAppStarter {
  public static void main(String[] args){
    SpringApplication.run(MyAppStarter.class, args);
  }
}
```

#5 = create one or more runner classes under src/main/java folder with package "com.app"

#Runner#1

```
package com.app;
//ctrl+shift+o
@Component
public class MyTestRunner implements CommandLineRunner, Ordered
{
```

```
    public void run(String ... args) throws Exception {
        Sopln("- from My Test Runner -");
    }
    public int getOrder() {
        return 55;
    }
}
```

#Runner #2

```
package com.app;
//ctrl+shift+o
@Component
@Order(666)
public class TypeARunner implements CommandLineRunner {
    public void run(String ... args) throws Exception {
        Sopln("- from Type A Runner -");
    }
}
```


Note:-

(1) Boot Application can have multiple runners
ex: EmailRunner, ImsRunner, SecurityRunner,
CloudEnvRunner, DevOpsRunner, DatabaseRunner ... etc.

(2) Boot provides default execution order. To specify programmer defined order use Interface:

Interface : Ordered (OR)

Annotation : @Order

Input Data to Runners :—

Programmer can pass one time setup data using CommandLine Arguments, in two formats :-

a. option Arguments

b. NonOption Arguments

Syntax :- --Key=Val [option Arguments]

ex :- --db=MySQL --db=Oracle

--env=Prod --server.port=9898

---etc:

Syntax: data [NonOption Arguments]

test clean package execute rollnone

--etc..

*) Data is converted into String[] (String ...) [Var-args] and sent to Runner class.

*) Read data based on Index

---Code---

package com.app;

@Component

public class ~~MyInputNumber~~ MyInputRunner implements CommandLineArgument {

public void run (String ... args) throws Exception

{

```
        Sopln(args[1]);
        Sopln(Arrays.asList(args));
    }
}
```

=> Execution:

Right click on starter class code (main)

Run As > Run Configuration

choose "Arguments" tab

Enter data in program arguments (with space)

--name=AJ Hello hi --db=mysql --db=oracle

click on Apply and Run

=> It is internally converted to:

String[] (String ...) ← var-args

--name=AJ	0
Hello	1
hi	2
--db=mysql	3
--db=oracle	4

12/02/2019

Anonymouse Inner class :-

⇒ A Nameless class and object created for an interface having abstract methods

*) In simple create one class without name and create object at same time without name. used only one time.

---Syntax--

```
new InterfaceName() {
    // Override all methods
}
```

---#Ex1-----

```
interface Sample {
    void show();
}
```

--- Anonymouse Inner class ---

```
new Sample() {
    public void show() {
    Sop("Hi");
}
}
```

---Ex#2---

```
interface CommandLineRunner {
    void run(String ... args) throws Exception;
}
```

---Anonymouse Inner class ---

```
new CommandLineRunner() {
  public void run(String ... args) throws Exception {
    Sopln("Hi");
  }
}
```

*) Java style configuration for CommandLineRunner

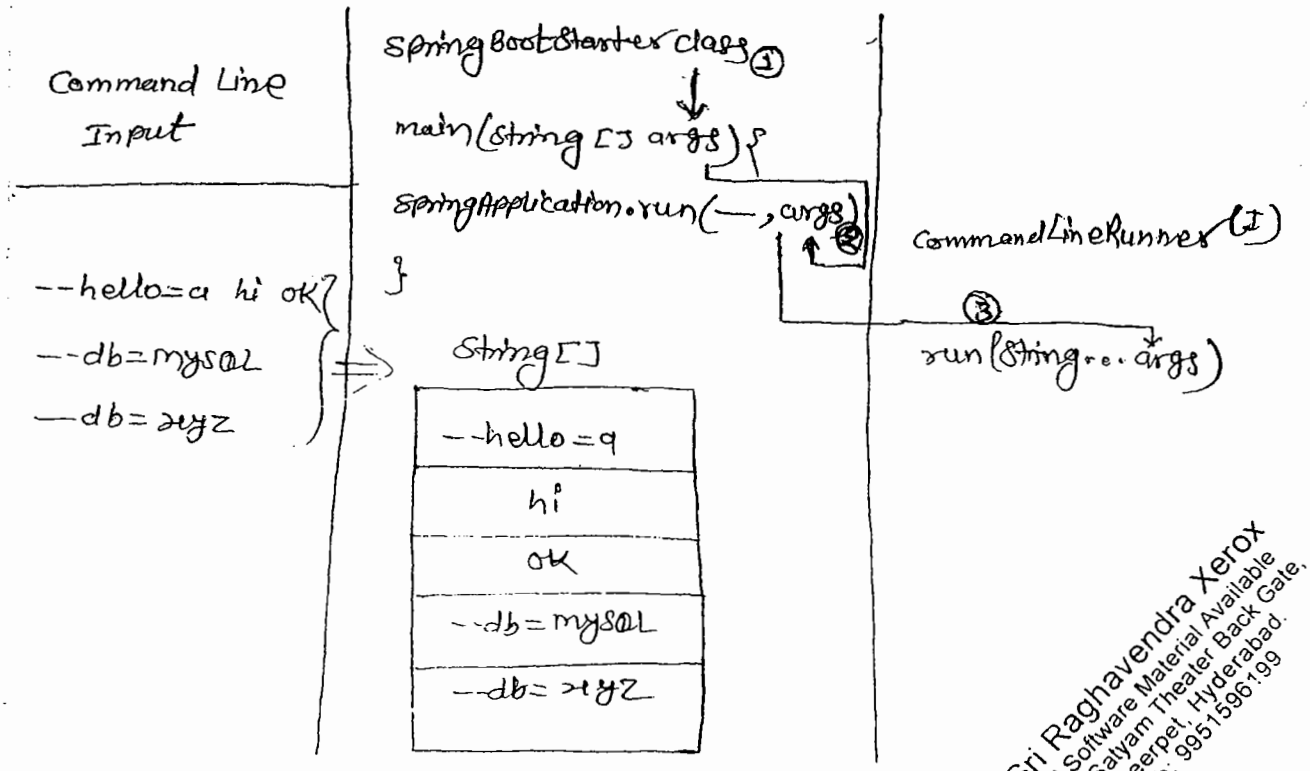
---Code ---

```
Package com.app;
//ctrl+shift+o (imports)
@Configuration
public class AppConfig {
// JDK 1.7 or before (Inner class style)
@Bean
public CommandLineRunner cob() {
  return new CommandLineRunner(){
    public void run(String ...args) throws Exception {
    Sopln (Arrays.asList(args));
  }
};
}
// JDK 1.8 or higher (Lambda)
@Bean
public CommandLineRunner cob2() {
  return (args) -> {
  Sopln (Arrays.asList(args));
};
}
```

Q) How CommandLineRunner works?

A) Command Line Arguments which are passed to application will be given to spring Boot starter main(...) method.

Those are stored as "String Array" (String [] SpringApplication.run(...) read this input and internally calls run(...) method of all RunnerImpl classes and pass the same data.

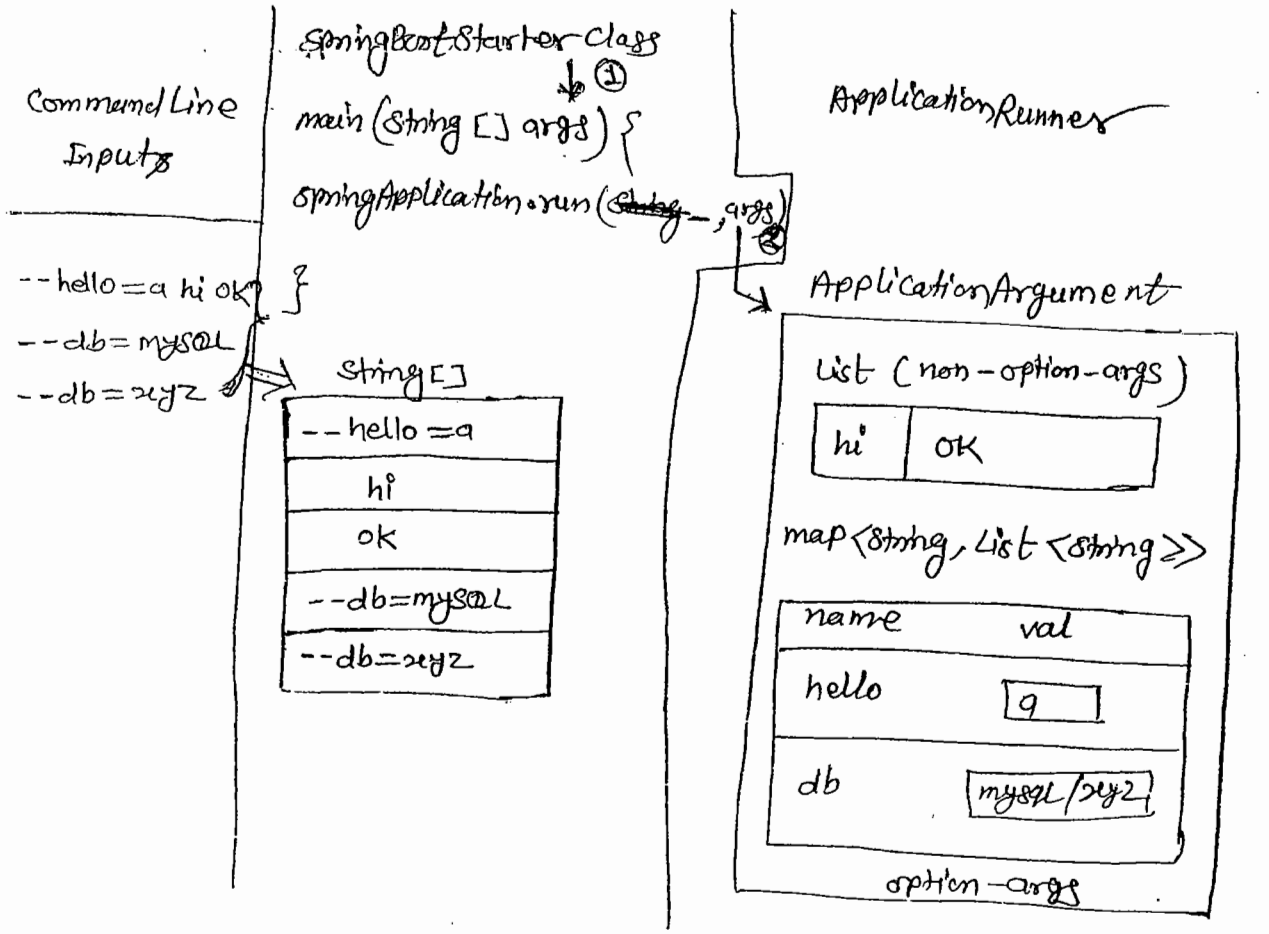


Q. ...CommandLineRunner (I)?

It is new type runner added in spring boot 1.3 which makes easy to access arguments.

⇒ This will separate option Arguments (as map<String, List<String>>) and Non-option Arguments (List<String>)

⇒ This Data stored in object of "ApplicationArguments" as given below



.... code

```
package com.app;
// ctrl + shift + o
```



```
@Component
public class AppRunner implements ApplicationRunner {
public void run(ApplicationArguments args) throws Exception {
Sopln(Arrays.asList(args.getSourceArgs()));
Sopln(args.getNonOptionArgs());
Sopln(args.getOptionNames());
Sopln(args.getOptionValues("db"));
Sopln(args.containsOption("bye"));
}}
```

Q.) What is the diff b/w CommandLineRunner and ApplicationRunner?

A) Working process is same. But CLR holds data in String[] format where as AR holds data as ApplicationArguments as option / Non-option format.

13/02/2019

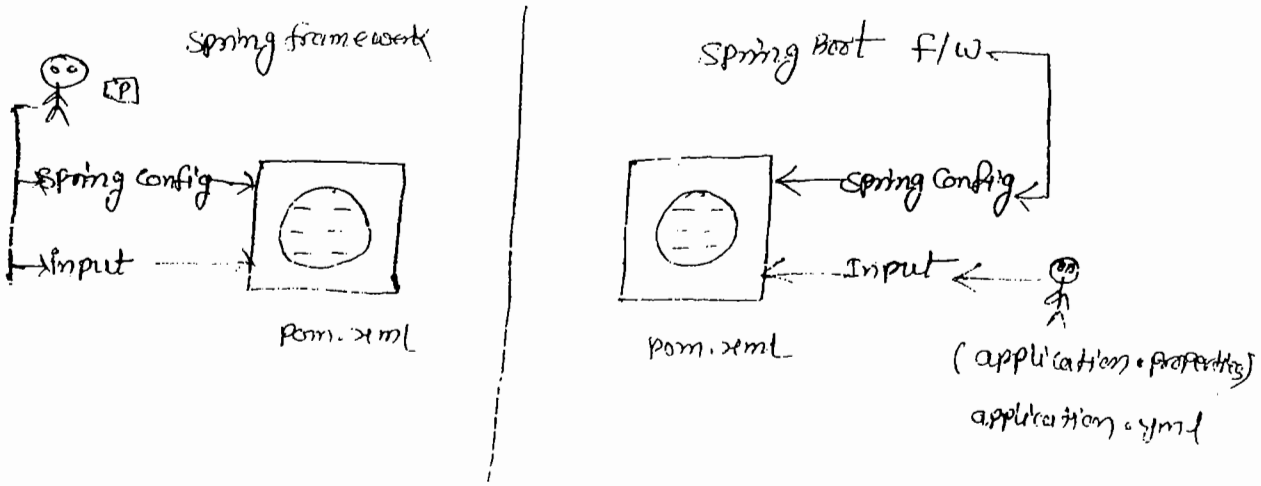
Spring Boot Input Data :—

application.properties or application.yml is a primary source input to spring boot (Spring Container).

⇒ Spring Boot f/w write configuration code (xml / Java config) for programmer.

⇒ Here we are not required to write (@Bean or <bean...>) Configuration for Common application setup like JDBC Connection, Hibernate properties, DispatcherServlet config, Security Beans...etc

⇒ But programmer has to provide input to above beans (objects) using properties or YAML file (any one)



application.properties

1. It holds data in key=value format

2. Keys are two types

   (a) Spring Boot defined (predefined)

   (Reference Link: https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html

   (b) Programmer defined

---code ------

#1. Create maven project and provide pom.xml and starter class

#2. application.properties (src/main/resources)

> Right click on src/main/resources folder

> new > other > search and select "file"

> enter name "application.properties"

> finish

---application.properties---

my.info.product.id=gggA

my.info.product.code=xyz

my.info.product.model-version=44.44

my.info.product.release_dte_enable=false

my.info.product.start-key=N

Note:

(a) allowed special symbols are
dot(.), dash(-) and underscore(_).

(b) key=value both are string type, spring supports type conversion (ex: String -> Int) automatically

(c) To read one key-value in code use ka Legacy
Syntax: @Value("${key}").

#8 Runner with Keys data (class)

```
package com.app;
//ctrl + shift + o (imports)
@Component
public class PropsInfo implements CommandLineRunner {

@Value("${my.info.product.id}")
 private int prdId;

@Value("${my.
 private String prdCode;

@Value("${my.info
 private double mdlVer;

@Value("${my.info
 private boolean isDteEnbl;

@Value("${my.info
 private char startKey;
```

```
public void run(String ... args) throws Exception
{
  Sopln(this);
 //Sopln(this.toString());
}
// Generate set/get and tostring methods
//alt+shift+S,R -> selectAll -> OK
//alt+shift+S, S -> OK
}
```


Note:

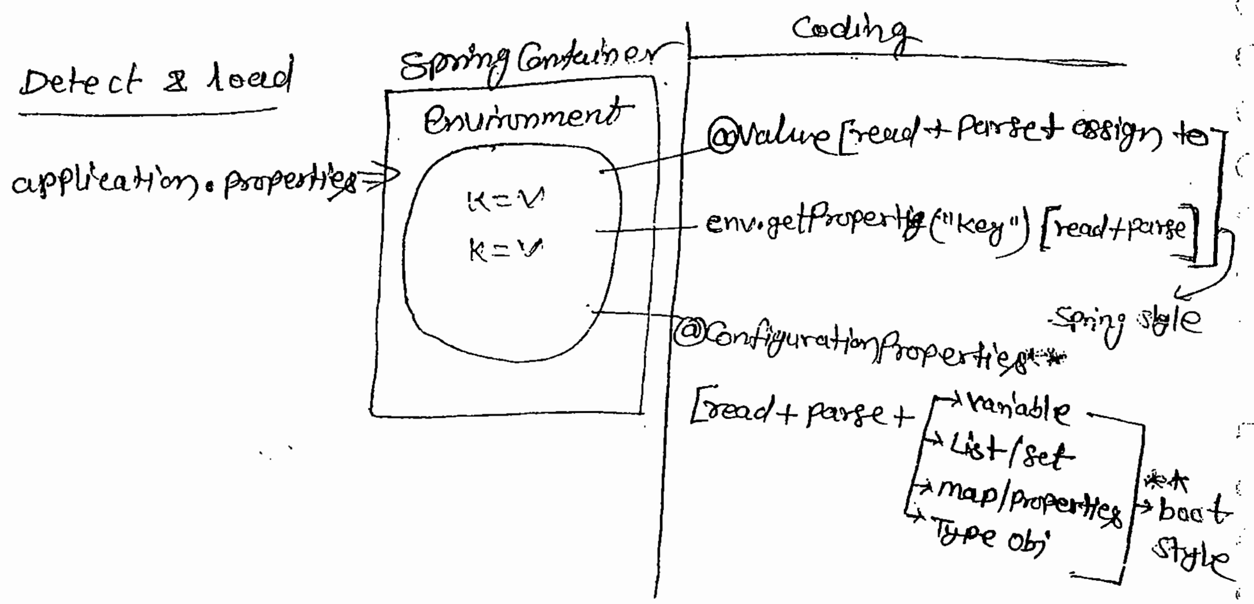
if Key data is mismatched with variable data type, then Spring container throws Exception: TypeMismatchException: Failed to convert value...

⇒ Spring Boot will search for file "application.properties" in project (4 different location)

⇒ one found (detected) then load into container and store as "Environment" object.

⇒ we can read data in legacy style @value or env.getProperty(..)

⇒ Boot Style (Bulk Loading) can be done using Annotation: @ConfigurationProperties



14/02/2019

@ConfigurationProperties :—

This Annotation is used to perform bulk data reading (multiple keys at a time) and parsing into one class type (stores in object)

⇒ Possible Conversions

1 Key = 1 variable

multiple keys with index = List / set / Array

multiple keys with Key-val Format = map or Properties

multiple keys with Common type = class object (HAS-A)

- - - - - Ex : - - - - - - - - - - - - - - - -

1. pom.xml, Starter class same as before

2. application.properties

```
# one variable data
my.prod.ID = 999
my.prod.code = AB
my.prod.ty-pe = true
```

my.prod.MODEL=P

List<DT> / Set<DT> / DT[]

my.prod.prjnm[0] = p1

my.prod.prjnm[1] = p2

my.prod.prjnm[2] = p1

Map or Properties

my.prod.mdata.s1 = 55

my.prod.mdata.s2 = 66

my.prod.mdata.s3 = 88

one class obj

my.prod.dpt.dname = AAA

my.prod.dpt.did = 8989

--- 3. Model class ---

```
Package com.app;
public class Dept {
  private int did;
  private String dname;
//alt+shift+S, R > selectAll > OK
// alt+shift+S,S > OK
}
```

4. model with Runner code

```
@ConfigurationProperties("my.prod")
@Component
public class ProductData implements CommandLineRunner {
```

```
@Autowired // legacy
~~private void run(String ... args~~
private Environment env;

public void run(String ... args) throws Exception {
 Sopln(env.getProperty("my.prod.ID"));
 Sopln(this);
}
private int id;
private String code;
private boolean type;
private char model;
private Set<String> prjnm;
// private List<String> prjnm;
// private String[] prjnm;

// private Map<String, Integer> mdata;
 private Properties mdata;
 private Dept dpt; //HAS-A

// alt+shift+S, R > Select all > OK
// alt+shift+S, S > OK
}
```

---- Ex #2 Load all key-values based on (common) prefix

** Do not provide any prefix at annotation level. make sure create variable with first level prefix (name before first dot) will auto loaded as a map or properties

---Code---

```
@ConfigurationProperties
@Component
public class products implements CommandLineRunner {
 private map my;
 public void run(String ...args) throws Exception {
    SopIn(this);
 }
  // generate set, get and tostring
}
```

*) Generating Random Values :-

⇒ we can use a direct expression

@value("${random.-}") in java code or in properties file.

⇒ possible random data is

Hexa Decimal value

int OR Long type

int OR long with range

UUID (Universal Unique Identifies)

Code

① application.properties

```
my.rnd.svl = ${random.value}
my.rnd.num = ${random.int}
my.rnd.bignum = ${random.long}
my.rnd.num.rng = ${random.int[30]}
```

my.rnd.num-rng-from-to = ${random.int[100,500]}

my.rnd.uuid-type = ${random.uuid}

② model class with Runner -

```
Package com.app;
@Component
public class ProductTest Implements CommandLineRunner {
  // @Value ("${my.rnd.sval}")  // recomended
  //@Value ("${random.value}")
  @Value ("${my.rnd.uuid-type}")
  private String code;
  //@Value ("${my.rnd.num}")
  //@Value ("${my.rnd.num-rng}")
  @Value ("${my.rnd.num-rng-from-to}")
  private int num;
  @Value ("${my.rnd.bignum}")
  private long numbig;
public void run(String...args) throws Exception {
   Sop(this);
}
// generated set, get, toString
}
```

15/02/2019

possible Locations for Properties (yaml) file

Spring Boot supports 4 default and priority order locations, which are loaded by container for Key = value data.

1. under project – under config folder

   project/config/application.properties

   (file:./config/application.properties)

2. under project (only)

   project/application.properties

   (file:./application.properties)

3. under project – under resources/config

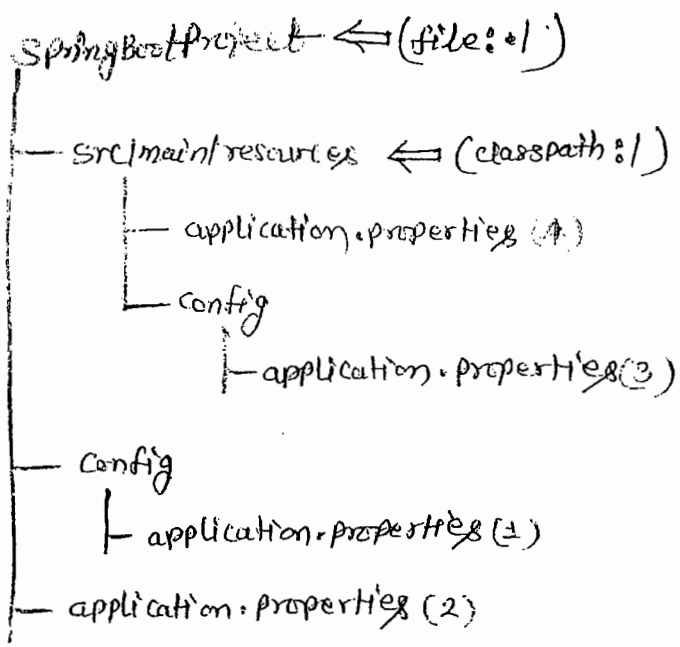
   project/src/main/resources/application.properties

   (classpath:/config/application.properties)

4. under project – under resources /

   project/src/main/resources/application.properties

   (classpath:/application.properties)



Specify programmer file name for application.properties (even yaml)

Spring Boot supports programmer defined properties (YAML) file name.

which can placed in any of 4 locations given as before (priority order is applicable if same file exist in all places)

Step#1 create your file under one location

ex: src/main/resources

mydata.properties

(only or any other location also valid)

Step#2 use run configuration and provide option argument > apply and Run

Ex: --spring.config.name = mydata

NOTE: To avoid default location select priority order and select exact properties file use option argument:

eu#1 --spring.config.location=classpath:/config/mydata.properties

eu#2 --spring.config.location=file:./config/mydata.properties

Create GIT Account:

follow (me):

https://github.com/javabyraghu

18/02/2019

YAML (YAMLian Language) :—

It is representation style of key=val without duplicate levels in keys if they are lengthy and having common levels.

⇒ File extension is ".yml"

⇒ It will hold data in below format key:<space>value

⇒ default name used in spring boot is application.yml

⇒ At least one space must be used, but same should be maintain under same level.

⇒ Spring Boot System converts .yml to .properties using SnakeYaml API

⇒

⇒ Snake YAML will

a. check for space and prefix levels

b. trace keys for data find

c. convert .yml to .properties

Internally system is while loading.

⇒ Consider below example properties file

---- "application.properties" ----

my.data.id-num=10

my.data.code_val=AB

my.data.enabled.costBit=3.6

my.data.enabled.valid=true

Its equal YAML file looks as

---- application.yml ----

```
my:
  data:
    id-num: 10
    code_val: AB
    enabled:
        costBit: 3.6
        valid: true
```

eze#2 application.properties

~~my.data~~ my.code.id = 56

my.code.mdn.type = big

my.code.str.service = ALL

my.code.str.service.info = true

my.code.mdn.obj = CRL

my.code.cost = 3.36

my.code.mdn.sale = yes

⇒ its equal yml file is:-

(I)

```
My:
  code:
    id: 56
    mdn:
      type: big
    str:
      service: ALL
        info: true
      obj: CRL
    cost: 3.36
      sale: yes
```

(II)

```
My:
  code:
    id: 56
    cost: 3.36
    mdn:
      type: big
      obj: CRL
      sale: yes
    str:
      service: ALL
        info: true
```

⇒ Key = Val format

List <DataType> / set <DataType> / Array(<DataType>[]) style :—

*) In properties file we use index based data representation, starts from zero.

*) In yml file use just dash (-) with <space> value under same level.

--Ex: -- application.properties ----

my.code.version[0] = V1

my.code.version[1] = V2

my.code.version[2] = V3

--- application.yml ----------

my:
  code:
    version:
      - V1
      - V2
      - V3

final meaning is :-

V1	0	List<String>
V2	1	set<String>
V3	2	String[]

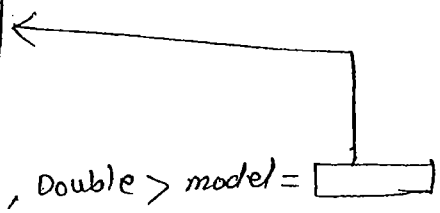
=> Key = value format

map/properties style :-

*) Consider below Example : —

map / properties (java.util)

key	value
a1	6.6
a2	7.7
a3	8.8



map<String, Double> model = [ ]

*) Its equal properties file will be

ex: application.properties

my.data.model.a1 = 6.6

my.data.model.a2 = 7.7

my.data.model.a3 = 8.8

Its equal : application.yml file

```
my:
  data:
    model:
      a1: 6.6
      a2: 7.7
      a3: 8.8
```

---Example Application---

1. pom.xml

Add one extra dependency for auto detection of keys in properties/yml file

```
<dependency>
  <groupId>org.springframework.boot</
  <artifactId>spring-boot-configuration-process</
  <optional>true</.
</dependency>
```

2. write Starter class (same as before)
[main method]

3. application.yml

```
my:
  prod:
    id: 5
    code: AB
    cost: 8.8
    versions:
      -v1
      -v2
    models:
      m1: 2
      m2: 3
```

4. Model class:

```
package com.app;
//ctrl+shift+o
@ConfigurationProperties("my.prod")
@Component
public class Product {
    private int id;
    private String cost;
    private List<String> version;
    private map<String, Integer> models;
  //set, get ... toString ...
}
```

5. Runner class:

```
package com.app;
//ctrl+shift+o
@Component
public class ConsoleRunner implements CommandLineRunner{
  @Autowired
  private Product prod;
  public void run(String ... args) throws Exception {
      System.out.println(prod);
  }
}
```

19/02/2019

*) Child data (HAS-A) using YAML

yaml data to pojo/Bean

At a time multiple values (key pairs) can be converted to one pojo (Java Bean/spring Bean) using @ConfigurationProperties annotation.

*) Pojo Rules :-

-> class, variables, **default Constructor with set/get methods

*) Java Bean :-

Pojo Rules + inheritance + special methods + param Constructor

*) Spring Bean :-

Pojo Rules + inheritance(spring API) + Annotations (spring API) + special methods (object class and spring API) [toString()....]

#1 Beans (Pojos)

```
package com.app.beans;
//ctrl+shift+O (imports)
public class model {
private int mid;
private String mcode;
private List<String> colors;
//def. const
(alt+shift+S, O -> Deselect All -> ok)
//set/get methods
(alt+shift+S, R -> select All -> ok)
//toString :
(alt+shift+S, S -> OK)
}
```

```
package com.app.bean;
//ctrl+shift+O (imports)
@Component
@ConfigurationProperties("my.dt")
public class Product{
 private int pid;
 private String Model mo; //HAS-A
//Const, set, get ...toString ..
}
```

#2 Runner class

```
package com.app.runner;
//ctrl+shift+O (imports)
@Component
public class ConsoleRunner implements CommandLineRunner{
 @Autowired
 private Product pob;
 public void run(String ...args) throws Exception{
  sopln(pob);
}}
```

#3 application.yml

```
my:
  dt:
    pid: 55
    mo:
      mid: 88
      model: AB
      colors:
        - RED
        - GREEN
```

----4#4 Starter class + pom.xml ------ --(Same as before)

*) Place holder- pieces in yml or properties :—

internall placeholder are used to re-use existed key value for another key as a part or full.

⇒ Read as ${fullPathKey} (it must be properties style even in yml file)

--application.properties ------

my.dt.pid = 68

my.dt.mid = ${my.dt.pid}

my.dt.msg = Welcome to ${my.dt.pid}

--application.yml ------

```
my:
  dt:
    pid: 68
    mid: ${my.dt.pid}
```

NOTE:-

1. Symbol '#' indicates a comment line in yml file

2. using 3 dash (---) Symbols in yml is allowed which indicates one file is devided into multiple files internally (mainly used for Profiles***)

-- ex: application.yml ------

```
#Hello data to Select
my:
  dt:
    pid: 59

my:
  dt:
    mo:
      mid: 89
```

3. Priority order for .yml is same as .properties file

Search locations in order

a. file:./config

b. file:./

c. classpath:/config

d. classpath:/

**file:./ = Under project folder

classpath:/ = under src/main/resources

Priority order for key search :-

Spring boot has provided default priority to "option Arguments" (Command Line Args) with format --key=value

⇒ If not found, next level is .properties

⇒ else finally choosen .yml

⇒ No where found default value.

---

priority order	Search Index
1	1. Option Arguments
2	2. properties file ⓐ file:./config ⓑ file:/ ⓒ classpath:/config ⓓ classpath:/
3	3. YAML file ⓐ file:./config ⓑ file:./ ⓒ classpath:/config ⓓ classpath:/

Q) Difference B/w @Value (Spring Annotation) & @ConfigurationProperties

Type / Cases	Spring Boot @ConfigurationProperties	Spring F/w @Value
1. Kebab Case Ex: std-name Std_name STD-NAME STD-name	✓	×
2. Camel Case Ex:- stdName stdNameModel	✓	×
3. SpEL (Expression Language) #{Language} Ex:- #{2+3} #{'hello'.toUpperCase()}	×	✓

Spring Boot Profiles :-

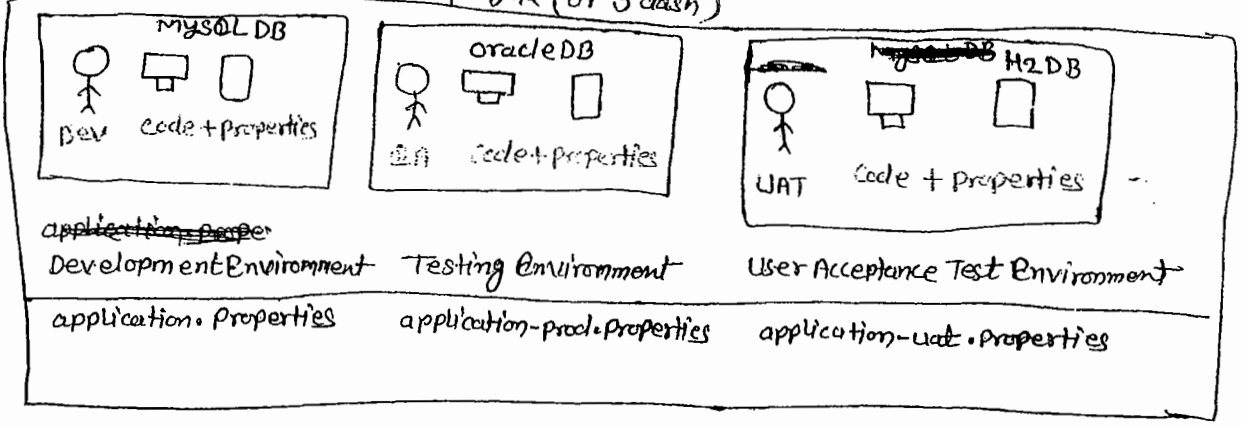
⇒ In Real Time, Application is Developed in - Dev Environment, Tested in - Quality Analyst (QA) Environment, Maintained in - PROD Environment, client tested in - UAT Environment, Go Live in - Cloud/Prod Environment

⇒ Environment is place where our application is running or what is current state of application.

Example:- dev = development
QA = Quality Analysis.
MS = Management Service,
PROD = Production
UAT = User acceptance Testing
Cloud = Cloud Environment

In this case we should not modify existed properties classes rather we have to create with key = val data. File naming rule is:

application-{profile}.properties
application-{profile}.yml (or 3 dash)



20/02/2019

Profile Specific Tasks :—

Profile supports Environment based code (Task) selection, not only properties (yaml).

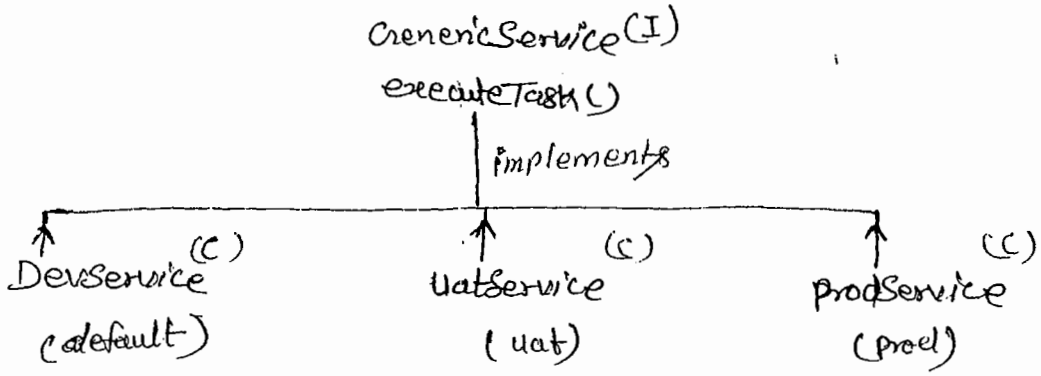
But in this case class should have @Profile("—") with @Configuration or @Component (or its equal stereotype)

stereotype Annotations :—

@Component, @Controller, @Service, @Repository, @RestController

⇒ Consider Example profiles default production (prod), uses Acceptance Test (uat) then

Profile code	properties file	class level Annotation
default	application.properties	@Profile("default")
prod	application-prod.properties	@Profile("prod")
uat	application-uat.properties	@Profile("uat")



---- Example ----------

⇒ File > new > spring starter project > Enter Details

Project Name : SpringBootProfiles

and also

GroupId : org.sathyatech

artifactId : SpringBootProfiles

Version : 1.0

** Package : org.sathyatech.raghu

> Next > Next > finish

---- code ----

1. Starter class

2. Properties files

(a) application.properties

my.prf.code = Hello from default

(b) application-uat.properties

my.prf.code = Hello from UAT

© application-prod.properties

my.prf.code = Hello from prod

3. Service interface and impl classes

```
package org.sathyatech.raghu.service;
public interface GenericService {
    public void executeTask();
}
```

```
package org.sathyatech.raghu.service.impl;
// ctrl+shift+o
@Component
@Profile("default")
public class DevService implements GenericService {
    @Value("${my.prf.code}")
    private String code;
    public void executeTask() {
        sop("from default profile");
        sop("code is:"+code);
    }
}
```

```
package org.sathyatech.raghu.service.impl;
// ctrl+shift+o
@Component
@Profile("prod")
```

```
public class ProdService implements GreenetService {
    @Value("${my.prf.code}")
    private String code;
    public void executeTask() {
    Sopln("from production profile");
    Sopln("code is:" + code);
    }
}
```

```
package org.sathyatech.raghu.service.impl;
//ctrl+shift+o
@Component
@Profile("uat")
public class UatService implements GreenetService {
    @Value("${my.prf.code}")
    private String code;
    private vo
    public void executeTask() {
    Sopln("from UAT profile");
    Sopln("code is:" + code);
    }
}
```

NOTE:

[1] use key spring.profiles.active = profile we can provide using 3 ways. Those are:

(a) Command Line Arguments (option Argument)

Ex:- --spring.profiles.active = prod

(b) in application.properties

Ex:- spring.profiles.active = prod

(c) VM (JVM/System) Argument

Ex:- -Dspring.profiles.active = prod

> Right click on ~~project~~ starter class > Run As > Run Config
> choose Arguments > Enter below format in Vm Arguments
-Dspring.profiles.active = prod
> Apply and Run

[2] Key search highest ~~property~~ priority is given in same profile properties file, if key is not found in current profile properties file then goes to default properties file.

if no where key is found then
@Value - IllegalArgumentException
@ConfigurationProperties
-Default value of DataType
is choosen by container.

Example

application.properties	application-prod.properties	application-uat.properties
a=10 b=7 c=6	a=15 b=20	a=30

Case#1 spring.profiles.active = prod

then

Key	value
a	15
b	20
c	6
d	0

Case#2 spring.profiles.active = uat

then

Key	value
a	30
b	7
c	6
d	0

21/02/2019

Including childs for Active Profiles :—

In Spring Boot Applications Active profile can be specified using key

--spring.profiles.active = [__]

⇒ In this case one properties file is loaded into memory which may have more set of key = value pairs.

⇒ These can be devided into multiple child properties file and loaded through active profile, also called as "Profiles Include".

⇒ This can be done using key spring.profiles.include=__,__,__,....

--- Example ---

spring.profiles.active = prod

(main)

application-prod.properties

my.id=10

spring.profiles.include = pemail,db

(child#1)

application-pemail.properties

host=a

(child#2)

application-db.properties

dcn=xe host=b

Spring Container will load

⇒ Parent (main) profile first (all its key=value pairs)

⇒ Then child Profiles in given order will be loaded.

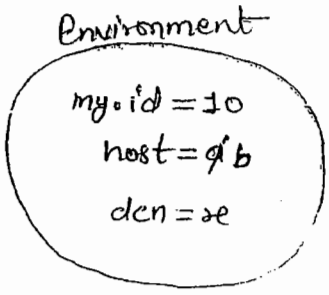
for above example, priority for loading (loading order is)

ⓐ application-prod.properties

ⓑ application-pemail.properties

ⓒ application-db.properties

⇒ Then Environment Looks like :--



Profiles using YAML :—

YAML Files also works same as properties file for both "active" and "include" profiles.

⇒ File Naming Rule :-

application-{profile}.yml

example :—

application.yml	(default)
application-prod.yml	(prod)
application-uat.yml	(uat)

Multiple profiles using YAML :—

YAML File supports writting multiple profiles in one file using Symbol 3 dash (---).

application.yml

```
my:
  prf:
    id: 666
---
my:
  prf:
    id: 999
spring:
  profiles: prod
```

application.yml

application-prod.yml

```
---
my:
  prf:
    id: 888
spring:
  profiles: uat
```

-- application-uat.yml

NOTE:

1# To specify active and include profiles use

(a) Option Arguments

--spring.profiles.active = prod

--spring.profiles.include = prodemail

(b) use properties file

spring.profiles.active = prod

spring.profiles.include = prodemail

(c) use YAML file

```
spring:
  profiles:
    active: prod
    include:
      -prodemail
```

(1) Starter and pom.xml same as before

(2) application.yml

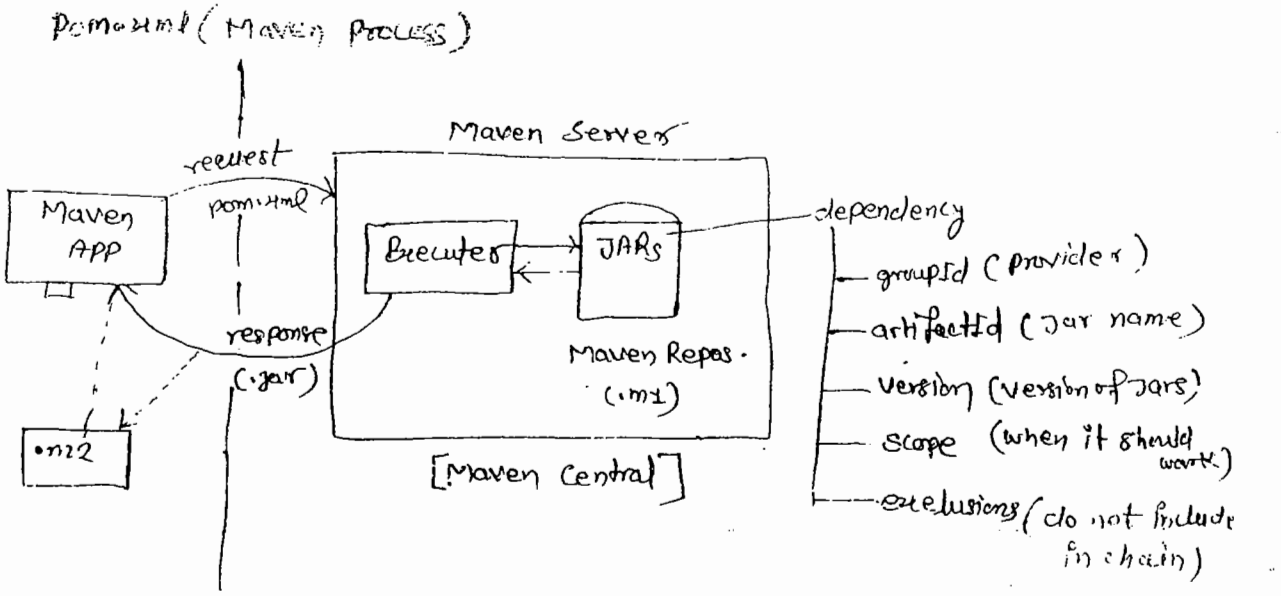
```
my:
  prf:
    id: 666
spring:
    profiles:
    active: prod
      include:
      - prodemail
---
my:
  prf:
    id: 999
  spring:
    profiles: prod
---
my:
  prf:
    id: BBB
  spring:
    profiles: uat
---
my:
  prf:
    email: abc
  spring:
    profiles: prodemail
```

(3) Bean and Runner class

```
package com.app.bean;
//ctrl+shift+o (imports)
```

```
@Component
@ConfigurationProperties("my.prf")
public class productRunner implements CommandLineRunner {
    public void run(String... args) throws Exception {
        Sop In (this),
    }
    private int id;
    private String email;
    // set, get ... toString ...
}
```

25/02/2019



Maven is dependency management and build tool used to handle both standalone and ArchType (web, restful...) applications.

*) Dependency management :—

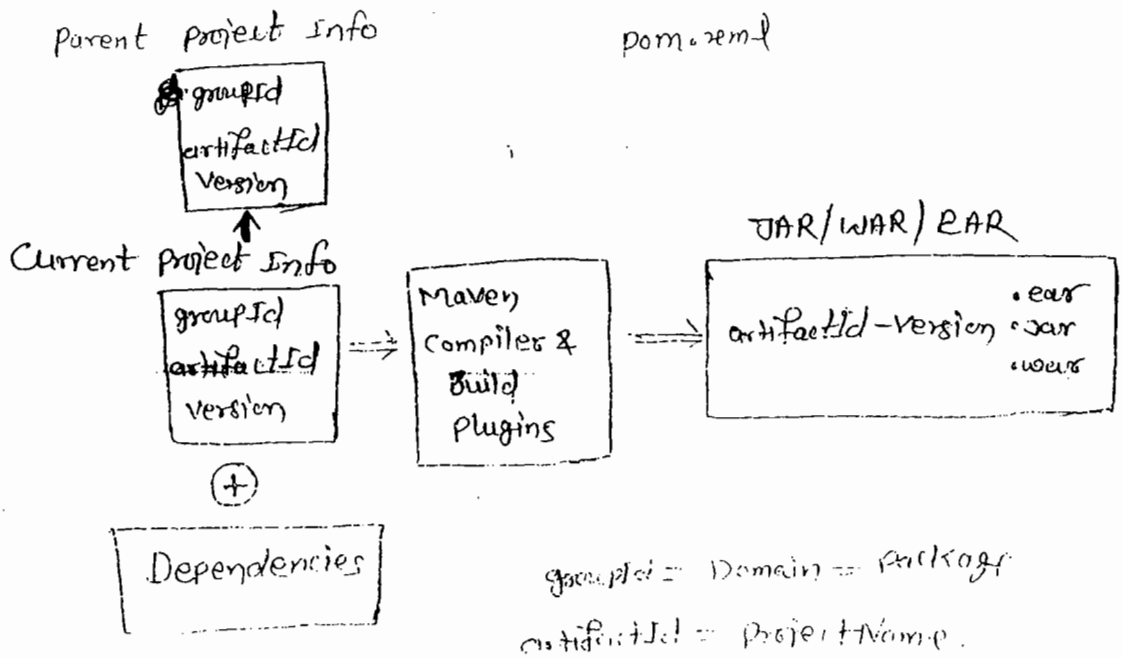
Getting main jars and its child jars with version support (without conflicts) into project workspace is called as Dependency Management.

*) build :—

Converting our application into final JAVA executable format i.e. .jar/.war/.ear

Major components of pom.xml :—

1. Current project Info
2. parent project Info
3. Dependencies
4. Build Plugins



groupId = Domain = package

artifactId = projectName.

---pom.xml format ---------

```
<project ....>
<modelVersion> 4.0.0 </modelVersion>
<!-- Current project Info -->
<groupId> a.b </groupId>
<artifactId> HelloApp </artifactId>
<version> 1.0 </version>

<!-- parent project Info -->
<parent>
    <groupId> a.a </groupId>
    <artifactId> DbApp </artifactId>
    <version> 6.3 </version>
</parent>

<!-- PROJECT JARS DETAILS -->
<Dependencies>
    <dependency>
        <groupId> a.b </groupId>
        <artifactId> Web-Test </artifactId>
        <version> 5.6 </version>
        <Scope> compile </Scope>
        <exclusions>
          <exclusion>
            ...
        [illegible]
        [illegible]
```

```
            </exclusion>
            </exclusions>
        </dependency>
    </dependencies>
<!-- PLUGINS DETAILS -->
    <build>
      <plugins>
        <plugin>
            <groupId> org.maven... </groupId>
            <artifactId> maven-compiler... </artifactId>
        </plugin>
      </plugins>
    </build>
</Project>
```

* Dependency exclusions :

when <dependency> tags is added in pom.xml then it will download parent jars and all its child jars also.

To avoid any one or more child jars from chain, use concept called exclusion.

Syntax —

```
<dependency>
    <groupId> </groupId>
    <artifactId> </artifactId>
    ...
  <exclusions>
```

```
        <exclusion>
            ..groupId, artifactId...
        </exclusion>
    </exclusions>
</dependency>
```

Ex:#

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.1.5.RELEASE</version>
    <exclusions>
        <exclusion>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</artifactId>
        </exclusion>
        </exclusions>
        <exclusion>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

26/02/2019

Scope (<scope> </scope>) in Dependency :—

for every dependency one scope is given by ~~maven~~ maven ie default scope : compile

*) This Tag is optional and indicates when a JAR should be used/loaded.

*) POM Format ---------

```
<dependency>
    <groupId> ... </groupId>
    <artifactId> ... </artifactId>
    <scope> .. </scope>
</dependency>
```


Possible Maven Dependency Scopes are (5) :—

(1) Compile :—

A JAR Required from Compilation time onwards .. it is only default Scope.

(2) Runtime :-

A JAR Required when we are running an application, not before that.

(3) test :—

A JAR Required only for UnitTesting time.

(4) Provided :-

A JAR provided by Servers or frameworks (Container..) ..

(5) System :-

A JAR ~~Required~~ loaded from file system (Like D:/abc/myjar/..).

In this case we should also give <Systempath> with location of JAR.

ex :- <Systempath>D:/asf/lib/</systempath>

Maven Goals Execution :—

Maven clean :-

It is used to clear target folder in maven project. i.e delete all old files from target.

Maven install :—

It will download all required ~~pal~~ plugins and also

> Compile the source files
> load required properties
> Executes JUnit Test cases
> Create final build (.jar/.war)

---

clean :—

> right click on project > Run as > Maven clean.

Install :—

> right click on project > Run as > maven install

Build :-

> Right click on project > Run As > Maven build ...

> provide goals like clean install > also choose SkipTests

> Apply and Run.

*) Update JDK to project before install or build else "BUILD FAILED" Error will be displayed.

*) A final jar created with name format artifactId-version.jar

*) Maven Build plugin (integrated with Spring Boot) must be provided in pom.xml

```
<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
```

*) Version management for Spring Boot applications :-

For all required Dependencies (mostly used) Spring Boot Parent project provides fixed and stable version management.

=> To see all jars provided with that version,

(a) Go to pom.xml

(b) ctrl + mouse over <artifactId> then click

(c) Search for tag properties

27/02/2019

Spring Boot Starter class concepts :-

[1] Spring boot starter class uses run(..) method from class "SpringApplication" defined in package: org.springframework.boot which creates spring container using "AnnotationConfigApplicationContext (C)"

ApplicationContext (I)
↑ IS-A
ConfigurableApplicationContext (I)
↑ IS-A
AnnotationConfigApplicationContext (C)

*) In case of Web/WebServices, Container is created using class "AnnotationConfigServletWebServerApplicationContext (C)

Note:

(a) ApplicationContext can be customized even using it supportive methods and API Types.

Ex:- for banner use Banner.Mode.Property Banner.Mode.OFF (to turn of banner)

(b) Mode is Inner enum defined in functional interface "Banner".

(c) We can provide our own banner using file: banner.txt (create under classpath) ie: src/main/resource/banner.txt (https://devops.datakolle.com/banner.txt/index.html)

---- Sample Code ----------

```
package com.app.starter;
// ctrl + shift + o (imports)
public class MyTest {
psv main (String[] args) {
SpringApplication sa = new SpringApplication(MyTest.class);
// sa.setBannerMode(Banner.Mode.OFF);
// some other configuration
ConfigurableApplicationContext c = sa.run(args);
sop(c.getClass().getName());
}
}
```

28|02|2019

2] Spring Boot Starter class package behaves as base package for component scan of all classes having annotated with @Component [or its equal]

Annotations are: (class should have any one)

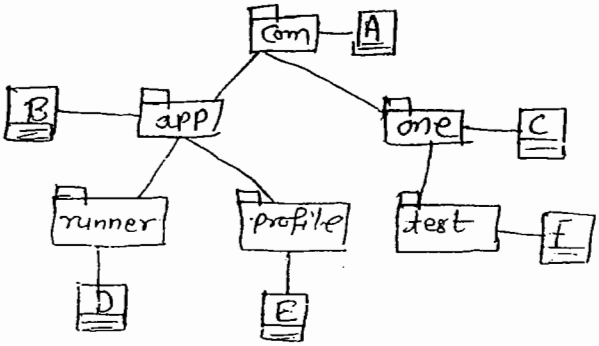
@Component
@Repository
@Service
@Controller
@RestController
@Configuration

---- Example ----------

*) Consider below starter class :-

```
package com.app;
//ctrl + shift + o (imports)
@SpringBootApplication
public class MyStarter{ ..... }
```

⇒ In this case only classes under package "app" and its sub package classes are detected by spring (Boot) container by default.



~~starter class~~

⇒ Package

⇒ class

⇒ Programmer can provide externally basepackage using @ComponentScan

(a) Avoid starter class package and provide our own.

Ex :- @ComponentScan("com.one")

** In this case starter package classes are not included.

(b) provide our own and starter package (even other package) using array style {—, —, —, ...}

Ex:- @ComponentScan({"com.one", "com.app", "com"})

(c) we can provide one common package name which covers all sub-levels.

Ex:- @ComponentScan("com")

---- Example code ----

```
package com.app;
//ctrl+shift+o (imports)
@SpringBootApplication
//@ComponentScan("com.one")
//@ComponentScan({"com.one", "com.app", "com"})
@ComponentScan("com")
public class MyStarter{ ..... }
```

[3] Every SpringBoot Application starter class itself Component. i.e. @SpringBootApplication is having @Component Annotation internally.

⇒ It is only highest priority component by default, if app has multiple components.

Ex: We can convert starter even as Runner.

```
@SpringBootApplication
public class MyStarter implements CommanelLineRunner {

    public void run(String ... args) throws Exception {

     Sopln("from starter");
    }
    ... main ...
}
```

[4] Every Spring Boot starter class itself A Configuration class (@Configuration) which auto detects other Config classes even without @Import annotation.

ie we can define @Bean (objects) with Java Style object creation in Starter class.

---Example---

① Model classes

```
Package com.app.model;
Public class Admin {
   Private int aid;
   Private String aname;
//Const, set/get .. toString ...
}
```

```
package com.app.model;
public class Product {
    private int prodId;
    private String prodName;
// Const, set/get ... toString ...
}
```

② AppConfig

```
Package com.app.config;
//ctrl + shift + o
@Configuration
public class AppConfig {
    @Bean
    public Admin aobj() {
        Admin a = new Admin();
        a.setAid(15);
        a.setAname("GTH");
        return a;
    }
}
```

③ Starter class

```
package com.app;
//ctrl + shift + o
@SpringBootApplication
//@Import(AppConfig.class) not required
public class MyTestApp implements CommandLineRunner {
```

```
@Autowired
 private Product p;
@Autowired
 private Admin a;

public void run(String ...args) throws Exception {
System.out.println("from starter::" +p);
System.out.println("from starter::" +a);
}
public static void main(String[] args) {

 SpringApplication.run(MyTestApp.class, args);

}
@Bean
 public Product pobj() {
   Product p=new Product();
   p.setProdId(100);
   p.setProdCode("AA");
   return p;
  }
}
```

01/03/2019

Spring INITIALIZR :- Link : https://start.spring.io/

This website is used to Generate one maven (or Gradle Project) for spring Boot Apps with all Configuration and setup.
Like starter class, application.properties, pom.xml, folder System etc..

⇒ By using this we can create Boot App which can be imported to normal Eclipse IDE or any other equal (No STS Required)

⇒ Even STS (or Manual Approches) uses internally SPRING INITIALIZR only.

Step #1 open Browser and type url

https://start.spring.io/

Step #2 provide all details and click on Generate Project

Step #3 It will be downloaded as .zip, Extract this to one folder

Step #4 open Eclipse (or any IDE), then

> Right click on project Explorer > choose Import...
> type Maven > select Existed Maven Project
> *** Write/browse location of extracted folder where pom.xml is available > click Enter > choose next
> finish

Spring Boot Data JPA

#1 Data JPA provide @NoRepositoryBean (s) which are auto configured and self logic implemented for basic operations.

ie; Programmer not required to write any logic for basic operations (No Implementation class and method)

⇒ Configuration for Datasource(I), SessionFactory (I), HibernateTemplate (c), HibernateTransactionManager (c) all are not required.

⇒ When we add below dependency in pom.xml it will download Jars and above @Config code given from parent project.

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

#2 Data JPA provides "Embedded Database support". It means Database provided in application itself.

⇒ It is not required to download and Install, not even properties required (Like driver class, url, user, password)

⇒ Spring Boot support 3 embedded Dbs.

Those are: H2, HSQLDB, Apache Derby

⇒ We can use any one Embedded Database which runs in RAM (Temp Memory)

⇒ It uses hbm2ddl.auto = create-drop.
ie. Tables are created when App starts and deleted before App stops.

⇒ These DBs are used in Both Development and Testing Environments, but not in production.

#3 Spring Boot also supports Both SQL (MySQL, Oracle, ...) and NoSQL (MongoDB) Databases also..

#4 Data JPA Supports Special Concept "@Query methods an easyway to code and fetch data from DB" (ex: findBy, @Query..)

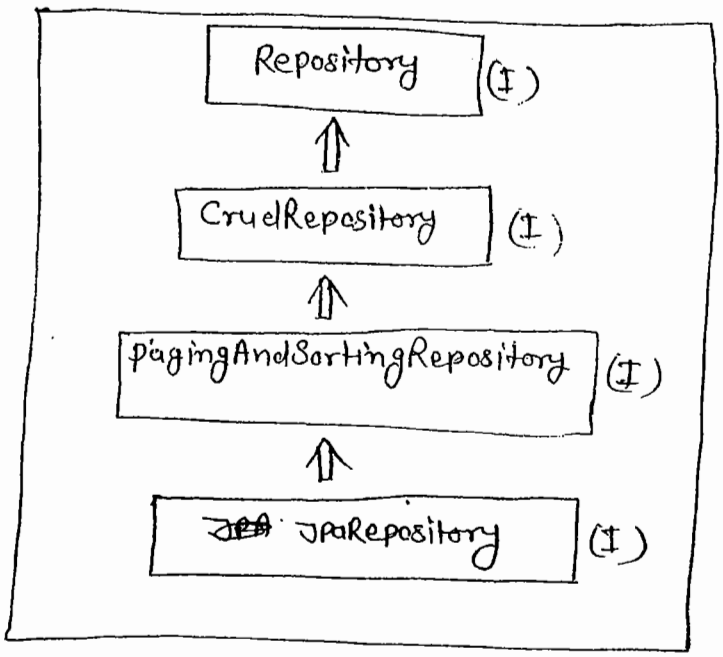
#5 Data JPA Supports Easy Connection pooling (Auto Config) Concept

#6 Data JPA Supports Cache Management (Autoconfig).

---

*) Repository Interfaces :—

Data JPA has provided Repository interfaces in package "org.springframework.data.repository"



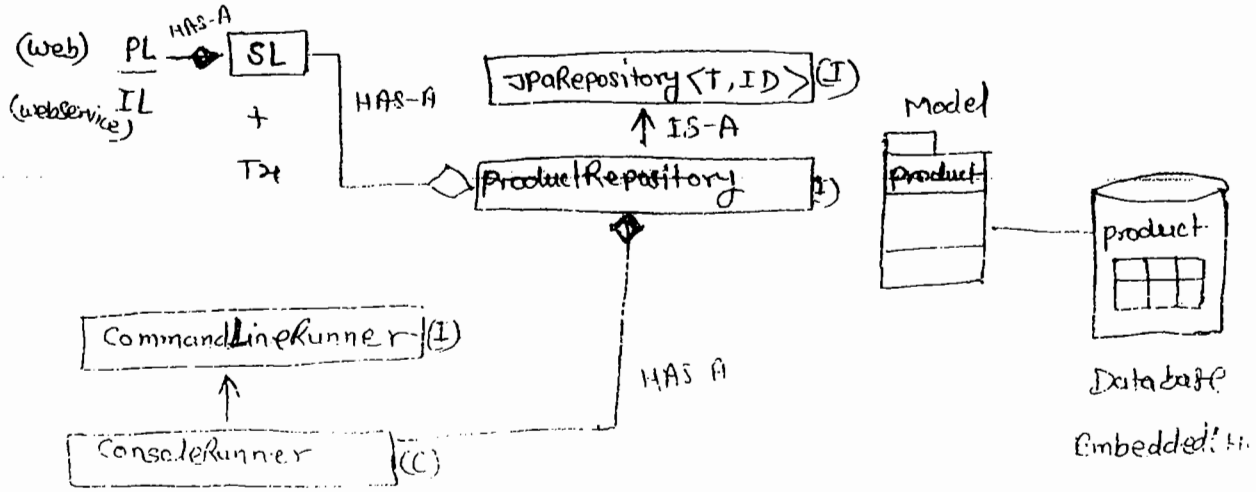
IS-A

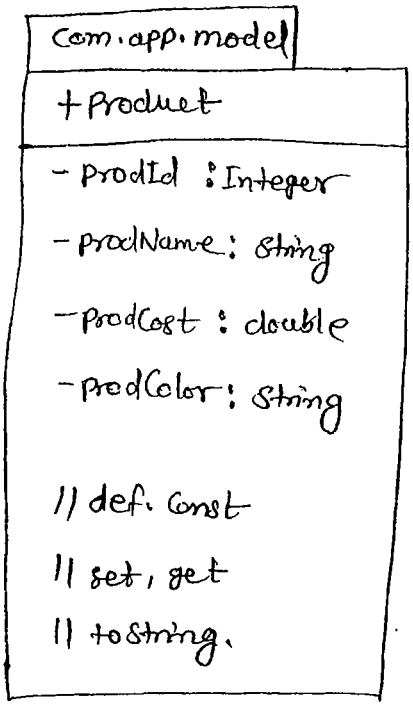
02/03/2019

Spring Boot Data JPA Module Design :-

required :-

1. Database (Using Embedded : H2)
2. Model class : Product (C)
3. Repository : ProductRepository
4. Runner : ConsoleRunner





T=? = model className = Product

ID=? = PK Datatype = Integer

PK Prodtab

pid	pname	mrp	color

*) Primary key Data Type must be a wrapper class ~~or~~ or any other class which implements java.io.Serializable

*) Primitive Types are not accepted as PK DataTypes for models and for Repository coding.

--Eclipse shortCuts ----------

F3 → go to code

ctrl + O → overview (press again for super type also)

ctrl + alt + DownArrow → copy current line, paste

Select lines:

+ ctrl + shift + / → comment lines

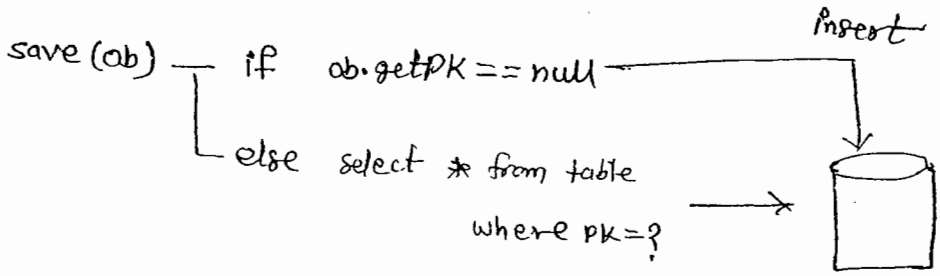
+ ctrl + shift + \ → uncomment lines

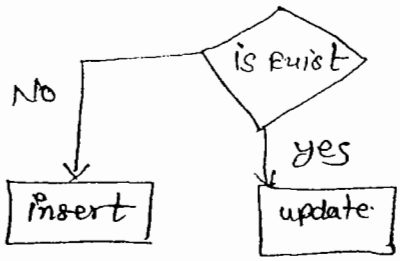
~~save(ob) if ob...~~

save(T ob) : T ;

This method is from CrudRepository (I) which is used to perform Save or update operation.

If primary key value is NULL or not exist in DB then performs insert operations, else record found in DB based on PK then performs update operation.





05/03/2019

Setup : Create project with dependencies H2, JPA, WEB

Step#1 add in application.properties

```
server.port=9898
spring.h2.console.enabled=true
spring.h2.console.path=/h2
spring.jpa.show-sql=true
```

Step#2 Define model class

```
package com.app.model;
//ctrl+shift+O(imports)
@Entity
@Table(name="prodtab")
public class Product {
  @Id
  @GeneratedValue
  @Column(name="pid")
  private Integer prodId;
  @Column(name="pname").
  Private String prodCode;
```

```
    @Column(name="Pcost")
     private double prodCost;

  //generate const, set|get, tostring....
}
```

Step #3 write Repository Interface

```
package com.app.Runner;
// ctrl + shift + o (imports)

@Component
public class BasicOperations implements CommandLineRunner {

    @Autowired
    private ProductRepository repo;

    public void run(String ... args) throws Exception {

  /* * * * * SAVE / UPDATE * * * * * */

    repo.save(new Product("PEN", 12.36));
    repo.save(new Product("CAR", 202.86));
    repo.save(new Product("MARKAR", 68.47));

  /* * * * * FIND * * * * * */

  Optional<Product> p = repo.findById(3);
  if(p.isPresent()){
   Sop(p.get());
  }
```

```
        else {
            sopln(" No data found");
        }

        repo.findAll().forEach(System.out::println);
/***********DELETE**********/
        repo.deleteById(4);
        // repo.deleteAll();
        repo.deleteAllInBatch();
        // This will stop server
        System.exit(0);
    }
}
```

**) Start App > Go to browser and enter url:

localhost:9898/h2 > click on Console

Method Description :- ———

① save(obj) :-

behaves like save or update, If PK exist in DB table then "UPDATE" else "INSERT"

② findById(ID) : Optional<T>

It will return one row as one object based on primary key in Optional<T> format.

⇒ use methods isPresent() to check, record is exist or not? if exist use method get() : T to read object.

③ findAll():

It returns collection of objects (=no. of rows is DB Table)

In Simple select * from tableName;

④ deleteById(ID):

To delete one Row based on PK

⑤ deleteAll():

To delete all rows [one by one row]

⑥ deleteAllInBatch():

To delete All rows at a time

ex: delete from <tableName>

---

*) H2 is a Embedded Database provided by SpringBoot which uses "hbm2ddl.auto = create-drop", which means create table when server/app starts and drop all tables when Server/app stopped.

*) H2 Console works only if use webapp & default path is /h2-console, default port: 8080

06/03/2019

Using MySQL DB :-

Setup: Create one Spring boot project with JPA and web Dependencies -

⇒ open pom.xml and add dependency for MySQL based on version installed.

```
<dependency>
  <groupId> mysql </>
  <artifactId> mysql-connector-java </>
  <version> 5.1.6 </>
  <scope> runtime </>
</dependency>
```

---- application.properties ----

```
## Server Details ##
server.port = 9898
## Datasource Details ##
spring.datasource.driver-class-name = com.mysql.jdbc.Driver
spring.datasource.url = jdbc:mysql://localhost:3306/test
spring.datasource.username = root
spring.datasource.password = root
```

Hibernate Details

spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL55Dialect

spring.jpa.show_sql = true

spring.jpa.hibernate.ddl-auto = update

------ Code ------

① Model class

```
package com.app.model;
import javax.persistence.*;
@Entity
public class Product{
   @Id
   @GeneratedValue
   private Integer prodId;
   private String  prodCode;
   private Double  prodCost;
   private String  vendorCode;
//def, param const, set/get, toString...
}
```

② Repository

```
package com.app.repo;
//ctrl+shift+o
public interface ProductRepository extends JpaRepository<Product, Integer>
{ }
```

---

③ Runner class

```
Package com.app.runner
//ctrl+shift+o
@Component
public class FindAllTestRunner implements CommandLineRunner
{
@Autowired
private ProductRepository repo;

public void run(String ... args) throws Exception {

Sopln("/*********/#");

repo.findAll().forEach(System.out::println);

Sopln("/*******/");

Product p= new product();

p.setVendorCode("V1");

//P.setPrice(25.5);

Example<Product> ex = Example.of(p);
```

```
        repo.findAll(ex).forEach(System.out::println);
    Sopln("/* * * * * * * * * */");
    // Default order is ASC
    repo.findAll(Sort.by(Direction.DESC,"prodCode")).forEach(System.
                                                    out::println);

    Sopln("/==========/");
    // pageNumber, pageSize
    PageRequest pageable = pageRequest.of(0,4);
    repo.findAll(pageable).forEach(System.out::println);
    Sopln("/@@@@@@@@@@/");
    // example and sort
    repo.findAll(ex, Sort.by(Direction.DESC,"prodCost")).forEach(
      System.out::println);
    Sopln("/* * * * * * * * * */");
    // example, pageable
    repo.findAll(ex, pageable).forEach(System.out::println);
    System.exit(0);
  }
}
```

① findAll() :

This method will select all rows and all columns from DB Table

[One Row = one object ⇒ finally List<T>]

② findAll(Example<S>) :

This method is used to fetch required rows from Db table, by taking some data as Object and compare with every rows, If matched select else ignore row.

** Compares only not null variable values with columns data.

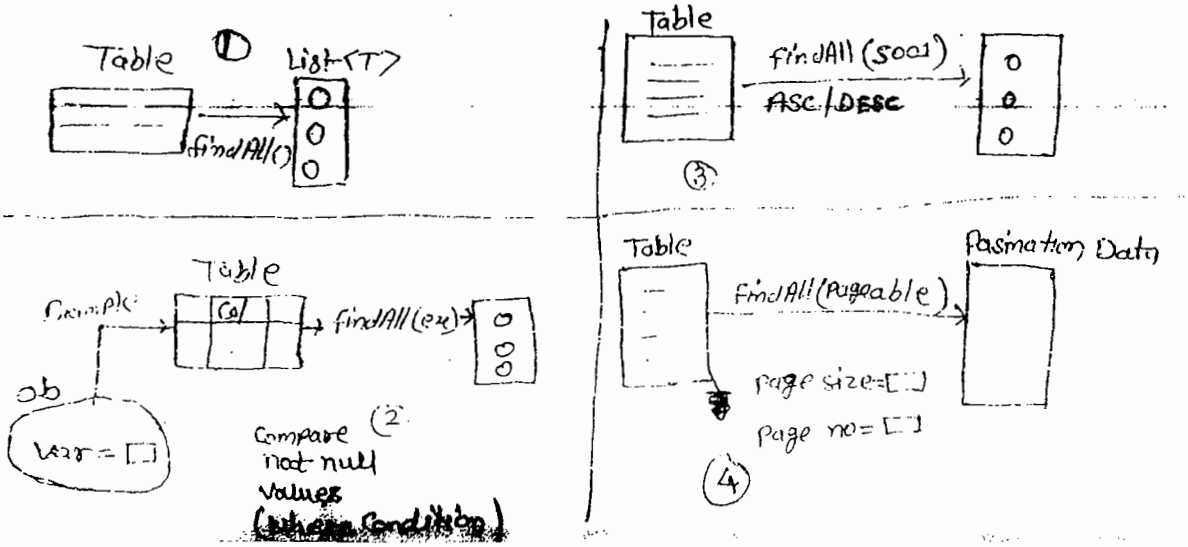
③ findAll(Sort) :

To fetch all rows based on "order by" condition, default is ASC.

Use Direction enum (ASC/DESC) to provide external order for property/column.

④ findAll(Pageable) :

This is to fetch data by using inputs pagesize and PageNumber (pagination). Pageable is interface. So, use PageRequest(C) Impl class.



Query Methods :- in Spring Boot Data :- 08/03/2019

Spring Data Generates a query based on method written in Repository by Programmer

Types of Query methods (3) :-

(a) findBy

(b) @Query(manual query)

(c) Special Parameters/ReturnTypes

*) These are used to specify our columns (projections) and rows (restrictions) details.

(a) findBy :-

It will Generate Select query based on abstract method given by programmer. we can provide columns and rows details.

=> It will be Converted to equal SQL query based on Database at runtime.

---Syntax---

```
RT findBy__(Parameters ...);
```

Here, RT = ~~ReturnType~~,

Ex:- List<T>, T, Object, Page<T>, Slice<T>, ~~Object[]~~, Object[], specific projection etc.

Example:- (add below methods in Repository)

```
// Select * from prodtab where prod_code = pc
Product findByProdCode(String pc);

// Select * from prodtab where prod_code like pc
List<Product> findByProdCodeLike(String pc);

// Select * from prodtab where prod_code is null
List<Product> findByProdCodeIsNull();

//select * from prodtab where prod_cost = cost
List<Product> findByProdCostGreaterThan(Double cost);

// select * from prodtab where prod_cost in (cost)
List<Product> findByProdCostIn(Collection<Double> costs);

// select * from prodtab
                    where pid = ?
                       or pcost = ?
List<Product> findByProdIdOrProdCost(Integer pid, Double cost);

// Select * from prodtab where pid between pid1 and pid2
List<Product> findByProdIdBetween(Integer pid1, Integer pid2);
```

// select * from prodtab where P_Cost=? order by Prod_Code asc

List<Product> findByProdCostLessThanOrderByProdCode(Double code);

// Select * from prodtab where pid<=? and pcost>=? and vendor is not null order by pcode;

List<Product> ~~findByProdPidAndProdCost~~
findByProdIdLessThanAndProdCostGreaterThanAndVendorCodeIs
NotNullOrderByProdCode(Integer pid, Double pcost);

Download and Install MongoDB (Local)

url: https://www.mongodb.com/download-center/community

Details:-

Version : 3.6 or 3.4

Type: MSI (Do not use zip)

⇒ While Installing Full (Complete) Type

⇒ After install create one folder in "C" driver.

(Create folder data >> Create folder db

Like: c:/data/db

=> Set path to MongoDb

Location: c:\Program Files\MongoDB\Server\3.4\bin

=> open cmd-1 and type >mongod (to start server)

=> open cmd-2 and type >mongo (to open and work on DB

09|03|2019

@Query("hql") :-----

This annotation is used to perform HQL (Hibernate Query Language) operations work for both select and non-select operations.

=> To pass where clause inputs use positional parameters in style

?1, ?2, ?3, ... (or)

named parameters in style [:name]

:a, :b, :hi, :hello, :mydata

*** But Variable name must be same as named parameter.

=> Providing package name for model class in @Query is optional.

ie: select p from com.app.model.Product,

select p from Product (are same)

(add below methods in ProductRepository)

// ~~select * from prodtab~~

// where ven_code = ? or prost >?

@Query("select P from Product p where p.vendorCode = :a or P.prodCost >:b")

List<Product> getAllProducts(String a, Double b);

// select pcode from prodtab where vencod=? or pcost=?

@Query("select p.prodCode from product p where
p.vendorCode=?1 or p.prodCost=?2")

List<~~Product~~ String> getAllProductsCodes(String a, Double b);

// Select _____

@Query("select p.prodCode, p.prodCost from Product p")

List<Object[]> getAllProductData();

⇒ CommandLineRunner code for getAllProductData() method

// JDK 1.5

```
List<Object[]> obs = repo.getAllProductData();
  for(Object[] ob:obs) {
    Sopln(ob[0]+","+ob[1]);
}
```


// JDK 8 (Streams + Lambda + method Reference)

```
repo.getAllProductData()
.stream()
.map((ob)->ob[0]+","+ob[1])
.forEach(System.out::println);
```

@Query(" ") non-select operations :—

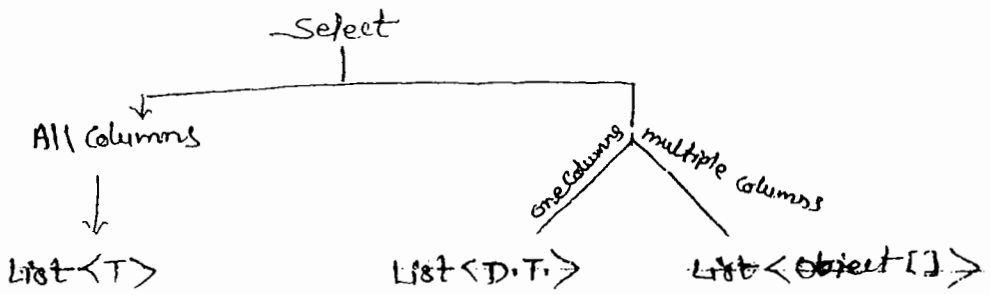
To perform non-select operation define HQL (@Query) and apply @Modifying and @Transactional over query method.

⇒ @Transactional can be applied over repository method or in service layer method.

------ProductRepository code ----------

```
@Modifying //non-select operation
@Transactional //service layer
@Query("update Product p set p.prodCode = :a, p.prodCost = :c
        where p.prodId = :b")
Void updateMyData(String a, Double c, Integer b);

@Modifying // non-select operation
@Transactional // service layer
@Query("delete from Product p where p.prodId = :prodId")
Void deleteMyData(Integer prodId);
```



T = model class = Ex: Product

D.T. = DataType of variable = Ex: String for prodCode.

> Notepad > type mongod > save with .bat

ex: "mongo-server.bat"

>notepa > type mongo >save with .bat

ex: "mongo-client.bat"

11/04/2019

Projections Using Query Methods :—

By default every query method (findBy) returns all columns (full loading)

⇒ Here, Projections are used to select few columns (variables).

⇒ It is two types:

ⓐ Static Projections

ⓑ Dynamic Projections

ⓐ Static Projections :-

This concept is used to for always fixed type of columns selection for multiple runs (or calls).

Steps to implement Static Projections:

#1) Define one child interface (inner interface) in Repository Interface with any name.

#2) Copy variables equal getMethods() from model class to child interface.

#3) use this return type for findByMethods()

Format:

```
interface Repository extends JpaRepository...
{
```

```
        interface <childType>
        {
          DataType   getVariable();
          DataType   getVariable();
        }
    List<childType> findBy__(...);
}
```

Example: (1) Repository :—

```
package com.app.repo;
// ctrl+shift+O (imports)
public interface ProductRepository extends JpaRepository<Product, Integer>
{
   interface MyView
   {
     String ~~getProduct~~ getProdCode();
     Double getProdCost();
   }
   // static projections
  // select code, cost from ... where ven...code = ?
   List<MyView> findByVendorCode(String vc);
}
```

CommandLineRunner code :—

```
repo.findByVendorCode("V2")
.stream()
.map((p) -> p.getProdCode() + ", " + p.getProdCost())
.forEach(System.out::println);
```

(b) Dynamic projection's :-

In this case findBy() method return Type is decided at runtime (It is generic)

Format :

```
<T>  List<T> findBy_(..., Class<T> cl);
```

Here, T = child interface Type, provided at runtime.

Repository code :-

```
public interface ProductRepository extends JpaRepository<Product, Integer>
{
  interface MyViewTypeTwo
   {
     Integer getProdId();
     Double  getProdCost();
   }
  interface MyView
  {
    String getProdCode();
    Double getProdCost();
  }
<T> List<T> findByVendorCode(String vc, Class<T> cl);
}
```

CommandLineRunner code :-

```
repo.findByVendorCode("V2", MyViewTwo.class)
.stream()
.map(ob -> ob.getProdId()+", "+ob.getProdCost())
.forEach(System.out::println);
```

* * *

---***---

```
repo.findByVendorCode("V2", MyView.class)
 .stream()
 .map(a) -> a.getProdCode() + "," + a.getProdCost())
 .forEach(System.out::println);
```

⇒ Dynamic projections are show compared to static projections.

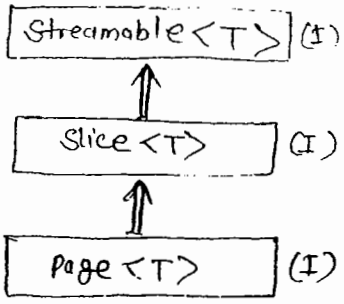
⇒ For Dynamic projections "Type select, ~~Variable~~ validate and execute" done at time, for static select and validate done at compile time.

⇒ Static projections are also called as compile time (selected) projections and Dynamic projections also called as Runtime (selected) Projections.

12|03|2019

Special Types in Query Methods :-

1. Page<T>
2. Slice<T>
3. Stream<T>



⇒ These are ~~special Type outputs for query methods to get~~ pagination data.

1. Streamable :

It will help to return collection data (one page only) in Stream<T> (java.util) format, where can apply functions like filter, sort, map and collect ... etc.

2. Slice :—

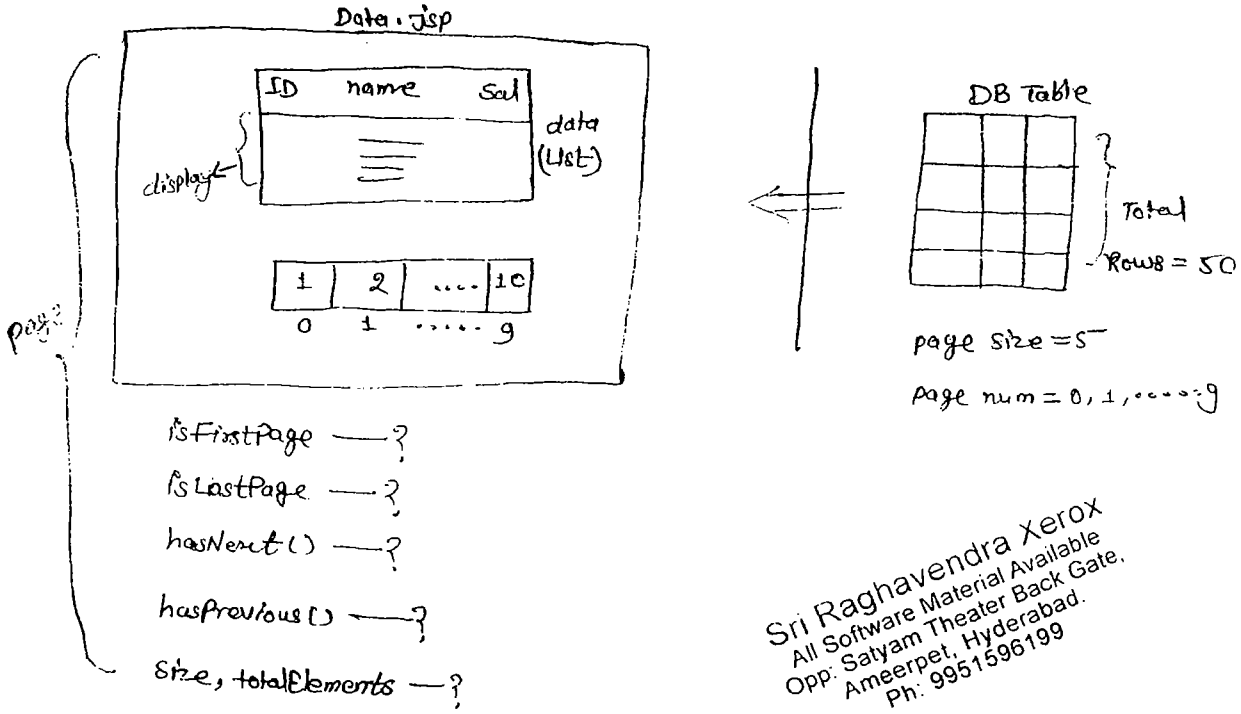
It is used to provide current page data and links with ~~previous~~ previous and next page details.

⇒ It never holds all page's details like total no. of pages and total elements.

3. Page :—

It is extension to Slice and holds even total no. of pages (links to other pages) and total elements (total row is details).

Example Details :—



Page (and Slice) :—

Step #1 Insert few records in db (ex: 20 rows)

Step #2 Define a Repository method with Pageable as parameter (it is last param).

Ex: Page<Product> findByVendorCode(String vc, Pageable pageable);

Step #3 : In Runner class call method and print details like, isEmpty ? , first ? , last ? , ... etc.

----Code :----

#1) Repository Interface :-

```
package com.app.repo;
//ctrl +shift +o (imports)
public interface ProductRepository extends JpaRepository<Product, Integer>
{
  Page<Product> findByVendorCode(String vc, Pageable pageable);
/* Slice<Product> findByVendorCode(String vc, Pageable pageable); */
}
```

#2) Runner class code :-

```
Page<Product> page = repo.findByVendorCode("V1", PageRequest.of(0,3));
Sopln("Empty Page ? : " + page.isEmpty());
Sopln("First Page ? : " + page.isFirst());
Sopln("Last Page ? : " + page.isLast());
Sopln("Next Page ? : " + page.hasNext());
Sopln("Previous page ? : " + page.hasPrevious());
// not for Slice
Sopln("Total Page ? : " + page.getTotalPages());
// not for Slice
Sopln("Total Rows ? : " + page.getTotalElements());
// total rows data
Page.forEach(System.out::println);
```

*) Streaming is a process of :

read → filter → Sort → map → Collection over collection data which Stream's concept given in JDK 1.8

⇒ Streams are used to execute series of operations in one or less Iterations, which reduces execution time compare to normal programmer iterations.

---Code :---

#1) Repository code (call in ProductionRepository) :-

```
Streamable<Product> findByVendorCode(String vc);
```

#2) Runner Code :-

```
Streamable<Product> p = repo.findByVendorCode("V1")
p.stream()
.filter((pob) -> pob.getProdCode() != null)
.sorted((p1,p2) -> p1.getProdCode()
.compareTo(p2.getProdCode());
.map((pa) -> "code" = " + pa.getProdCode() + ", cost =" + pa.getProdCost()
.forEach(System.out::println);
```

Connection Pooling in Spring Boot Data :—

Spring Boot 1.x works on Tomcat connection pooling as default and Spring Boot 2.x works on Hikari connection pooling which is very fast in service.

*) This is not required to be configured by developer. By default Spring Boot configures using class input 'HikariConfig' with all default values (in it self).

*) To provide Programmer specific values use key ~~string~~ :
spring.datasource.hikari.* in application.properties.

--- application.properties ---

spring.datasource.hikari.connection-timeout = 20000

spring.datasource.hikari.minimum-idle = 5

spring.datasource.hikari.maximum-pool-size = 12

spring.datasource.hikari.idle-timeout = 300000

spring.datasource.hikari.max-lifetime = 12000

spring.datasource.hikari.auto-commit = true

Task :- Write Spring Data JPA application properties file in yml format.

13/03/2019

Chapter - 3

SpringBoot NoSQL Database [Mongo DB] :-

Mongo DB is a simple and open source document database and leading NoSQL Database.

⇒ MongoDB can be integrated with spring Boot using Dependency:

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

*) Boot also supports ~~Enable~~ Embedded Mongo DB for ~~easy~~ easy coding and Testing Process.

```
<dependency>
    <groupId> de.flapdoodle.embed </groupId>
    <artifactId> de.flapdoodle.embed.mongo </artifactId>
    <scope> test </scope>
</dependency>
```

*** Remove scope = test for using same for development process, if we have no installed DB.

*) Keys you need to add in application.properties for non-embedded mongo are:

```
spring.data.mongodb.host = localhost
spring.data.mongodb.port = 8707
spring.data.mongodb.database = sathya
```

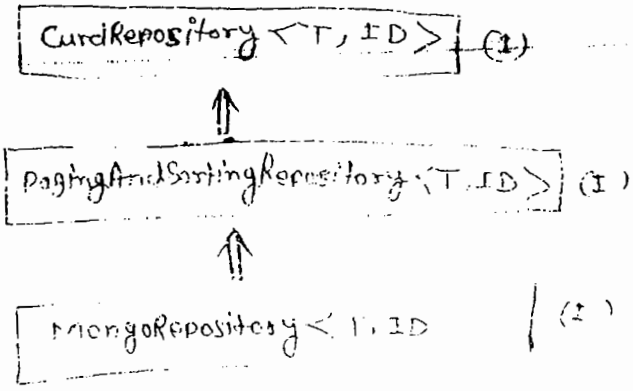

Download location of Mongo DB :---

https://www.mongodb.com/download-center/community

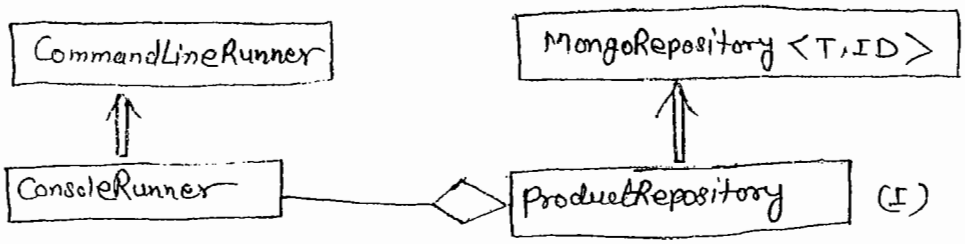
> choose version, os and package and download.

*) Before starting Mongo server and client you need to create a folder in "C" drive as: C:/data/db

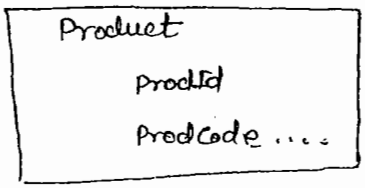
Design #1 : Spring Boot Mongo DB Repository levels



Design #2 : Spring Boot Mongo DB coding files :



DOCUMENT



MongoDB :-

It is a NoSQL DataBase which holds data in Collections format.

⇒ Every collection (similar to table, but not table) holds data in JSON objects (similar to rows, but not rows).

⇒ JSON (Java script object Notation) format looks as :

{ " key " : value, ... }.

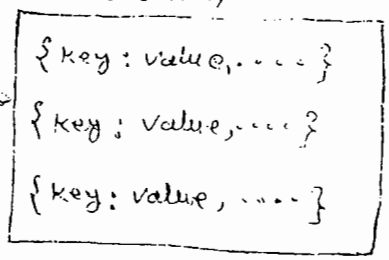
⇒ Every collection also called as Document [word used in Programming language].

*) Collection Format :

[Document]

Collection

JSON object



Example Collection :—

Student

{ "stdId" : 1 , "stdName" : " ANIL " , "stdfee" : 2.6 },

{ "stdId" : 2 , "stdName" : "SATHYA" , "stdfee" : 3.4 },

{ "stdId" : 3 , "stdName" : " CHIKKU" , "stdfee" : 4.8 }

Step#1 : After Intalling Mongo DB , set path location

c:/programfiles / mongoDB /Server/3.4/bin

Step#2 : Start Server using Command " Mongod "

d:/user/ >mongod

Step#3 : Start client using Command " Mongo " .

d:/user/mongo

Step#4 : Work on database

View all databases

>show databases

Create one new database /use old

>use sathya

Create one collection / insert one row

>db.student.insert ({ "std" : 1 , "stdName" : "ANIL" , "stdfee" : 3.2 })

View all Collections in Current DB .

>show collections

View data in one collection

```
>db.student.find() (or)
>db.student.find().pretty()
>db.student.find({"stdId":8.4}).pretty()
```

Delete one collection object

```
>db.student.remove({"stdId":8.4})
```

---- Coding :----

Step #1 Create one Spring starter project

Ex: SpringBootMongoDB

Dependencies: Mongo DB, Embedded MongoDB and Web

Step #2 : Design one document (collection) class

```
package com.app.document;
//ctrl + shift + o (imports)
@Document
public class Product
{
 @Id //Id-type must be String - UUID
 private String ProdId;
 private String ProdName;
 private Double ProdCost;
 private String Vendor;
//default and parameterized constructors (without PK)
//set/get & toString()
}
```

Step #3: Define one Repository interface

```
package com.app;
//ctrl+shift+O (imports)
public interface ProductRepository extends MongoRepository<Product, String>
{
}
```

Step #4: Define Runner class

```
Package com.app.runner;
//ctrl+shift+O (imports)
@Component
public class ConsoleRunner implements CommandLineRunner
{
 @Autowired
 private ProductRepository repo;
public void run(String.... args) throws Exception
{
 repo.deleteAll();
 repo.save(new Product("CAR", 6.5, "V1"));
 repo.save(new Product("PEN", 7.6, "V1"));
 repo.save(new Product("Book", 8.9, "V1"));
 repo.findAll().forEach(System.out::println);
 System.exit(0);
}
}
```

Step #5: Remove <scope> test </scope> for Embedded Mongo dependency in pom.xml (or) to work on ~~extend~~ external (Install) MongoDB.

> Delete/Comment Embedded Mongo dependency in pom.xml

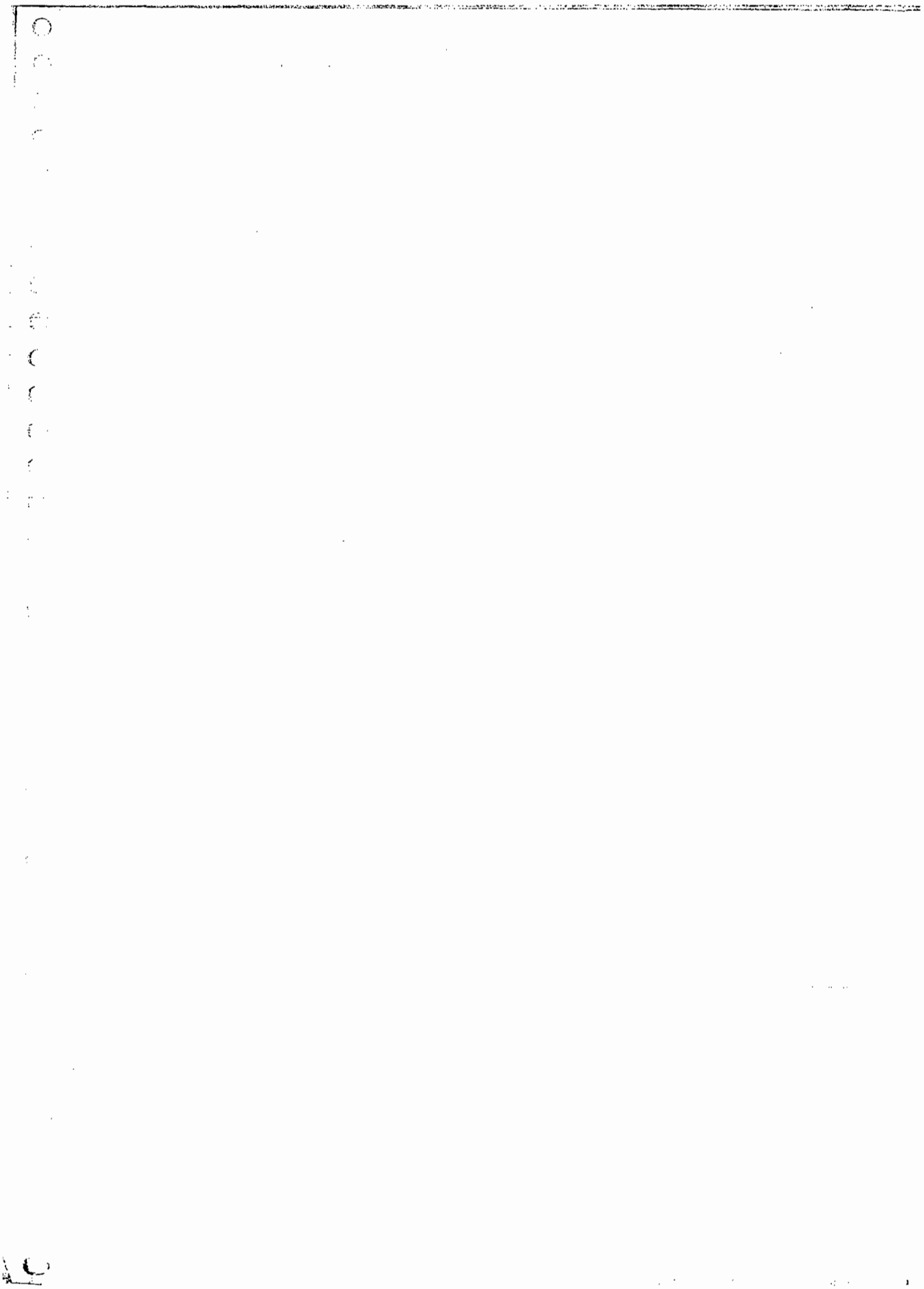
> add keys in application.properties

```
# Mongo External Details
spring.data.mongodb.host = localhost
spring.data.mongodb.port = 27017
spring.data.mongodb.database = sathya
```

---

* Equal application.yml file is:-

```
server:
    port: 3097
spring:
    data:
        mongodb:
            host: localhost
            port: 27017
            database: sathya
```



Sri Raghavendra Xerox
All Software Material Available
Opp: Satyam Theater Back Gate,
Ameerpet, Hyderabad.
Ph: 9951596199

14/03/2019

Chapter #4 Spring Boot - Spring WEB MVC

- Dependency Details :

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

*) In spring boot coding is similer to spring WEB MVC only, but boot provides "Required JARS, Server setup, App plugins, Common Code" which is also called as AutoConfiguration when we add above dependency to pom.xml else not.

#1 Spring Boot provides 3 embedded Servers
Tomcat, Jetty and Undertow. default is Tomcat

#2 Default port Number is 8080 (server.port)

#3 FrontController named as "DispatcherServlet" is configured by Spring boot with URL-Pattern = /

#4 Context-path (projectName) also ~~called~~ mapped to /

#5 Handler mapper (DefaultAnnotationHandlerMapper) also configured by Spring boot.

#6 Component Scan (basePackage) is ~~set~~ to starter class package name by Spring boot.

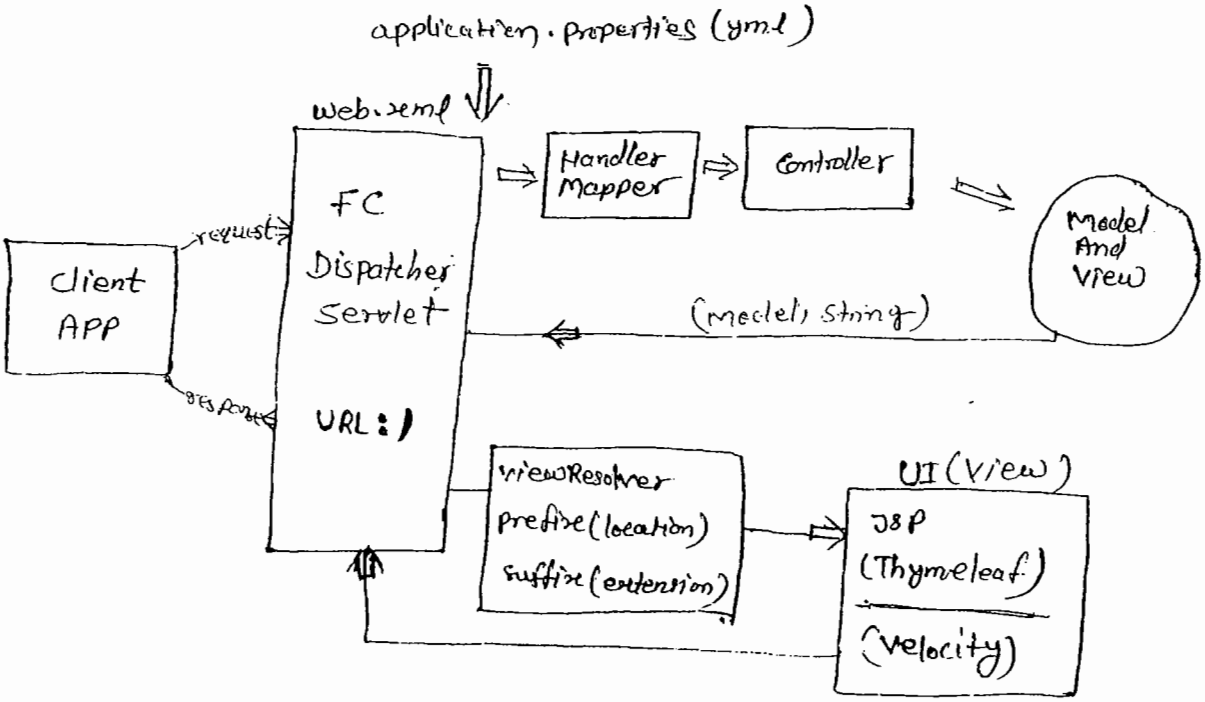
#7 ViewResolver is autoconfigured by spring boot, but programmer has to provide input using Properties/yml file.

#8 All WEB MVC Required JARs are set to project by Spring boot

#9 Boot provides Maven plugin for "JAR/WAR" packing for cloud Deployment.

#10 Spring Boot - spring WEB also supports RestFul webservices also.

Spring Boot - Spring WEB MVC



Note :

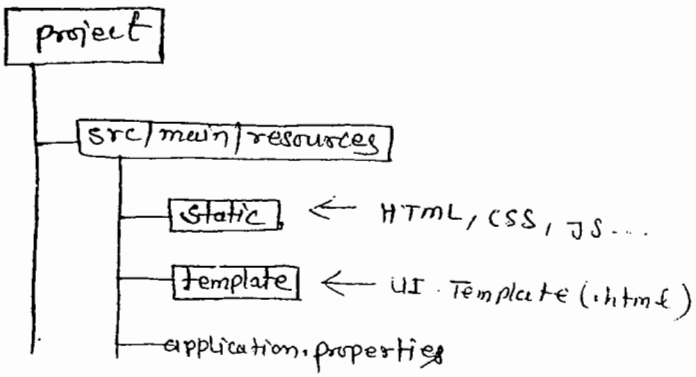
a. Spring Boot creates container using New Container Style ie ApplicationContext for non Web/Web services Applications

Impl class is : AnnotationConfigApplicationContext

for Web Related Applications Impl class is :

AnnotationConfigServletWebServerApplicationContext

b. Default Static Resource Handler is added to folder static and template which are provided under src/main/resources folder



Example App #1 - - - - - - - - - -

Step#1 create new spring starter App with Web Dependency and also add below one for JASPER ENGINE.

```
<dependency>
    <groupId>org.apache.tomcat.embed</groupId>
    <artifactId>tomcat-embed-jasper</artifactId>
</dependency>
```

** Embedded Tomcat comes with only lightweight servlet Engine.
So, JSP Translate will not work hence show error.

Step #2 add details in application.properties

```
# ServerSetup
server.port=9898
# View Resolver setup
spring.mvc.view.prefix=/WEB-INF/views/
spring.mvc.view.suffix=.jsp
```

Step #3 Create folder(s) /Webapp/WEB-INF/views under src->main folder

Step #4 create one jsp (Home.jsp) under views

--- Home.jsp ---

```
<%@ page isELIgnored="false" %>
<html> <body>
    <h3> Welcome to Home page !! </h3>
    ${msg}
</body> </html>
```

Step #5 Define Controller class

```
package com.app.controller;
//ctrl+shift+o
```

```
@Controller
@RequestMapping("/home")
public class HomeController {
@RequestMapping("/show") // GET
public String showHome(Model model) {
    model.addAttribute("msg", new Date());
    return "Home";
}
}
```

Step #6 Run Starter class and enter url

http://localhost:9898/home/show

15/03/2019

~~Working~~

Spring Boot Web Applications - working on Embedded Servers :-

By default when we add dependency spring-boot-starter-web then Boot also provides Embedded Default Server "Tomcat" (Apache Product)

⇒ It works on port number 8080 by default

⇒ Embedded Tomcat is a light weight engine which contains only Catalina (Servlet Engine). No JASPER (No JSP Engine).

⇒ Two more Embedded servers are Jetty (Eclipse [GlassFish] Product) and Undertow (JBoss product)

#1 Working with Jetty Server

⇒ Create one Starter Application with web dependency

⇒ create one index.html file under static folder

⇒ ** Exclude Tomcat first before adding any other Server.

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

⇒ Now include (add dependency for) Jetty Server

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

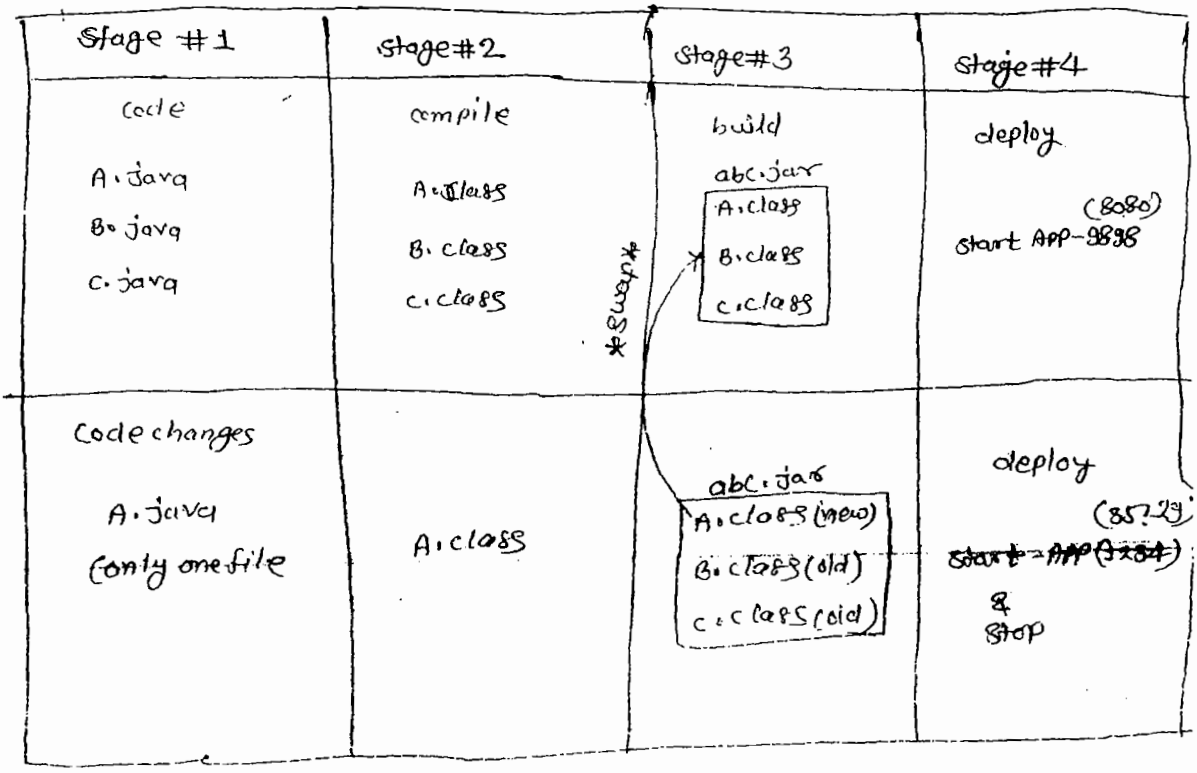
⇒ To use Undertow Server include (add dependency for)

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

DevTools :—

This is also called as Developer Tools used at DEV environment mainly To Avoid multiple re-starts of applications for even small changes.

⇒ Used mainly for HOT-DEPLOYMENT process. It means only copy file which are changed into current JAR/WAR without stopping Server.



** This swap process is known as Hot Deployment which is done using DevTools

** for this process we need to add dependency

```
<dependency>
    <groupId>org.springframework.boot</
    <artifactId>spring-boot-devtools</
    <scope>runtime</scope>
</dependency>
```

** DevTools will not support HotDeployment for few locations. These are:

META-INF/maven/**,
resources/**,
templates/**,
**/*Tests.class,
META-INF/build-info.properties
META-INF/resources/**,
static/**, public/**,
**/*Test.class,
git.properties,

=> bydefault all these places are under exclude locations.
(spring.devtools.restart.exclude)

=> To enable any specific location to be Hot deployed, then add that location using key: spring.devtools.restart.additional-paths

Example

spring.devtools.restart.additional-paths = static/**, templates/**

Spring Boot Application

Web MVC + Data JPA + UI Thymeleaf + Bootstrap + Hazalcast + MySQL DB

16/03/2019

Check Email (Application code)

18/03/2019

Thymeleaf UI Template :-

Thymeleaf UI is light weight Java UI Engine used to design Dynamic web pages in web applications.

⇒ JSP internally servlet (heavy weight) but Thymeleaf is simple (Core) Java code so, compared to JSP, Thymeleaf is lightweight (less memory)

⇒ Use Thymeleaf namespace in HTML file

<html xmlns:th = "https://www.thymeleaf.org/">

⇒ Every Tag or attribute related to Thymeleaf must start with prefize 'th'

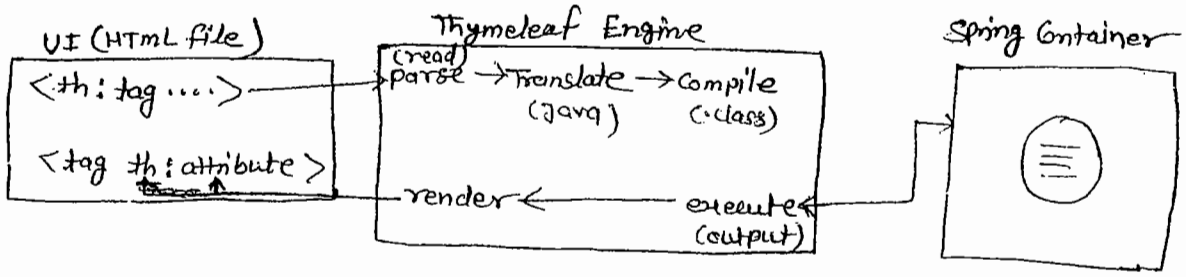
⇒ File must be saved with .html only under src/main/resources and inside 'templates' folder.

Step #1 write Thymeleaf tag/attribute in UI file.

Step #2 Thymeleaf Engine converts it into Simple Java code.

Step #3 This code will be Executed by Engine only and renders output back to HTML file.

Execution Flow of Thymeleaf Engine :-



#1 parse data from HTML (UI) file which has Prefix 'th' (No other tags like HTML or any)

#2 Translate tag/attribute into equal java code.

#3 Compile Generated Java code (.class)

#4 Execute code and comp communicate with Container to read or write data.

#5 Generated output will be placed back to UI file inplace of tag/attribute Known as Data-Rendering.

Thymeleaf Simple Codes :—

#a. Link CSS file with UI file

```
<link rel="stylesheet" th:href="@{/__/__.css}" >
```

#b. Link Javascript files with UI file

```
<script type="text/javascript" th:src="@{/__/__.js}" >
</script>
```

#c. Define one form

```
<form action="#" th:action="@{/url}"
      th:object="${modelClassObj}" >
```

#d. Display text taken from container

```
<span th:text="${message}" > </span>
```

#e. Input field (variable) linking

```
<input type="_" th:field="*{variable}" />
```

#f. Read List (collection) and print using for-each

```
<tr th:each="ob:${list}" >
  <td th:text="${ob.variable}" > </td>
  ....
</tr>
```

// Here, Symbol @ indicates URL upto contextpath.

Ex: @{/product/save} means https://localhost:8080/AppName/product/save

// Symbol ${ } indicates expression (may read data from memory),

// Symbol *{ } indicates pointing to variable (or any property)

Coding Order :—

1. application.properties
2. Model class
3. Repository
4. IService and ServiceImpl
5. Controller
6. UI File (.html)

Code given Email

Cache Management in Spring Boot :—

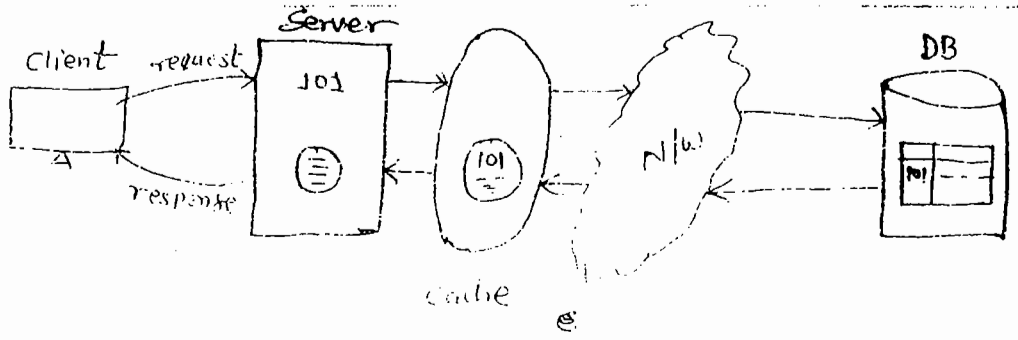
Cache is a temp. memory which exist at server side to hold ~~memory~~ commonly accessed data by client.

→ Cache concept is used to reduce network calls between Application (server) and Database.

→ Cache should never hold large data, apply only for few modules which are accessed more times by client.

→ *** Do not apply cache for findAll() type method.

——Design #1 ——



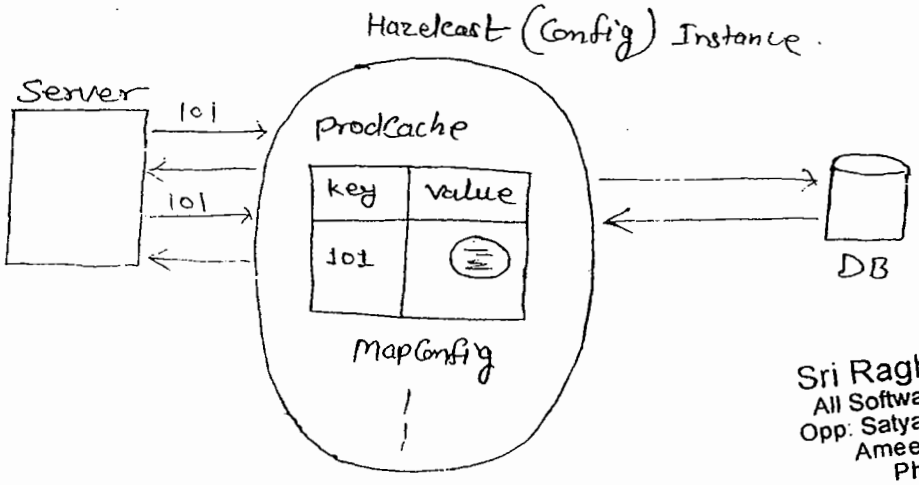
Hazelcast - cache Config:-

It is 3rd party cache configuration process, supported for any java application caching.

Step #1 we should create cache memory also called as Hazelcast Instance using class "Config" (com.hazelcast)

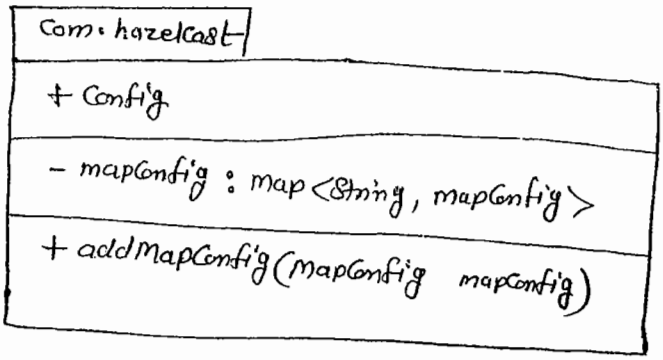
Step #2 for one module, provide one cache are known as MapConfig, which holds data in key (String-ID) and value (Object-data)

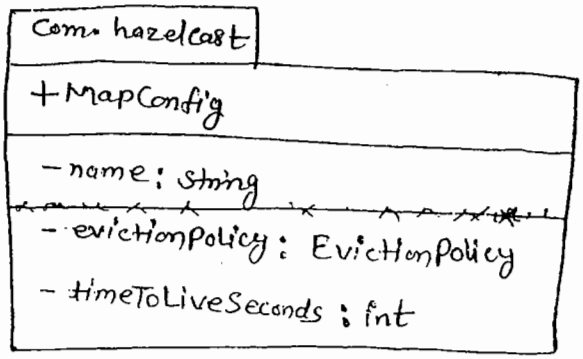
Design #2 overview for product



In application write Java Configuration code for Config() (com.hazelcast) and MapConfig() (com.hazelcast) classes

——— Design #3 UML Designs - - - - - - - -





19/03/2019

Step #1

```
<dependency>
    <groupId> org.springframework.boot </groupId>
    <artifactId>
</dependency>
<dependency>
    <groupId> com.hazelcast </groupId>
    <artifactId> hazelcast-spring </artifactId>
</dependency>
```

Step#2: Add below annotation at starter class level, to enable (disable) cache management:

```
@EnableCaching
```

Step#3: Add below annotations at service methods level

(a) over getOne(findOne) method

```
@Cacheable(value="cache-name", key="#PKID")
```

(b) over delete method (deleteById)

```
@CacheEvict(value="cache-name", key="#PKID")
```

Step#4 Define class for cache Configuration using classes Config and MapConfig.

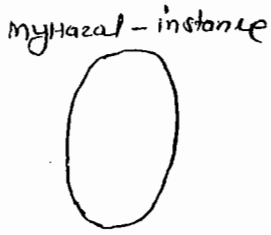
```
Package com.app.config;
//ctrl+shift+O
@Configuration
Public class HazalCastCacheConfig {
@Bean
Public Config cacheConfig() {
 return new Config().setInstanceName("Hazal-instance")
        .addMapConfig().setName("products-cache")
        .setMaxSizeConfig(new MaxSizeConfig(200,
                MaxSizePolicy.FREE_HEAP_SIZE))
        .setEvictionPolicy(EvictionPolicy.LRU)
        .setTimeToLiveSeconds(20000)
    );
}
}
```

Step #5 Make your class "implements Serializable" interface.

NOTE:-

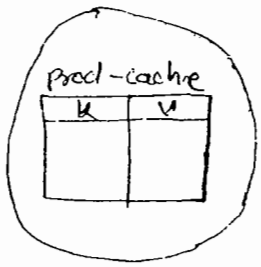
Step #1 Create cache memory and provide one name for that.

```
@Bean
public Config cacheConfig(){
  return new Config()
  .setInstanceName("myHazal-instance");
  ....
}
```



Step #2 Define one MapConfig memory for one module cache.

```
new MapConfig().setName("prod-cache");
```



Step #3 Provide Max size of cache (maximum no. of objects to be hold by Cache)

```
setMaxSizeConfig(new MaxSizeConfig(100, MaxSizePolicy.FREE_HEAP_SIZE
))
```

Step #4 Provide EvictionPolicy.

*) If cache is full and another object is in waiting state to get into cache then EvictionPolicy will not allow another obj'. (Default is : NONE)

*) EvictionPolicy.LRU : will remove last accessed object (Least Recently Used) from cache.

NONE : No Evit (No remove from cache)

LRU : Least Recently Used

LFU : Least Frenquently Used

RANDOM : Any random object to be removed

Step #5 Provide Life Time of Object to be in cache (in sec.)

~~setTimeLive~~ setTimeToLiveSeconds (3000),

@CacheEvit :-

This annotation is used to remove object from cache (not from DB),

⇒ on calling Service.delete...() method, SQL query deletes row only in DB but not in cache.

At same time object to be removed from cache, for that add this Annotation over delete(...) method in Service layer.

- @Cacheable :—

  on calling select.... SQL (load one row as object) same object will be placed in cache before give to server (APP).

Coding

#1. Add below method in ProductController

```
@RequestMapping("/one/{id}")
public String showOnePage(@PathVariable Integer id, Model model)
{
System.out.println("In Controller");
Product p = service.getProductById(id);
model.addAttribute("product", p);
return "productOne";
}
```

#2 Define HTML UI (productOne.html)

```
<html xmlns:th = "http://www.thymeleaf.org">
<body>
<h3> Welcome to product view one </h3>
<table border = "1">
<tr>
<td> ID </td>
```

```
<td th:text="${Product.ProdId}"></td>
</tr>
<tr>
  <td>NAME</td>
  <td th:text="${Product.ProdName}"></td>
</tr>
<tr>
  <td>COST</td>
  <td th:text="${Product.ProdCost}"></td>
</tr>
</table></body></html>
```

#3 Add below line in ProductData.html

```
<td>
<a th:href="@{/Product/one/{id}(id=${ob.ProdId})}">
<span class="btn btn-success">VIEW</span>
</a>
</td>
```

Chapter #5 Spring Boot Batch Processing :—

Batch processing is used to execute length (or heavy) tasks in steps by step.

⇒ Every task is called as JOB.

⇒ One JOB can contain one or more steps (Step can also called as Sub Task)

⇒ One Step contains

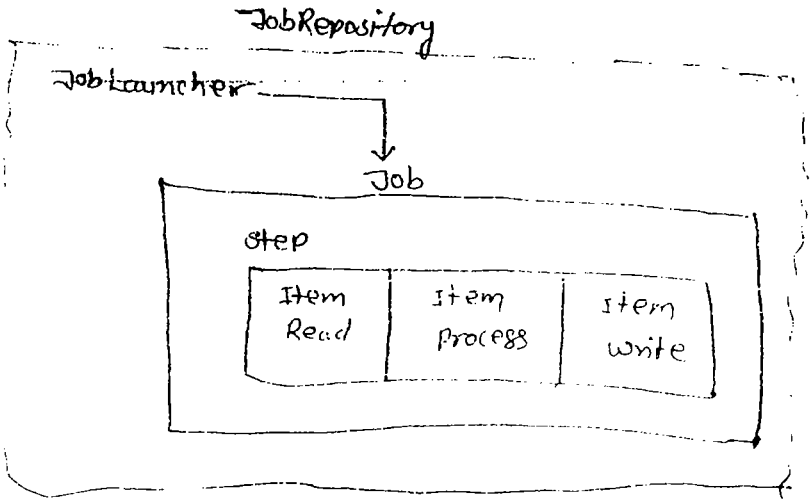
a. Item Reader (Read data from Source)

b. Item Process (Do calculations and Logics / operations etc...)

c. Item Writer (Provide output to next step or final output)

⇒ JOBs are invoked by JobLauncher also known as Job Starter

⇒ JOB, steps and Launcher details must be stored in JobRepository (config file).



1 Job = 1 Step

1 Step = 1 Read + Process + 1 write

20/03/2019

*) Step Implementation :— ******

In a Job (work) we can define one or multiple steps which are executed in order (step by step).

→ Job may contain 1 step, Job may contain 2 steps, ... Job may contains many steps so, finally 1 - Job = *(many) step

→ Every step having 3 execution stages

a. ItemReader<T>

b. ItemProcessor<I, O>

c. ItemWriter<T>

→ ItemReader<T> :-

is used to read data from source as input to current step.

→ ItemProcessor<I, O> :—

is used to process input data given by Reader and returns in Modified (or same) format of data.

→ ItemWriter<T> :—

is used to write data to any Destination (Ex: DB, Text File, CSV, Excel, xml ... etc)

NOTE:

1. An Item can be String, Array, Object of any class, Collection (List / Set ...)

2. ItemReader will read one by one Item from source. for example source has 10 items then 10 times ItemReader will be executed.

3. ItemReader GenericType must match with ItemProcessor Input GenericType.

4. ItemProcessor will read item by item from Reader and do some process (calculate, convert, check conditions... convert to any class object etc...)

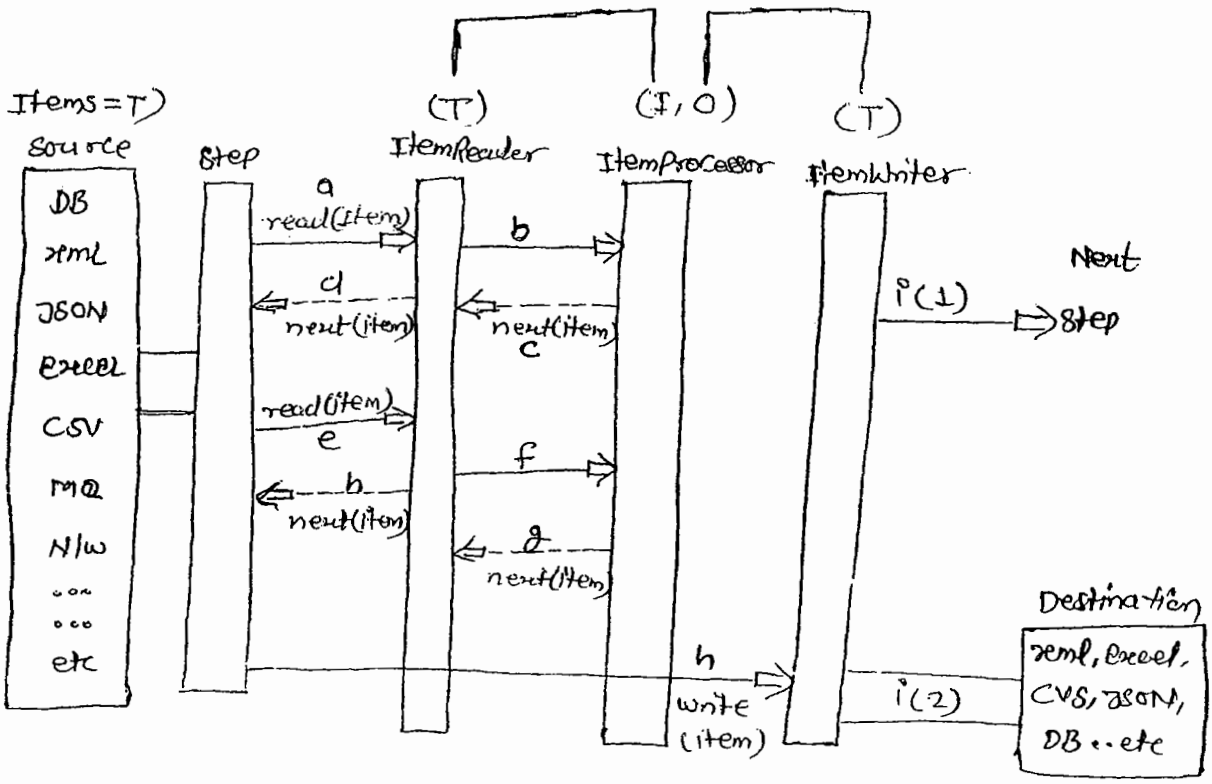
5. ItemProcessor will provide output (may be same as Input Type) which is called as Transformed Type.

6. ItemWriter will collect all output Item into one List Type from Processor at a time (only one time).

7. ItemWriter writes data to any destination.

8. Source/Destination can be Text File, DB, Network, MessageQueue Excel, CVS, JSON data, xml etc...

Execution Sequence of step : ----(next page)

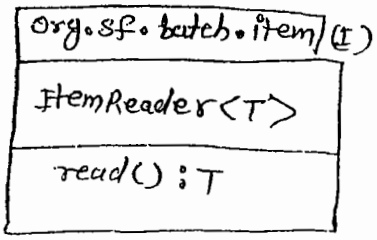


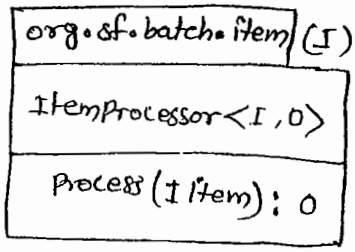
-> One step can be constructed using StepBuilderFactory (sf) class by providing name, chunk size, reader, Processor and writer.

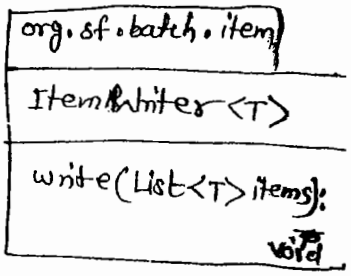
StepBuilderFactory (sf)

sf.get("step1")	provide name of step
• <String, String>chunk(1)	No. of items to be read at a time
• reader(readerObj)	Any Imple class of ItemReader<T>(I)
• Processor(ProcessorObj)	Any Impl class of ItemProcessor<I, O>
• writer(writerObj)	Any Impl class of ItemWriter<T>(I)
• build();	Convert to step (impl class) object

UML Notations :—







22/03/2019

JobExecutionListener (I) :—

This interface is provided Spring Batch f/w, which get called automatically for our Job.

⇒ for one Job — one Listener can be configured.

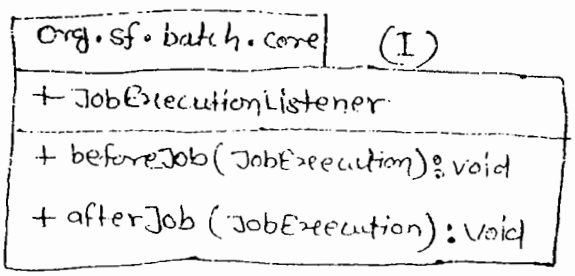
⇒ It is interface which has provided two abstract methods.

a. beforeJob(JobExecution) : void

b. afterJob(JobExecution) : void

⇒ JobExecution is class which is ~~called~~ used to find current Job details like JobParameters, BatchStatus, StepExecutions etc....

⇒ *** BatchStatus is enum having possible values : COMPLETED, STARTING, STARTED, STOPING, STOPPED, FAILED, ABANDONED, UNKNOWN.



JobBuilderFactory (C) :—

This ~~Job~~ Class is used to create one or more Jobs using Steps, Listener, Incrementer .....

⇒ Job (I) Construction Flow

JobBuilderFactory jf

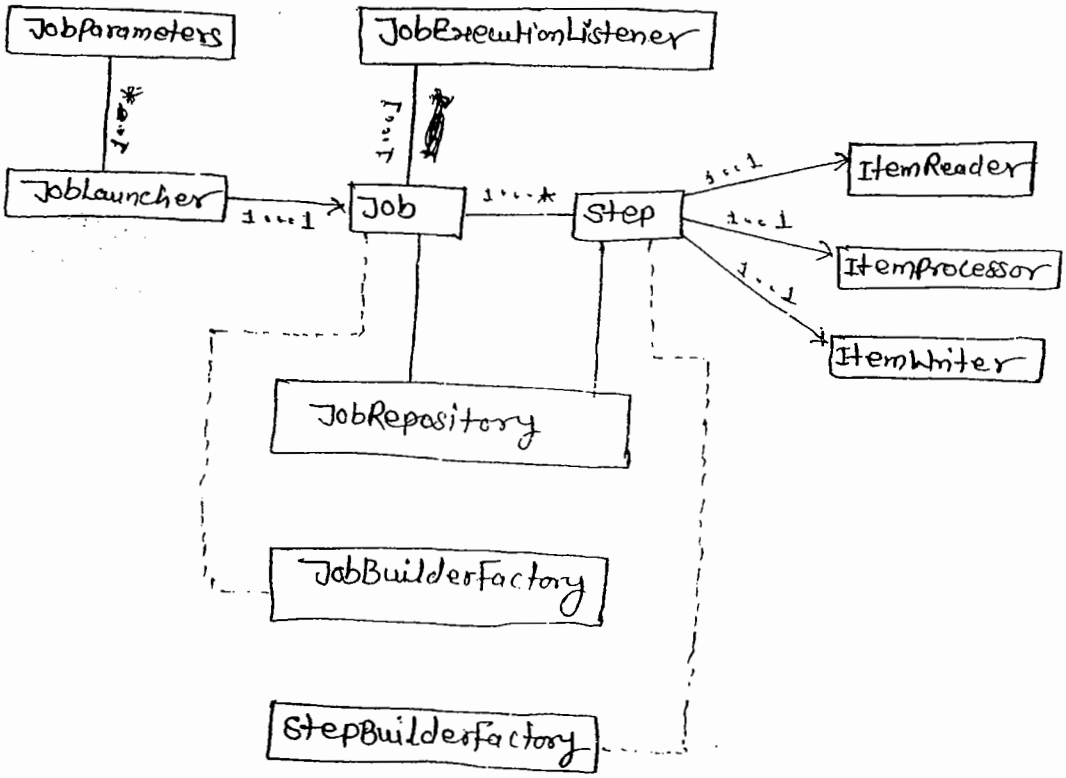
jf.get("jobA")	Job Name
.incrementer(runIdIncrementer)	incrementer
~~.incrementer~~ .listener(jobExeListener)	Job Execution Listener
.start(stepA)	First step
.next(stepB) .next(stepC) .next(stepD)	step in order
.build();	create Job Type

*) JobLauncher (I) :-

This interface is used to invoke our job with Parameters (inputs) like creationTime, uniqueId(name), programmerData etc...

*) This interface is having method run(Job, JobParameters)

*) ~~Job Parameter~~ JobParameters object is created by JobParametersBuilder which holds data in Map<key, value> style.



Code given Email

Coding order :—

1. Reader
2. Processor
3. Writer
4. Step configuration using StepBuilderfactory
5. JobExecutionListener
6. Job config — JobBuilderfactory
7. Jobparameters using JobParametersBuilder
8. JobLauncher using CommondlineRunner

*** Add key in application.property

spring.batch.job.enabled = false

To avoid execution of job multiple times (by starter class)

23/03/2019

Spring Boot Batch Processing : Converting .csv Data to DataBase Table :—

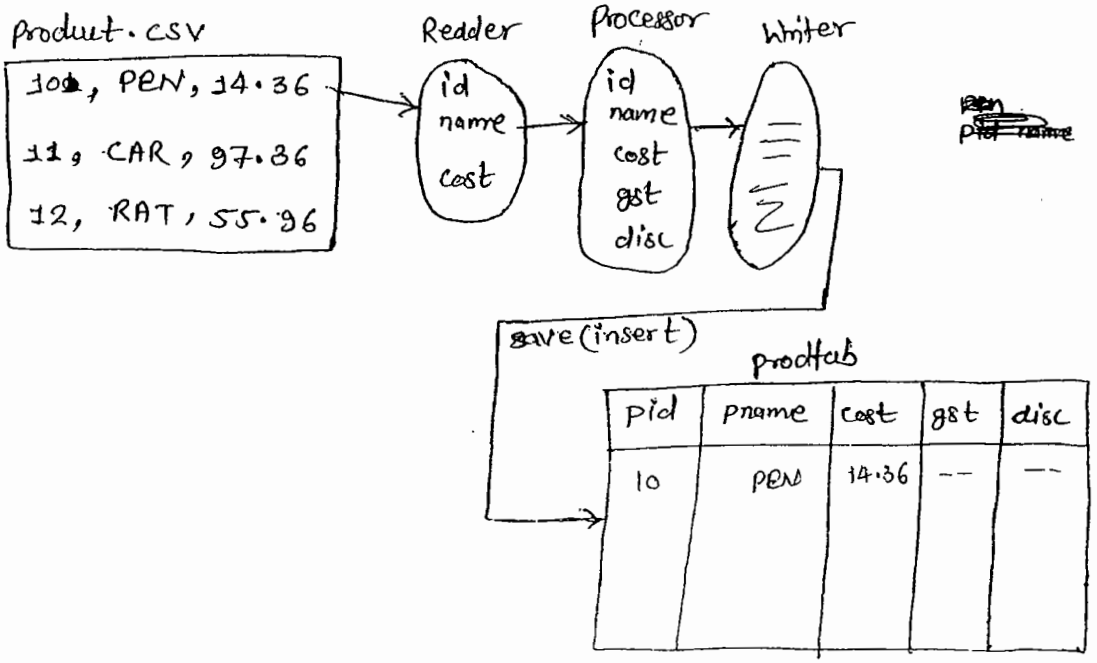
→ Consider input Data given by csv file is Related to Products having ~~product Id~~, name, cost.

→ By using one item ~~Reader~~ ItemReader convert .csv file data to product class object.

→ Define one processor class to calculate gst (Goods and Service Tax) and discount of product.

→ Finally product should have id, name, cost, gst, discount.

→ Use one ItemWriter class to convert one object to one ROW in DB Table



** Difference Ways of creating Object and calling method :-

--- Consider below class :---

```
class Sample {
    Sample() {
        sopln("CONST");
    }
    void show() {
        sopln("METHOD");
    }
}
```

---Test class---

```
Public class Test {
public static void main(String [] args) {
//1. Creating object and calling method
    Sample s = new Sample();
    s.show();
//2. creating object and calling method
     new Sample().show();
//3. Creating object (add extra code, override methods) and calling
   method
    new Sample() {
        public void show() {
            sopln("NEW LOGIC");
        }
    }.show();
//4. While creating object invoke method
    new Sample() {
        {
            show();
        }
    };
```

→ instance initializer block

*) FlatFileItemReader<T> :-

This class is provided by Spring Batch to read data from any Text Related file (.txt, .csv, ...).

*) DefaultLineMapper :-

It will load one row(\n) at a time as one Line object.

*) DelimitedLineTokenizer :-

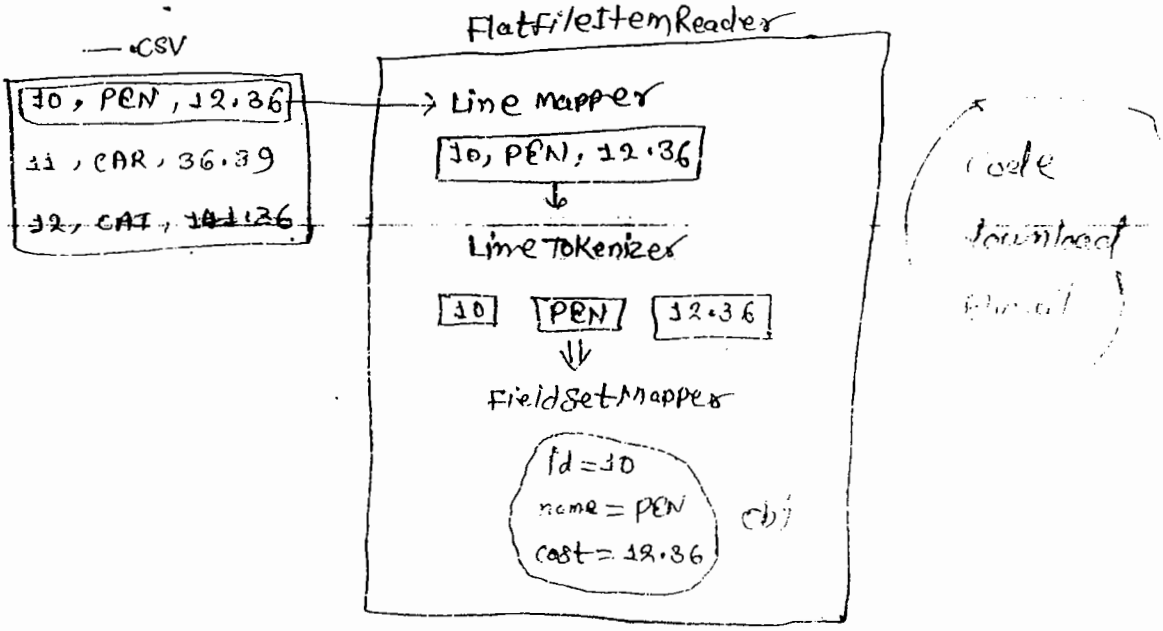
It will devided one Line values into multiple values using any Delimiter (like comma, space, tab, new line, dash-etc...)

⇒ Default Delimiter is "," [Comma]

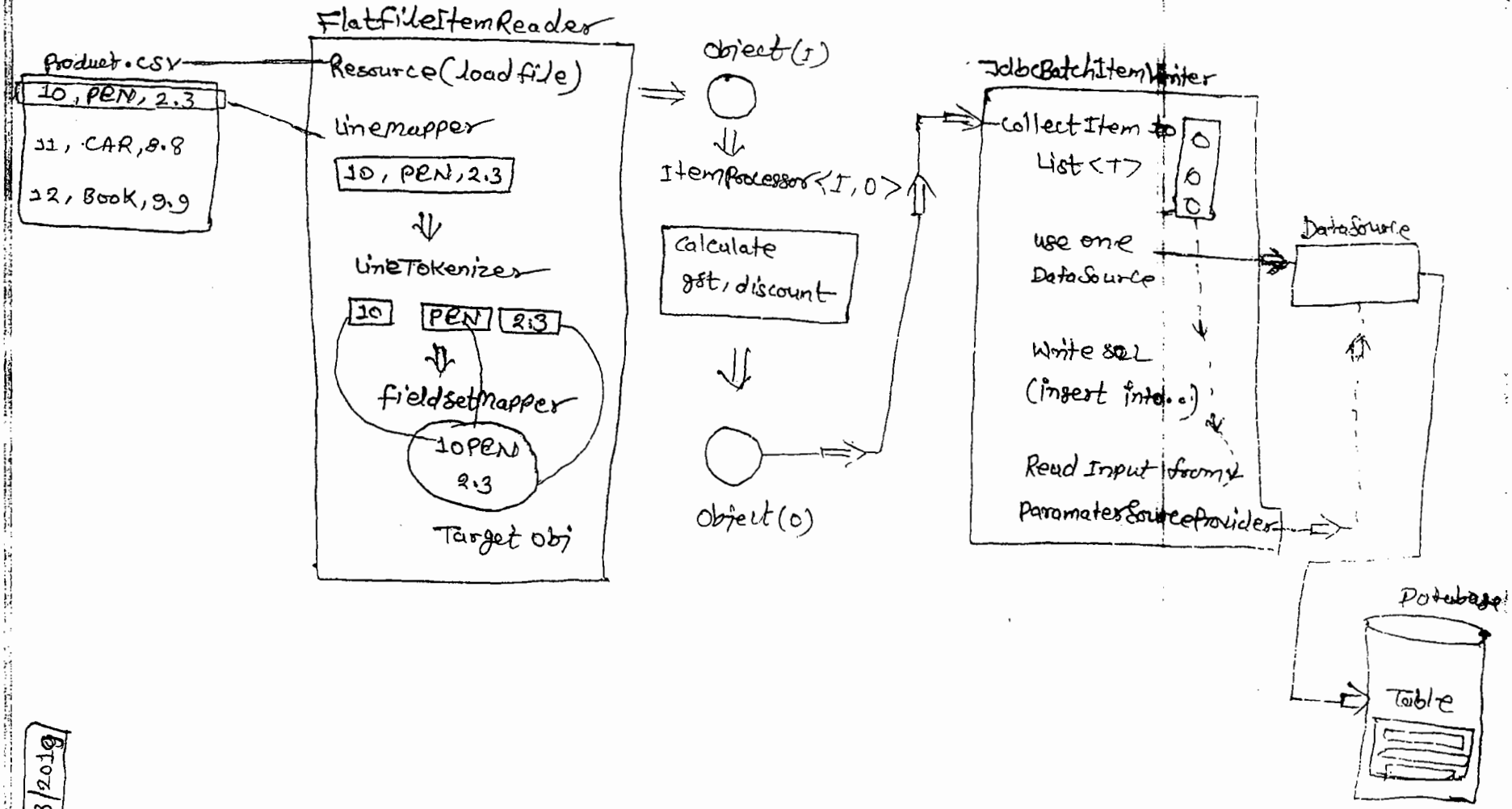
*) ~~BeanWrapperFile~~

*) BeanWrapperFieldSetMapper :-

~~It means~~ ~~will~~ It maps data to variables, finally Converted to one class type (TargetType)



Spring Boot Batch CSV to MySQL DB ⇒



05/03/2019

Execution Flow :-

Spring Boot Batch flw has provided Pre-defined ItemReader and ItemWriters.

→ FlatFileItemReader<T> is used to load any file (source) as input to read data. Eg:- .txt, .csv, ... etc.

→ It will read one Line data based on LineMapper (\n)

→ One Line data is provided into multiple values based on Tokenizer (Delimeter = ,)

→ These values are mapped to one class (T) type object also known as Target.

→ This Target object is input to ItemProcessor. After processing object (calculations, logical checks, data modifications ... etc) Processor returns output type object.

→ JdbcBatchItemWriter collects all items from ItemProcessor in List<T> based on chunk (int) size.

→ Provider one DataSource (DB connection) to communicate with DB Tables.

→ Define one SQL which inserts data into table based on parameter (Variable Name) SourceProvider.

→ multiple SQL are converted to one batch and send to DB Table.

* Here Batch size = chunk Size.

Example:- chunk size = 150 (int value)

→ chunk indicates maximum Item to be sent to writer in one network call from step.

→ For last network call chunk may contain few Items (no. of Items < chunk size)

→ In application.properties add below key-value pairs:

spring.batch.job.enabled = false

spring.batch.initialize-schema = always

→ Here job.enabled = false will disable execution of job one time on app starts by starter class.

→ initialize-schema = always will allow spring Batch to communicate DB to hold its Repository details (like Job, step, current status details...)

→ *** In case of Embedded Database initialize-schema not required.

** Task:

#1 Write Spring boot Batch Application to read data from Database (mySQL DB) using "JdbcCursorItemReader" and write data to csv file using "FlatFileItemWriter".

#2. Read data from MongoDB using "MongoItemReader" and write data to JSON file using "JsonFileItemWriter"

** code Snippet for csv TO MySQL DB : ——

```
@Bean
public ItemReader<Product> reader() {

  FlatFileItemReader <Product> reader = new
                              FlatFileItemReader <Product> ();

  reader.setResource(new ClassPathResource("myProds.csv"));

  reader.setLineMapper(new DefaultLineMapper<Product>() {{

   {
   setLineTokenizer(new DelimetedLineTokenizer() {{

    setNames ("prodId", "prodName", "prodCost");

  }
  }});
  setFieldSetMapper(new BeanWrapperFieldSetMapper<Product>()
  {{
     setTargetType(product.class);
  }});
  }});
     return reader;
   }
```

```
@Bean
public ItemWriter<Product> writer() {
 JdbcBatchItemWriter<Product> writer = new JdbcBatchItemWriter<>();
 writer.setDataSource(dataSource);
 writer.setItemSqlParameterSourceProvider(new BeanPropertyItemSqlParameterSourceProvider<Product>());
 writer.setSql("INSERT INTO PRODTAB(PID, PNAME, PCOST,
                                    PGST, PDISC)
                VALUES(:ProdId, :ProdName,
                       :ProdCost, :ProdGst, :ProdDisc)");
 return writer;
}
```

--- DB Table ---

```
CREATE TABLE prodstab ( PID int(11), PNAME varchar(50),
                        PCOST double; PGST double, PDISC double)
```

--- Product.csv : under src/main/resources ---

```
10, PPN, 14.36
11, CAR, 8.8
12, BUS, 99.84
```

**) To read file from outside Application from File system (D:/driver or C:/driver) then use FileSystemResource.

It same way to read file from network (internet ~~URL~~ URL based) then use UrlResource in place of ClassPathResource.

26/03/2019

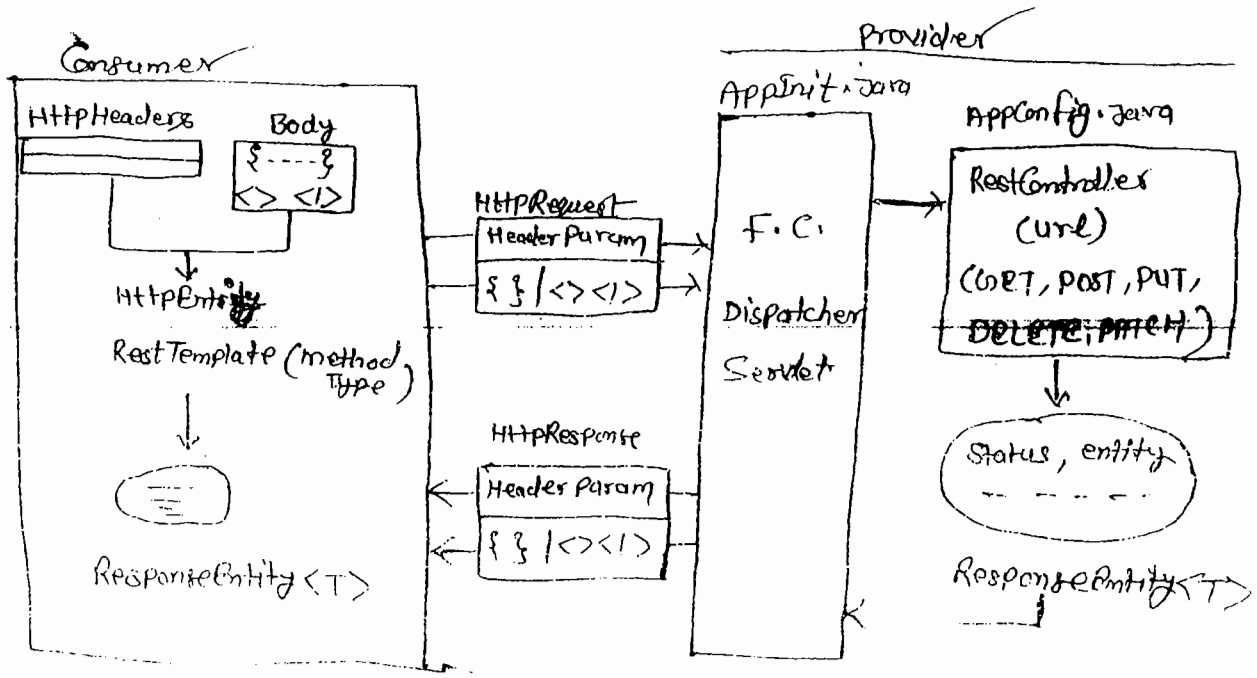
Chapter #6

Spring Boot Rest (Provider & Consumer)

To implement RestFul Webservices API using simple annotations and Templates spring Boot Rest F/w has been introduced.

Even to implement microservices design spring boot Rest API is used.

- - Design of Consumer and Provider - - - - - -



NOTE:-

(a) Consumer and provider inter change the data primitive (String format only), ClassType (object) and Collections.

(b) Data ~~file~~ will be converted to either JSON or xml format also known as ~~Global~~ Global formats.

(c) String data will never be converted to any other type directly.

(d) Rest Controller supports 5 Types of Request methods Handling. Those are Http method: GET, POST, PUT, DELETE,

(e) Controller class level use annotations @RestController (StereoType), @RequestMapping (for URL).

(f) Controller methods level use annotations

Type	Annotation -
GET	@GETMapping
POST	@PostMapping
PUT	@PutMapping
~~DELETE~~	@DeleteMapping
PATCH	@PatchMapping

(g) To provide Inputs to Request methods in RestController, use annotations

Type	Annotation:
url ? Key = val	@RequestParam
/url/val	~~@Path~~ @PathVariable
/url/key=val	@MatrixVariable
HttpHeader	@RequestHeader
Http Body	@RequestBody

** All above annotations work on HttpRequest only, supports reading input data.

(h) By default Matrix parameter is disabled (may not work properly). To enable this write below code in MVC Config File.

--- Example ---

```
@Configuration
public class AppConfig Implements WebMvcConfigurer {

@Override
public void configurePathMatch(PathMatchConfigurer configurer) {
UrlPathHelper helper = new UrlPathHelper();
helper.setRemoveSemicolonContent(false);
configurer.setUrlPathHelper(helper);
}}
```

(i) DispatcherServlet is ~~an~~ autoconfigured in spring boot with URL pattern mapped to /.

(j) AutoConfiguration provided even for ~~@Enable~~ @EnableWebMvc, ~~Web~~ @ComponentScan (starts pack) @PropertySource ("classpath: application.properties")

(k) ~~End~~ Embedded servers : Tomcat, default port : 8080

#1 Spring Boot RestT Provider Ex#1

S#1 Create Starter ~~class~~ Project using web and devtools Dependencies

Details :

GroupId : com.app

ArtifactId : SpringBootRestProvider

Version : 1.0

S#2 Writer one Controller class with class level URL and different method with HTTP Method Types

~~package com.app.controller.rest;~~

//ctrl+shift+o

@RestController

@RequestMapping("/admin") // optional

Public class AdminRestController {

```
@GetMapping("/show")
public String helloMsgGet() {
    return "Hello from GET";
}

@PostMapping("/show")
public String helloMsgPOST() {
    return "Hello from POST";
}

@PutMapping("/show")
public String helloMsgPut() {
    return "Hello from PUT";
}

@DeleteMapping("/show")
public String helloMsgDelete() {
    return "Hello from Delete";
}

@PatchMapping("/show")
public String helloMsgPatch() {
    return "Hello from PATCH";
}
}
```

--- application.properties ---

server.port = 8866

server.servlet.context-path = /myapp

---

*) Run Starter class and Test Application using POSTMAN.

GET http://localhost:8866/myapp/admin/show SEND

NOTE:

(a) URL is case-sensitive, same URL can be used at multiple method levels but Http Method Type (GET, POST, PUT ...) must be different.

(b) Difference between PUT and PATCH:

PUT: It indicates modify complete (full) data, based on identity (ID-PK)

PATCH: It indicates modify partial (few) data, based on identity (ID-PK)

Read Data from Http Request (Head and Body)

=> Http Request sends data using Header and Body Section (both).

To Read Any Header Parameter use code:

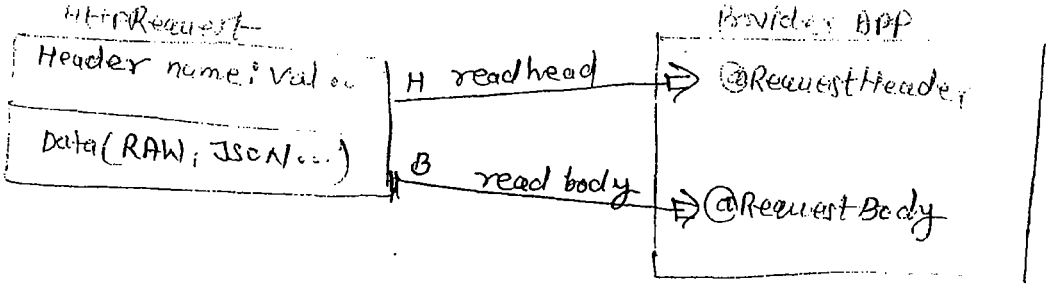
a) @RequestHeader Map<Type, Type> headers

b) @RequestHeader (required = false) Map<Type, Type> headers

@RequestHeader ("key") Type variable

→ To read data from body (Raw Data) use code

@RequestBody DataType localVar



---Code ---

Package com.app.controller.rest;

//ctrl+shift+t.
@RestController
@RequestMapping("/admin")
Public class AdminRestController {

@PostMapping("/head")
Public String readHead(@RequestHeader(required=false) String dnt,
@RequestHeader("Content-Type") String type,
@RequestBody String myData) {

}

---- POSTMAN ----

POST (1) http://localhost:8080/admin/head (2) SEND

head (3)

dept 4-SAMPLE

Content-Type application/json

POST http://localhost:8080/admin/head SEND (8)

5 Body

(*) raw (6) application/json

~~{"message": "This~~

{"message": "This is from Request Body"}

NOTE:

(1) Request Header is required (true) by default to make it optional, add code required = false

(2) If key name localVariable Name Same then providing key name is optional.

(3) Request Body Raw data (characters or any...) can be Stored in String with any variableName.

29/03/2019

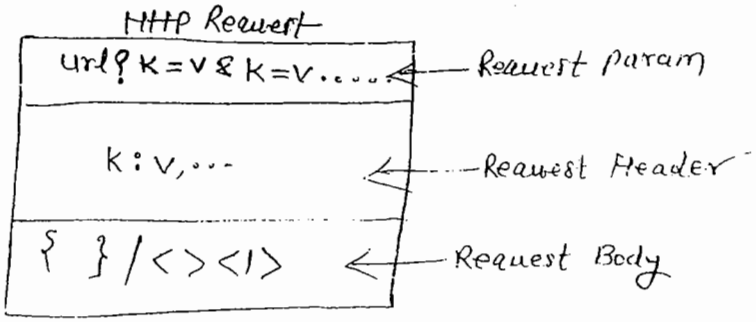
Passing Input to RestController :-

RestController will read input data either in primitive or in Type (obj/ collection).

→ To pass data/input to rest Controller spring has provided different Concepts.

Those are:

1. Request Header.
2. Request Body **
3. Request Parameter
4. Path Variable **
5. Matrix variable (disable mode)



① Request Header :

It provides data in key = value format (both are String Type)

⇒ Request Header also called as Header parameters.

⇒ These are used to provide instructions to Server (application/ Controller).

Ex:- content-type, Accept, host, Date, cookies...

② Request Body :

To send large/bulk data like objects and collections data in the form of JSON/xml

⇒ Even supports large text (RAW data).

⇒ Spring Boot enables JSON conversion by default. But not xml (no JAXB API).

③ Request parameters :

To pass primitive data (like id, name, codes ... etc) as input, request parameters are used.

⇒ format looks like : url?key=value&key=val....

⇒ Both key and values are string type by default.

⇒ Request parameters :

format :

@RequestParam (value = "key" , required = true/false , defaultValue = "-value-")

Datatype localVariable

Syntax #1

@RequestParam("key") Datatype localVar

Ex #1 ~~@RequestParam String data(~~

@RequestParam("sname") String sn

Syntax #2

@RequestParam DataType localVar

Ex#2 @RequestParam String sname

= if key name and local variable Name is same then key is not required to write.

Syntax #3

@RequestParam(required = false) DT localVar

To make key-value is optional

@RequestParam(required = false) String sname

Syntax #4

@RequestParam(required = false, defaultValue = "-DATA-") DT localVar

To change default value from null to any other value.

@RequestParam(required = false, defaultValue = "NODATA") String sname

==========Controller code ==========

```
package com.app.controller.rest;
//ctrl+shift+o
@RestController
Public class StudentController {
 @GetMapping("/show")
 public String showMsg(@RequestParam(value = "sname",
                 required = false, defaultValue="NO DATA") String sid)
 {
```

```
        return "Hello :" +sid;
    }
}
```

--application.properties--

```
server.port = 9988
server.servlet.context-path = /home
```

http://localhost:9988/home/show?sname=YT

01/04/2019

Path variable [Path Parameters] :--

We can send data using url (path). It supports only primitive Data.

-> Path are two types

ⓐ Static path [format : /url]

ⓑ Dynamic path [format : /{key}]

-> Statics path indicates URL, where as Dynamic path indicates Data at runtime.

-> While sending Data using Dynamic path key should not be used. only data.

-> Order must be followed in case of sending multiple parameters.

-> To read data at Controller method, use annotation: @PathVariable

-> Syntax is :

@PathVariable DataType KeyName

---Example --- Controller Code ------

```
Package com.app.controller.rest;
//ctrl+shift+o
@RestController
public class StudentController {
@GetMapping("/show/{sid}/{sname}/{sfee}")
public String showMsg(@PathVariable int sid,
                      @PathVariable String sname,
                      @PathVariable double sfee)
  {
   return "Hello!" +sid+ "," +sname+ "," +sfee;
  }
}
```

Example URL:

http://localhost:9999/show/10/SAM/3.3

Rest Controller - Method ReturnType :-

We can use String (For Primitive Data), A ClassType or Any CollectionType (List, Set, Map..) as method ReturnType.

-> If return Type is String then same data will be sent to client.

-> If return Type is non-String (Class or Collection Type) then Data Converted to Global Format (ex: JSON/xml).

-> Default Conversion Type supported by Boot is JSON (Java Script Object Notation)

→ Even "ResponseEntity<T>" can be used as Return Type which holds Body(T) and status (HttpStatus enum).

→ Possible HTTP status are (5)

Code	Types
1XX	Informational
2XX	Success
3XX	Redirect
4XX	client side Error
5XX	Server side Error.

→ To Convert object to JSON (and JSON to object) SpringBoot uses JACKSON API.

Step #1 Create one Spring boot Starter App with web, devtools dependencies

Step #2 Add below dependency in pom.xml for xml (JAXB) Supports.

```
<dependency>
<groupId> com.fasterxml.jackson.dataformat </groupId>
<artifactId> jackson-dataformat-xml </artifactId>
</dependency>
```

Step#3 model class

```
Package com.app.model;

@XmlRootElement
Public class Employee {

        Private Integer empId;
        Private String  empName;
        Private Double  empSal;

    //def, param const, set/get ...
}
```

Step#4 Controller class

```
Package com.app.Controller.rest;
//ctrl + shift + o
@RestController
Public class EmployeeController {

 @GetMapping("/showA")
  public String showA(){

    return "Hello-String";
}

@GetMapping("/showB")
  public Employee showB(){

    return new Employee(22, "AA", 2.2);
}
```

```
@GetMapping("/showC")
public List<Employee> showC(){
    return Arrays.asList( new Employee(22, "AA", 2.2);
                          new Employee(23, "BA", 8.2);
                          new Employee(24, "CA", 2.9);
    }
}

@GetMapping("/showD")
public Map<String, Employee> showD(){
    Map<String, Employee> map = new HashMap<>();
    map.put("e1", new Employee(22, "AA", 2.2));
    map.put("e2", new Employee(23, "DA", 8.2));
    return map;
}

@GetMapping("/showE")
public ResponseEntity<String> showE() {
    ResponseEntity<String> resp = new ResponseEntity<String>("
                Hello RE", HttpStatus.OK);
    return resp;
}
```

postman Screen

GET http://localhost:8080/showD SEND

Header

Key	value
Accept	application/xml

02/04/2019

Spring Boot Rest + Data JPA + MySQL CRUD operations (Rest API With Swagger) :-

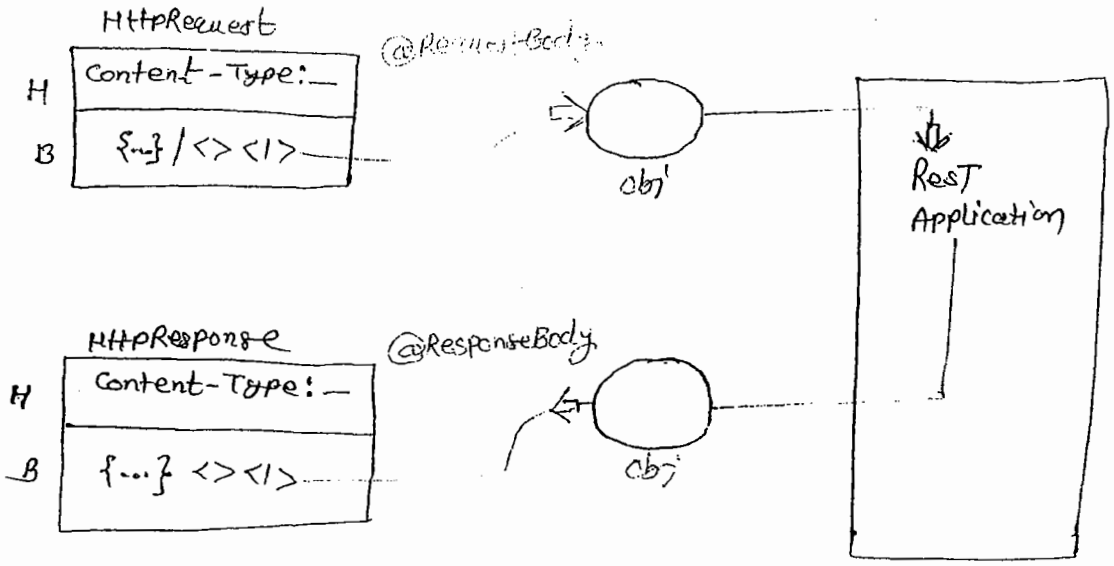
Here, define Rest Controller which works for JSON and xml Data Input/outpu for primitive inputs use PathVariable. Rest Controller must return output type as ResponseEntity<T> which holds Body(T) and HttpStatus. Ex:. 500, 404, 200, 400 etc.

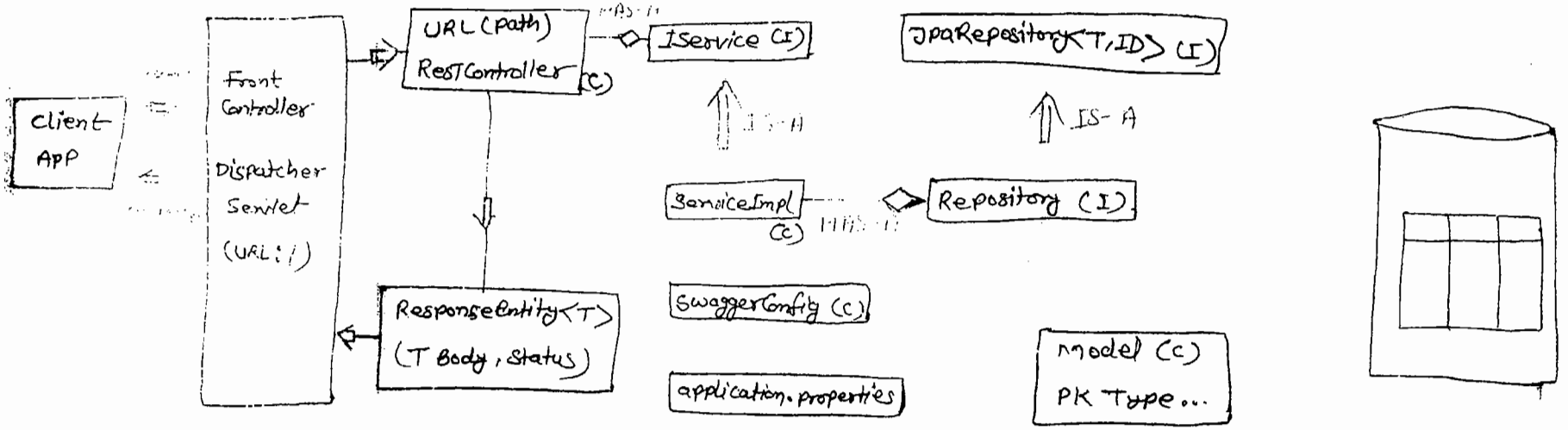
Operation Type	Http Method Annotation
Save	@PostMapping
update	@PutMapping
Delete	@DeleteMapping
get one	@GetMapping
get all	@GetMapping

MediaType Conversion annotations are :
@RequestBody and @ResponseBody.

Here, @RequestBody is applied when we write @RestController annotation over class (to Not handled by Programmer). It Converts [illegible] Type to JSON/xml

@RequestBody should be handled by programmer - It converts JSON/xml Data to object format.




Client App
Front Controller
Dispatcher Servlet
(URL: /)
URL(path)
RestController
(C)
IService (I)
JpaRepository<T,ID> (I)
IS-A
IS-A
ServiceImpl
(C)
Repository (I)
ResponseEntity<T>
(T Body, Status)
SwaggerConfig (C)
application.properties
Model (C)
PK Type...

** Click on show whitspace character symbol (looks like ¶ Symbol : ¶)

Coding Order =>

#1 create project with web, jpa, devtools, mysql dependencies

#2 add ~~jackson~~ jackson-dataformat-xml in pom.xml

#3 Define model class, Repository, IService and Impl in order.

→ Student.java = model

→ StudentRepository.java = Repository

→ IStudentService.java = IService

→ StudentServiceImpl.java = Impl

#4 In application.properties provide keys details server port, datasource and jpa (ddl-auto, show-sql)

#5 Define RestController

---------- StudentRestController ----------

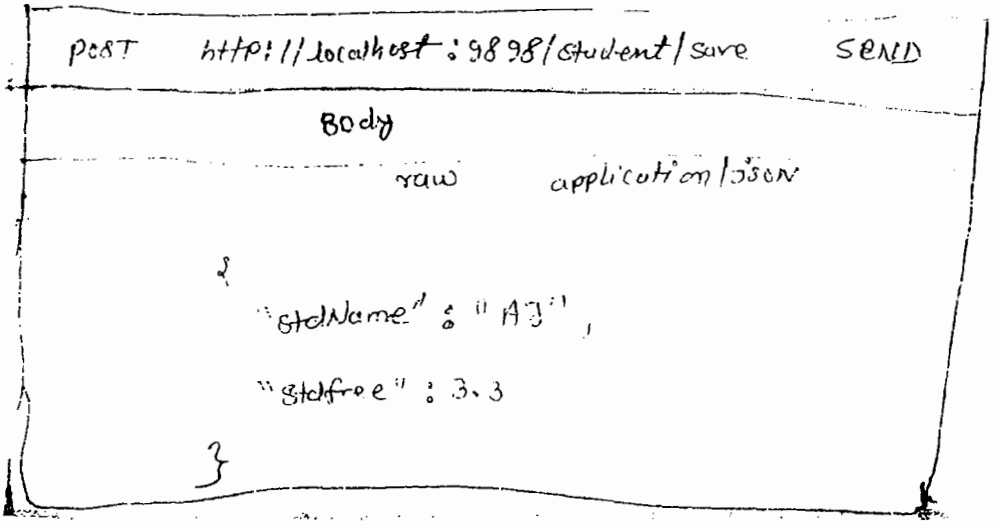
```
package com.app.controller.rest;
//ctrl + shift + o
@RestController
@RequestMapping("/student")
public class StudentRestController {
	@Autowired
	private IStudentService service;
```

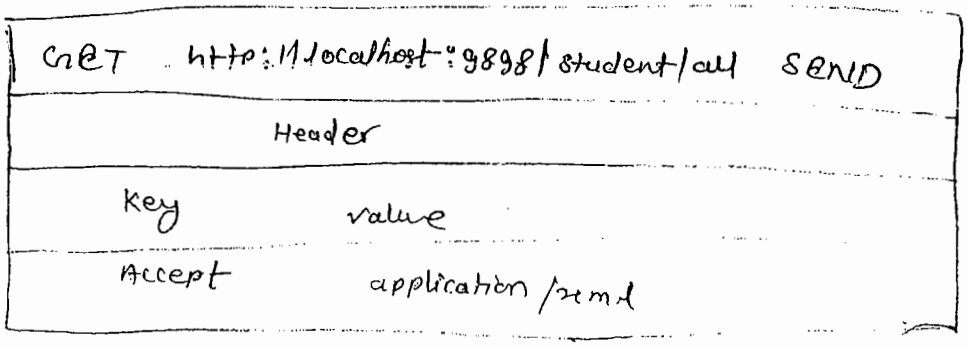
```
//1. Save student data
@PostMapping("/save")
public ResponseEntity<String> save(@RequestBody Student student)
{
ResponseEntity<String> resp = null;
try {
Integer id = service.saveStudent(student);
resp = new ResponseEntity<String>(e.getMessage(), HttpStatus.
                                  INTERNAL_SERVER_ERROR);
    e.printStackTrace();
}
resp = new ResponseEntity<String>("Student" + id + ":HttpStatus.OK);
}
catch(Exception e){

return resp;
}
```

```
//2. get All records
@GetMapping("/all")
public ResponseEntity<?> getAll() {
ResponseEntity<?> resp=null;
List<Student> list=service.getAllStudents();
if(list==null || list.isEmpty()) {
  String message="No Data found";
resp=new ResponseEntity<String>(message,HttpStatus.OK);
}
else{
  resp=new ResponseEntity<List<Student>>(list,Http.
  return resp;
}
}
```


- POSTMAN SCREEN -





03/04/2019

*) Add below two Methods for delete and update based on Id

*) If Request ID is not present then Return Http Status — 400 BAD-REQUEST.

---- Add in StudentRestController :- ----

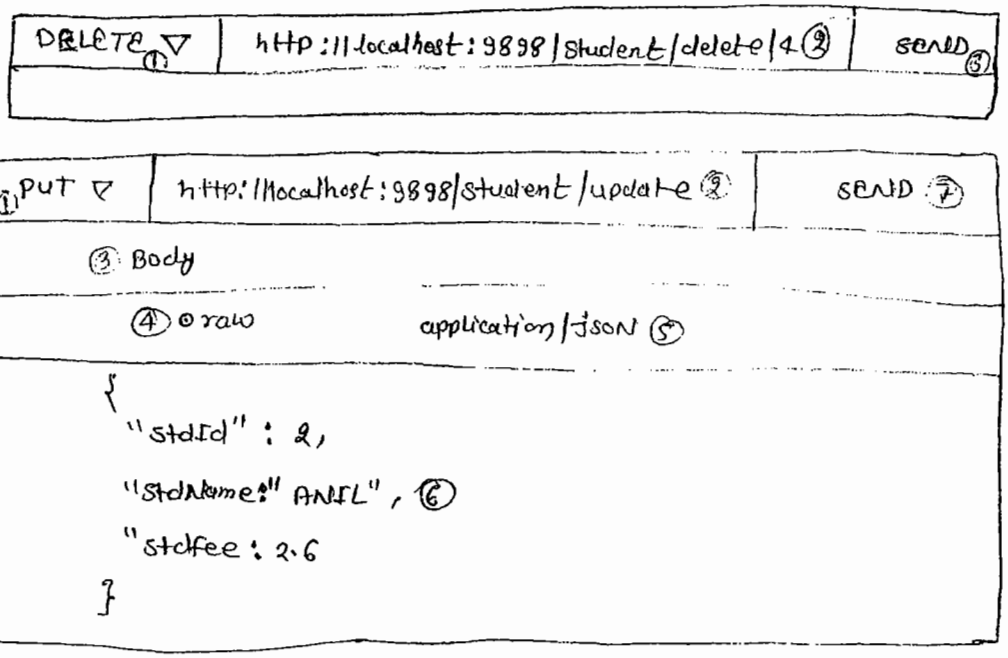
```
// delete based on id if exist
@DeleteMapping("/delete/{id}")
public ResponseEntity<String> deleteById(@PathVariable Integer id)
{
  ResponseEntity<String> resp = null;
// check for exist
  boolean present = service.isPresent(id);
  if(present)
  {
  // if exist
  service.deleteStudent(id);
  resp = new ResponseEntity<String>("Deleted '"+id+"' Successfully",
                                    HttpStatus.OK);
}
```

```
else (if not exist)
{
  resp = new ResponseEntity<String>("'"+id+"' Not exist", HttpStatus.
                                                    BAD_REQUEST);
}
  return resp;
}
```

#4. update data

```
@PutMapping("/update")
public ResponseEntity<String> update(@RequestBody Student student)
{
ResponseEntity<String> resp=null;
// check for id exist
  boolean present=service.isPresent(student.getStdId());
if (present)
{
  // if exist
  service.updateStudent(student);
  resp = new ResponseEntity<String>("updated successfully", HttpStatus.OK);
}
else
  // if not
{
resp=new ResponseEntity<String>("Record '"+student.getStdId()
          +"' not found", HttpStatus.BAD_REQUEST);
}
return resp;
}
```

POSTMAN SCREEN



Enable Swagger UI in Spring Boot Rest application :—

Compared to all other API tools swagger is a Richest API dynamic UI based on code written for Rest Controllers with common paths :

Step #1 Add below Dependencies in pom.xml

```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.7.0</version>
</dependency>

<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.7.0</version>
</dependency>
```

Flow	Meaning
-> Docket()	create Docket
-> select()	choose Rest class
-> apis(basePackage())	classes are in package
-> paths(regex())	having common path
-> build()	create final output

Step#2 Define SwaggerConfiguration class in Application

```
package com.app.config;
// ctrl+shift+o (imports)
@Configuration
@EnableWebMvcSwagger2
public class SwaggerConfig
{
 @Bean
  public Docket MyApi()
 {
  return new Docket(DocumentationType.swagger_2)
  .select()
  .apis(basePackage("com.app.controller.rest"))
  .paths(regex("/rest.*"))
  .build();
 }
}
```

*) basePackage is a static method defined in RequestHandlerSelectors (c) and in same any regex() is a static method defined in PathSelectors (c).

Step#3 Run starter class and enter URL:

http://localhost:8088/swagger-ui.html

04/04/2019

Monolithic Application :-

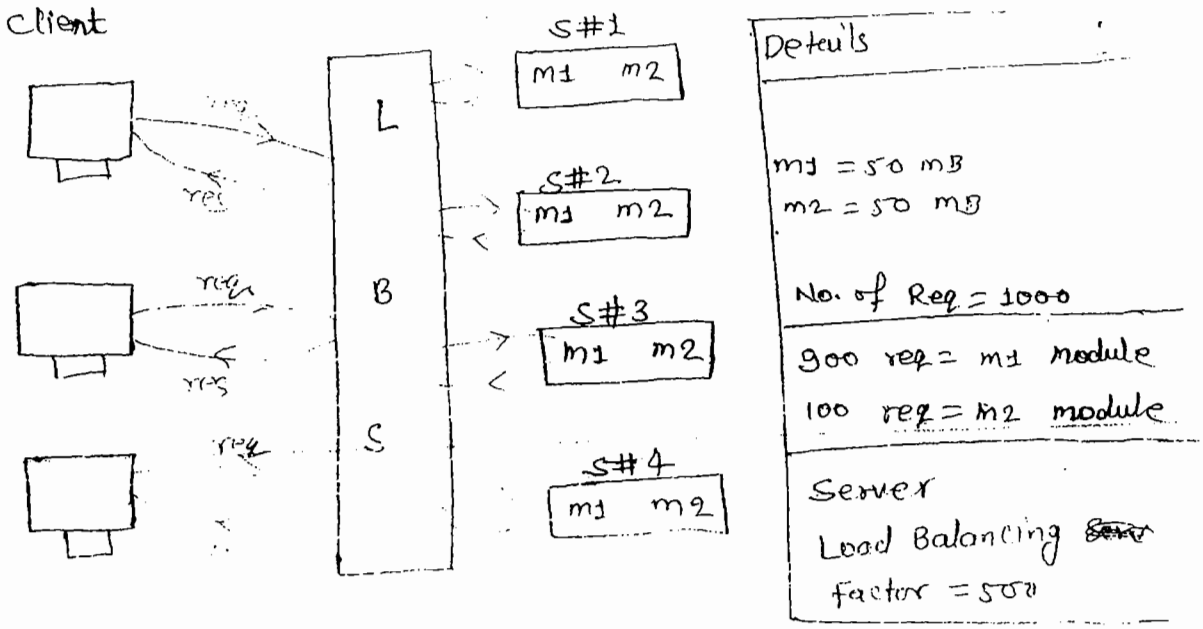
A project which holds all modules together and converted as one service (one .war file).

In this case, if no. of users are getting increased, then to handle multiple request (load) use LBS (Load Balancing Server)

But few modules needs extra load, not all. In this case other modules get memory which is waste (no use). Hence reduces performance of Server (application)

Consider Project P1 is having 2 Modules M1, M2 and their runtime memories are M1 = 50 MB, M2 = 50 MB.

Load Balancing is done using 4 servers. looks like :-



In above example, M2 module is getting less request from client. so, Max 2 instances are may be enough. other 2 instances (memories) are no use. It means near 100 MB memory is not used, which impacts Server performance.

05/04/2019

Microservices

It is a independent deployable component. It is a combination of one (or more) module(s) of a projects runs as one service.

Nature of Microservices :—

(a) Every service must be implemented using Webservices Except.

(b) Each service should be independent.

(c) Services should able to communicate with each other. It is also called as "Intra Communication".

(d) Required Services must be supported for load Balancing. (ie one service runs in multiple instances)

(e) Every service should able to read input data (_.properties/_.yml) from External Configuration Server [Config Server]

(f) Service Communication (chain of execution) problem should be able to solve using Circuite Breakers. [Find other possibles]

(g) All Services must be routed to single Entry Point called as routing Server [Proxy Gateway or API Gateway], it supports Securing, Metering and Routing.

Netflix Component Names :—

Service Registry and Discovery	= Eureka
Load Balancing Server	= ~~Rol~~ Ribbon
Circuite Breaker	= Hystrix
API Gateway	= ~~Zool~~ Zuul
Config ~~Service~~ Server	= Github
Secure Server	= OAuth2
Log and Trace	= Zipkin + Sleuth
Message Queues	= Kafka
Declarative REST clients	= Feign
Integration Service	= Camel
Production ready Endpoints	= Actuator
Metrics UI	= Admin (Server/client)
Cloud plateforms with Deploy services	= PCF, Docker

SOA (Service Oriented Architecture) :—

It is a Design pattern used to create Communication links between multiple service Providers and Consumers (users)

→ Components of SOA :—

a. Service Registry and Discovery [Eureka]

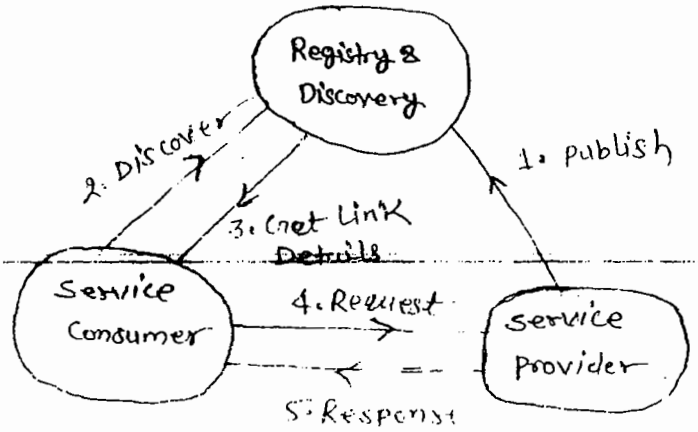
b. Service Provider [Webservice provider]

c. Service Consumer [Webservice client]

→ Operations :—

1. publish
2. Discover
3. Link Details of Provider
4. Query Description [make HTTP Request]
5. Access Service (HTTP Response)

SOA (Service oriented Architecture) :—

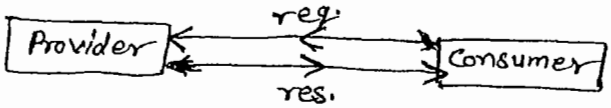


08/04/2019

Implementing MicroService Application using Spring Cloud :-

design #1

A Simple Rest Web-Service using Spring Boot



This Application is implemented using spring boot Restful web services which provides Tightly Coupled design. it means any changes in Provider Application effects Consumer Application. Specially server change, Port Number change, Context path change ---etc.
This design will not support Load Balancing. It is implemented using Rest Controller and Rest Template.

Step #1 create Provider Application

(Dependencies : web only)

GroupId : org.sathyatech

ArtifactId : AdminServiceProvider

Version : 1.0

Step #2 Define one RestController

```
package com.app.Controller;
//ctrl + shift + o
@RestController
@RequestMapping("/Provider")
```

```
public class AdminServiceProvider {
    @GetMapping("/show")
    public String showMsg() {
        return "Hello";
    }
}
```

Step#3 Create Consumer Application

(Dependencies : web only)

GroupId : org.sathyatech

ArtifactId : AdminServiceConsumer

Version : 1.0

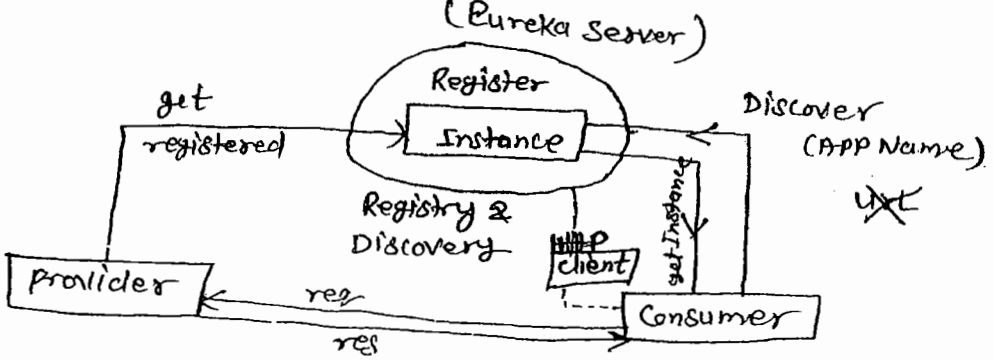
Step#4 Define Consumer (call) code

```
@Component
public class AdminConsumer implements CommandLineRunner {
    public void run(String... args) throws Exception {
        RestTemplate rt = new RestTemplate();
        ResponseEntity<String> resp = rt.getForEntity("http://localhost:
                9999/provider/show", String.class);
        Sop(resp.getBody());
        System.exit(0);
    }
}
```

→ First Run Provider (starter, then Consumer.

≡ MicroService Design and Implementation using Spring Cloud :—
(Netflix Eureka Registry Discovery)

Design #1 [Basic - No Load Balancing]



Step #1 create Eureka Server

Create one spring boot starter project with ~~Dependencies~~

Dependencies : Eureka Server

GroupId : org.sathyatech

ArtifactId : EurekaServerApp

Version : 1.0

Step #2 At starter class level add annotation
@EnableEurekaServer Annotation.

Step#3 In application.properties add keys.

server.port = 8761

eureka.client.register-with-eureka = false

eureka.client.fetch-registry = false

Step#4 Run Starter class and Enter URL as http://localhost:8761 in browser

---

Provider Application

Step#1 create one ~~star~~ Spring boot starter App with web and Eureka Discovery dependencies

GroupId : org.sathyatech

artifactId : StudentServiceProvider

Version : 1.0

Step#2 add below annotation at starter class level

@EnableEurekaClient (given by netflix) or

@EnableDiscoveryClient (given by Spring cloud) both are optional.

Step#3 In application.properties file

server.port = 9900

eureka.client.serviceUrl.defaultZone = http://localhost:8761/eureka

spring.application.name = STUDENT-PROVIDER

step#4 Define one provider Controller

```
Package com.app.provider;
//Ctrl+Shift+O
@RestController
@RequestMapping("/student")
public class StudentProvider {

    @GetMapping("/show")
    public String showMsg() {
     return "Hello from Provider";
    }
}
```

Execution order:(Run starter classes)

1. Eureka Server
2. Provider Application

→ GoTo Eureka and check for Application
→ Click on URL and add /student/show path

09/04/2019

Consumer Application :-

In general Spring Boot application by using any Http Client code Consumer makes request based on use (static / hard-coded) given is code.

Hard Coding :-

Providing a ~~client~~ direct value to a variable in .java file or fixed value.

Ex: int a = 5;

⇒ It provides always same output for multiple runs.

⇒ By using RestTemplates with URL (hard coded) we can make request But it will not support

a. If provider IP / port No. get changed.

b. Load Balancing Implementation.

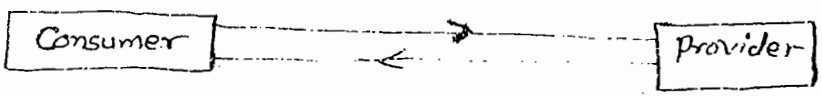
⇒ So, we should use dynamic client that gets url at runtime based on application names registered in "Registry and Discovery Server (Eureka)".

⇒ Discovery client is used to fetch instances based on application Name and we can read URI of Provider at runtime.

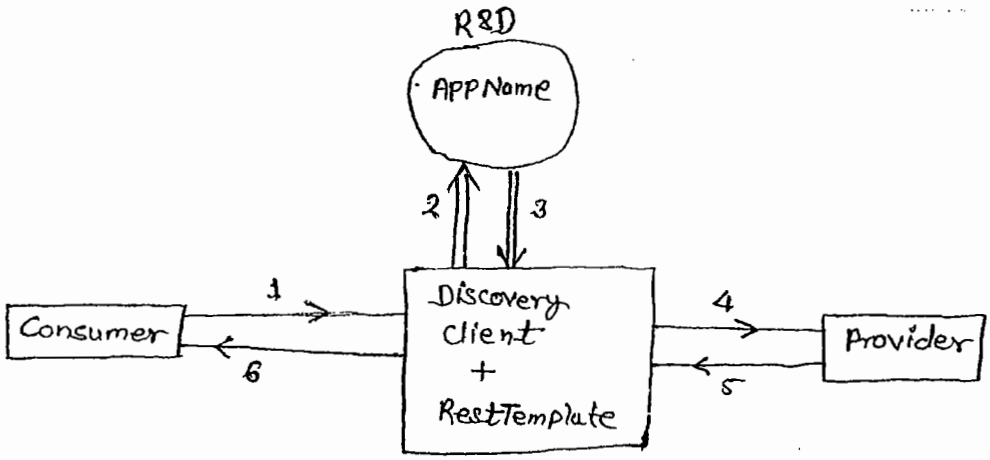
⇒ RestTemplate uses URI (+path = URL) and make request to Provider and gets ResponseEntity which is given back to Consumer.

Webservices Example ⇒

~~RestTemplate~~ (URL)



Microservices Examples :→



→ If one Application is Moved from one server to another server then URI gets changed (Paths remaing Same)

URI
http://192.168.0.1:8989/show
path

Server #1
App #1

URI
http://192.132.0.2:9090/show

Server #2
App #2

Consumer Code :—

#1 Create one Spring Starter Project using Dependencies web, Eureka Discovery

Details:

groupId : org.sathyatech

artifactId : StudentServiceConsumer

Version : 1.0

#2 At Starter class level add Annotation either @EnableEurekaClient or @EnableDiscoveryClient (both are optional)

3 In application.properties file

```
server.port = 9852
spring.application.name = Student-Consumer
Eureka.client.serviceUri.defaultZone = http://localhost:8761/eureka
```

4 Define Consumer code with RestTemplate and DiscoveryClient

```
package com.app.consumer;
//ctrl + shift +o
@RestController
Public class StudentConsumer {
 @Autowired
 Private DiscoveryClient client;

 @GetMapping("/consumer")
 public String consumerData()
 {
 RestTemplate rt = new RestTemplate();

 List<ServiceInstance> list = client.getInstances("STUDENT-PROVIDER");

 ResponseEntity<String> resp = rt.getForEntity(
                                list.get(0).getUri()+"/show", String.class);
```

```
        return "from Consumer => " + resp.getBody();
    }
}
```

---- Execution order ----

#1 Run starter classes in order:

→ Eureka Server, Provider App, Consumer App

#2 Goto Eureka and Client Consumer URL.

enter /Consume path after port Number,

10/04/2019

Declarative Rest client : [Feign client] :-

Spring Cloud supports any Http client to make communication b/w (microservices) Provider and Consumer.

→ RestTemplate is a legacy style which is used to make Http calls with URL and Extra inputs.

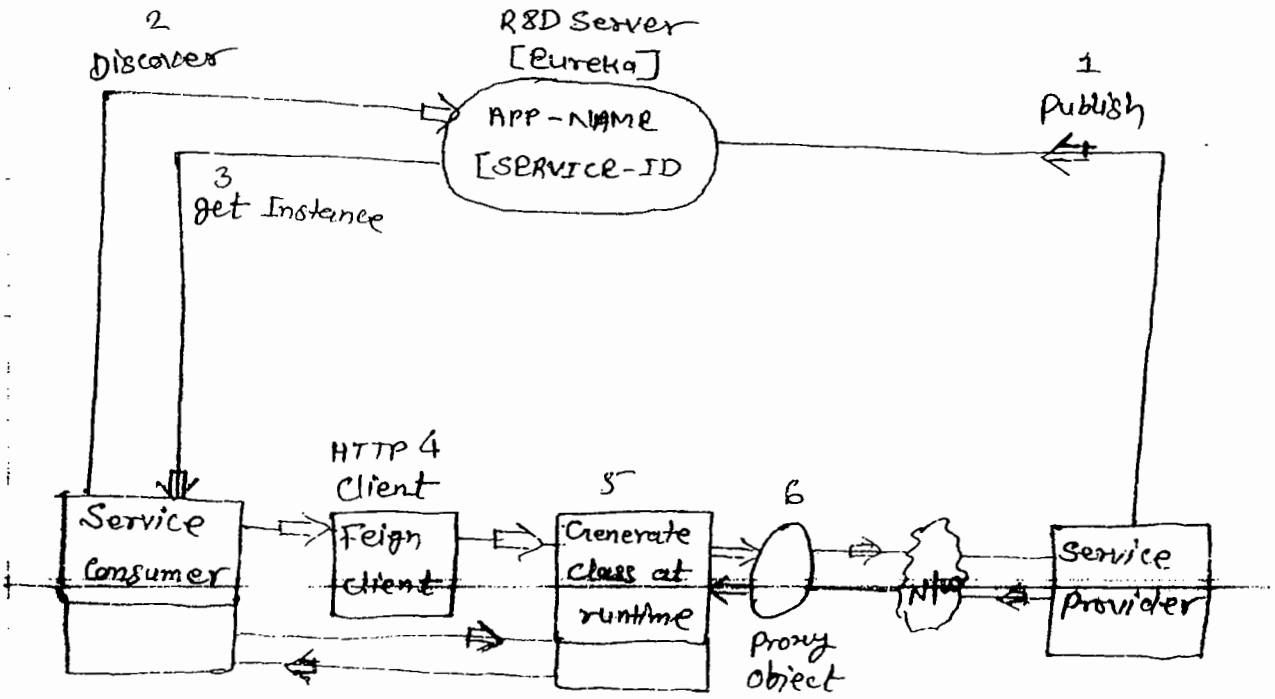
→ RestTemplate with DiscoveryClient makes mask to Provider URL. It means works based on application Name (serviceID). Even URL gets changed it works without any modification at Consumer side.

→ RestTemplate combination always makes programmer to write manual coding for HTTP calls.

→ Spring Cloud has provided one Active client [which behaves as client, but not]. It means, Provide abstraction at code level by Programmer and Implementation is done at runtime by Spring Cloud.

→ Feign is Declarative Client, which supports generating code at runtime and proxy object. By using proxy HTTP Request calls can be made.

→ It supports even parameters (path/Query....) and Global Data Conversion (XML/JSON).



Feign =

→ Feign client is a interface and contains abstraction details, like:

a) path (Provider path at class and method)

b) Http Method Type (GET, POST, ...)

c) Service Id (Application Name)

d) Input Details and Output Type

→ We need to apply Annotation at starter class level:

@EnableFeignClients.

→ At Interface level apply annotation:

@FeignClient (name = "ServiceId")

Syntax:- Feign Client

```
@FeignClient(name="serviceId")
Public interface <ClientName> {

@GetMapping("/path")
// or @RequestMapping("Path")
public <Return> <method> (<params>);

.....
}
```

Example:-

Example:

Provider Code | SID: EMP-PROV

```
@RestController
@RequestMapping("/emp")
public class EmpProvider{

@GetMapping("/show")
public String findMsg(){
.....
}
}
```

Consumer Code Feign Client

```
@FeignClient(name="EMP-PROV")
public interface EmpConsumer{
@GetMapping("/emp/show")
public String getMsg();
}
```

*) Consider last Example Eureka server and Provider Application (STUDENT PROVIDER)

Step#1 create one spring boot starter project for Consumer (using Feign, web, Eureka Discovery)

GroupId : org.sathyatech

ArtifactId : StudentServiceConsumerFeign

Version : 1.0

Step#2 Define one public interface as

```
package com.app.client;
//ctrl+shift+o
```

```
@FeignClient(name="STUDENT-PROVIDER")
public interface StudentFeignClient {

  @GetMapping("/show")
  public String getMsg();
}
```

Step#3 use in any Consumer class (HAS-A) and make method call (HTTP call.)

```
package com.app.consumer;
Ctrl+Shift+O
@RestController
public class StudentConsumer {

 @Autowired
 private StudentFeignClient client;

 @GetMapping("/consumer")
 public String showData() {

  sopln(client.getClass().getName());

   return "CONSUMER=>" + client.getMsg();
 }
}
```

12/04/2019

Load Balancing in Spring Cloud (microservices) :—

To Handle multiple request made by any Http client (or Consumer) in less time. one Provider should run as multiple Instance and Handle request in Parallel. Such concept is called as load Balancing. Load Balancing is to make request handling Faster (reduce waiting time in Queue).

Step to implementing Load Balancing :—

a) Create one Provider Application.

b) Register one Provider as multiple Instances in registry and Discovery (Eureka) Server Every Instance with Unique Id.

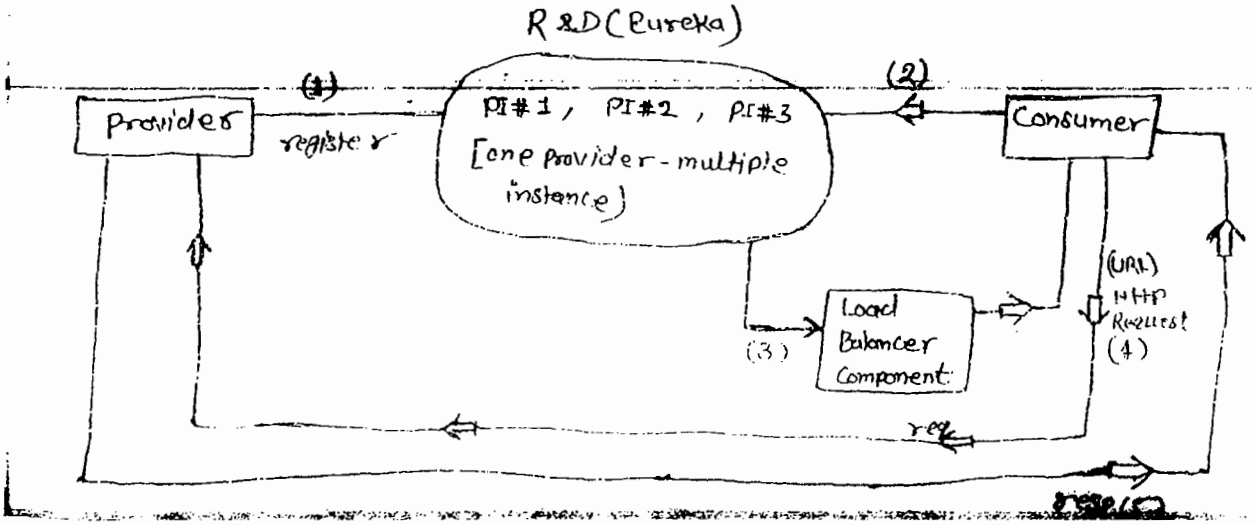
Ex:- P1-58266, P2-2353424, P3-746486 etc....

c) Describe Consumer with any one Load balancer Component (Ex:- Ribbon, Feign)

d) Ribbon choose one Provider URI based on Instance Id with help of LBS Register which maintains request count.

e) Consumer will add path to URI and makes Request using "RequestClient". (Ex: LoadbalancerClient CI) or @FeignClient)

f) ResponseEntity return by Provider to Consumer.



~~Load Balancing~~

Load Balancer Component :- Ribbon, Feign

Request Component :

LoadBalancerClient (I)

⇑ IS-A

RibbonLoadBalancerClient (C)

(Temp memory by Ribbon)

Load Balance Server Registry

Instance Id	Req count
PI # 1	11
PI # 2	10
PI # 3	10

* Ribbon :-

⇒ It is a netflix component provided for spring Boot cloud load Balancing.

⇒ It is also called as client side load balancing. It means Consumer App. read, URI (which one is free) using LBS Register.

⇒ Ribbon should be enabled by every Consumer

⇒ spring cloud uses LoadBalancerClient (I) for "choose and invoice" process.

⇒ It's impl is provided by Ribbon.

---Coding step:---

#1 In provider, define Instance Id for PROVIDER Application using Key "Eureka.instance.instance-id". If not provided default is taken as spring App name.

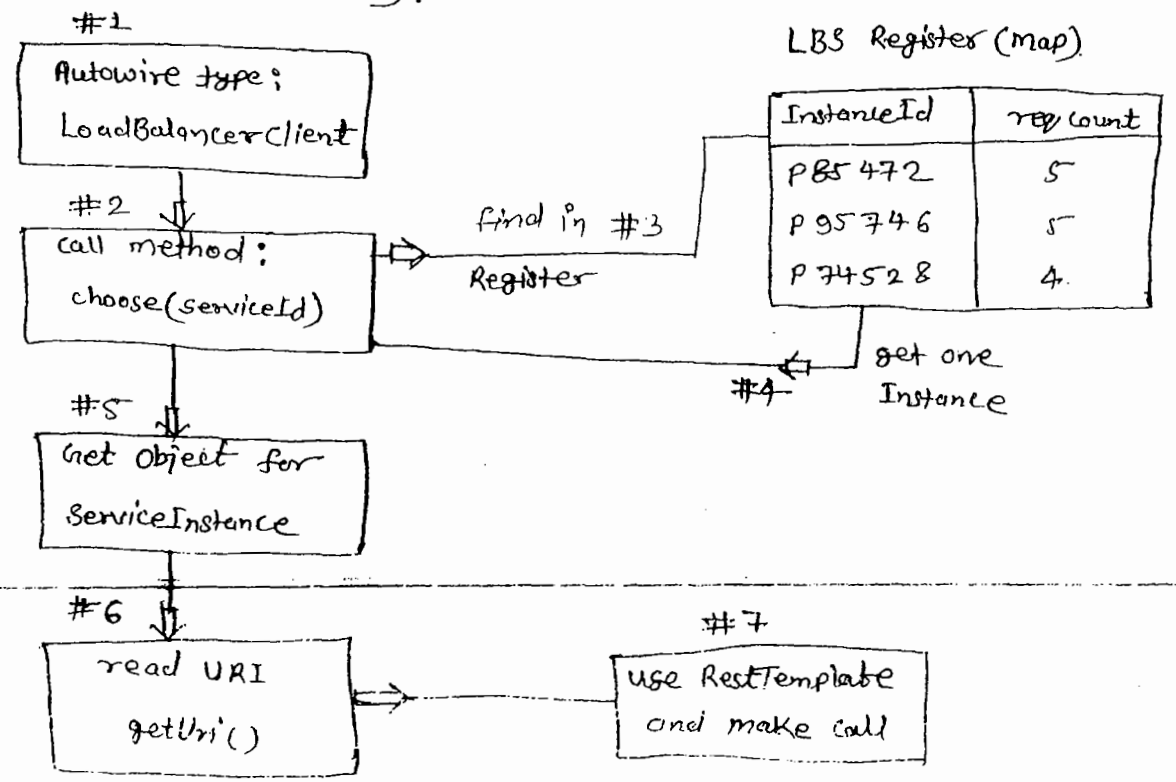
eureka.instance.instance-id = ${spring.application.name}:${random.value} [add in application.properties file]

#2 In Consumer, add dependencies for Ribbon and use LoadBalancerClient (Autowired) and call choose("App-Name") method to get service Instance (for URI)

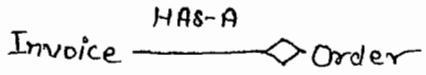
13 | 04 | 2019

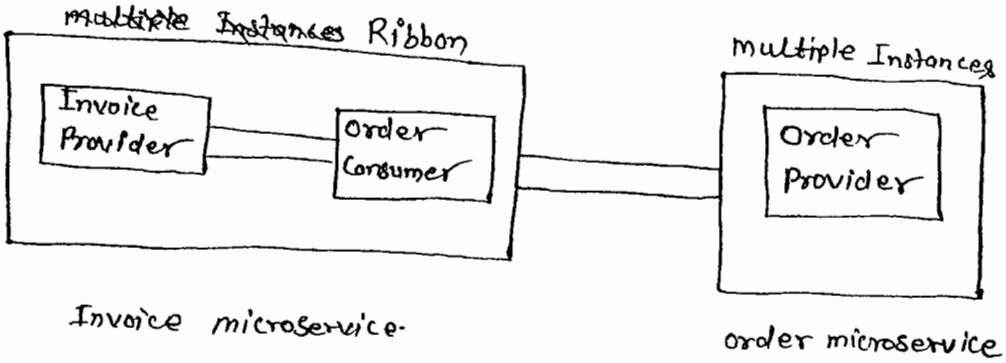
Consumer Execution flow for Request:-

Consumer Flow (LBS):-



*) Consider below Example for Ribbon:





Step#1 Configure Eureka Server

Dependencies: Eureka server only

---- application.properties ----

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

Step#2 Create order Service Provider Application

Dependencies: Eureka, Discovery, Web

---- application.properties ----

```
server.port=9800
spring.application.name=ORDER-PROVIDER
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka
eureka.instance.instance-id=${spring.application.name}:${random.value}
```

** If no instance-id is provided then application name (service Id) behaves as instance id.

---- Controller ----

```
package com.app.controller;
ctrl+shift+o
@RestController
@RequestMapping("/order")
public class OrderProvider {

    @Value("${server.port}")
    private String port;

    @GetMapping("/status")
    public String getOrderStatus() {
        return "FINISHED::" + port;
    }
}
```

Step #3 Define Invoice service provider

Dependencies: Eurek Discovery, Web, Ribbon

---application.properties---

```
server.port = 8800
spring.application.name = INVOICE-PROVIDER
eureka.client.serviceUrl.defaultZone = http://localhost:8761/eureka
```

---Consumer Code---

```
Package com.app.Consumer;
@Service
Public class OrderConsumer {
  @Autowired
  Private LoadBalancerClient client;
  Public String getStatus() {
    String path = "/order/status";
    ServiceInstance Instance = client.choose("ORDER-PROVIDER");
    String uri = Instance.getUri().toString();
    RestTemplate rt = new RestTemplate();
    ResponseEntity<String> resp = rt.getForEntity(uri+path, String.class);
    return "CONISUMER =>" + resp.getBody();
  }
}
```

----- Controller code -----

```
Package com.app.Controller;
// ctrl + shift + o
@RestController
@RequestMapping("/invoice")
Public class InvoiceProvider {
   @Autowired
    Private OrderConsumer consumer;
   @GetMapping("/info")
    Public String getOrderStatus() {
        return consumer.getStatus();
    }
}
```

Execution :-

(a) Start Eureka Server

(b) Run OrderProvider Starter class 3 times

*** change every time port Number like : 9800, 9801, 9802

(c) Run InvoiceProvider Starter class

(d) Go to Eureka and Execute Invoice Provider Instance type Full URL

http://localhost:8800/invoice/info

15/04/2019

Load Balancing using Feign Client :—

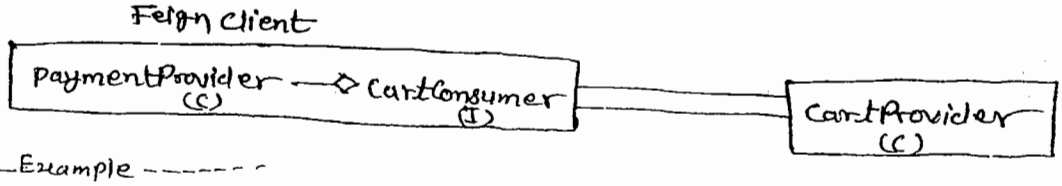
Incase of manual coding for Load Balancing Ribbon Component is used with Type "LoadBalancerClient" (I).

Here, using this programmer has to define logic of consumer method. feign client reduces coding lines by Programmer, by generating logic/code at runtime.

Feign Client uses abstraction process, It means programmer has to provide path with http method Type and also input, output details.

At Runtime RibbonLoadBalancerClient instance is used to choose ServiceInstance and make HTTP call.

Ex# Payment —HAS-A→ Cart



—Example ------

Step#1 create Spring starter Project for Eureka Server with port 8761 and dependency "EurekaServer".

Step#2 Cart Provider Application Dependencies : web, Eureka Discovery

groupId : org.sathyatech
artifactId : CartProviderApp
Version : 1.0

----- application.properties ---

server.port = 8603
spring.application.name = CART-PROVIDER
eureka.client.serviceUrl.defaultZone = http://localhost:8761/eureka

---

*) At Starter class level : @EnableDiscoveryClient

--- Model class ---

```
package com.appmodel;
public class Cart{
  private Integer cartId;
  private String cartCode;
  private Double cartFinalCost;
```

```
//default and param const, set/get, toString
}
```

---------- Cart Service Provider Code ----------

```
package com.app.provider;
@RestController
@RequestMapping("/cart")
public class CartServiceProvider {
 @Value("${server.port}")
 private String port;
 @GetMapping("/info")
 public String getMsg() {
     return "CONSUMER :" + port;
 }
 @GetMapping("/data")
 public Cart getObj() {
     return new Cart(109, "ABC:" + port, 536.36);
 }
 @GetMapping("/list")
 public List<Cart> getBulk() {
   return Arrays.asList(new Cart(101, "A:" + port, 636.36),
       new Cart(102, "B:" + port, 526.46),
       new Cart(103, "C:" + port, 859.38));
 }
}
```

Step #3 payment Provider App with Cart ~~Number~~ Consumer code

Dependencies : web, Eureka Discovery, Feign

~~groupId : org.sathyatech~~

artifactId : PaymentProviderApp

Version : 1.0

---------- application.properties ----------

```
server.port = 9890
spring.application.name = PAYMENT-PROVIDER
eureka.client.serviceUrl.defaultZone = http://localhost:8761/eureka
```

*) At starter class level : @EnableFeignClients

--------*** Cart Service Consumer ***----------

```
package com.app.consumer;
@FeignClient(name = "CART-PROVIDER")
Public interface CartServiceConsumer {
  @GetMapping("/cart/info")
   Public String getMsg();

   @GetMapping("/cart/data")
   public cart getObj();

   @GetMapping("/cart/list")
    public List<cart> getBulk();
  }
```

------Model class----------

Cart model class same as above project

---------PaymentServiceProvider----------

```
package com.app.provider;
@RestController
@RequestMapping("/payment")
public class PaymentServiceProvider {

  @Autowired
  Private CartServiceConsumer consumer;
  @GetMapping("/message")
  public String getMsg()
  {
    return consumer.getMsg();
  }
  @GetMapping("/one")
  public Cart getOneRow()
  {
    return consumer.getObj();
  }
```

```
    @GetMapping("/all")
    public List<Cart> getAllRows()
    {
      return consumer.getBulk();
    }
}
```

Step#4 Execution order

→ Run Eureka Server

→ Run Cart Provider 3 times (with diff. port)

→ Run Payment Provider 1 time

→ Go to Eureka and Run Payment Service

16/04/2019

Spring Cloud Config Server :—

It is also called as ~~Config Server~~ Configuration server. In Spring Boot / Cloud Projects, it contains file's like: Properties file [application.properties]

In some cases Key = Value need to be changed or new Key = vals need to be added At this time, we should

→ Stop Server (Application)

→ Open / find application.properties file

→ Add / Enter key = value pairs or do modifications

→ Save changes [save files]

→ Re-Build file [re-create jar/war]

→ Re-Deploy [re-start server]

And this is done in All Related project [multiple microservices] which is repeated task for multiple applications.

*) To avoid this's repeated (or lengthy) process use application.Properties which is placed outside of your project. i'e known as Config server.

*) Config Server process maintains 3-Properties.

Those are :-

ⓐ one in = under Project (microservice)

ⓑ one in = Config Server (Link file)

ⓒ one in = Config Server (native) or outside Config Server (External) also called as source file

---

Spring cloud Config server can be handled in two ways.

Those are:

ⓐ Native Config Server

ⓑ External Config Server

ⓐ Native Config server :-

It is also called as local file placed in Config Server only.

ⓑ External Config server :-

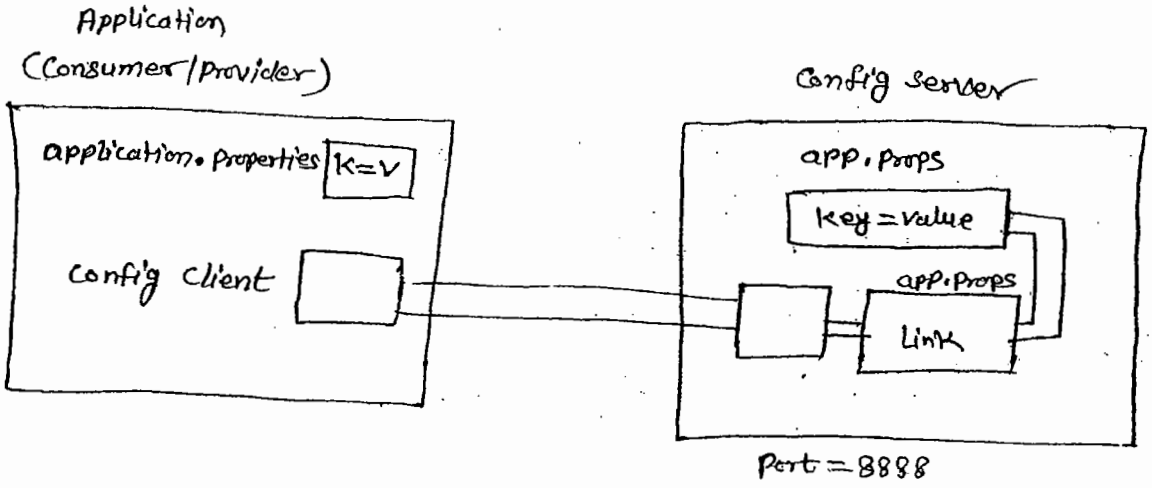
In this case properties file is placed outside the Config Server.

---

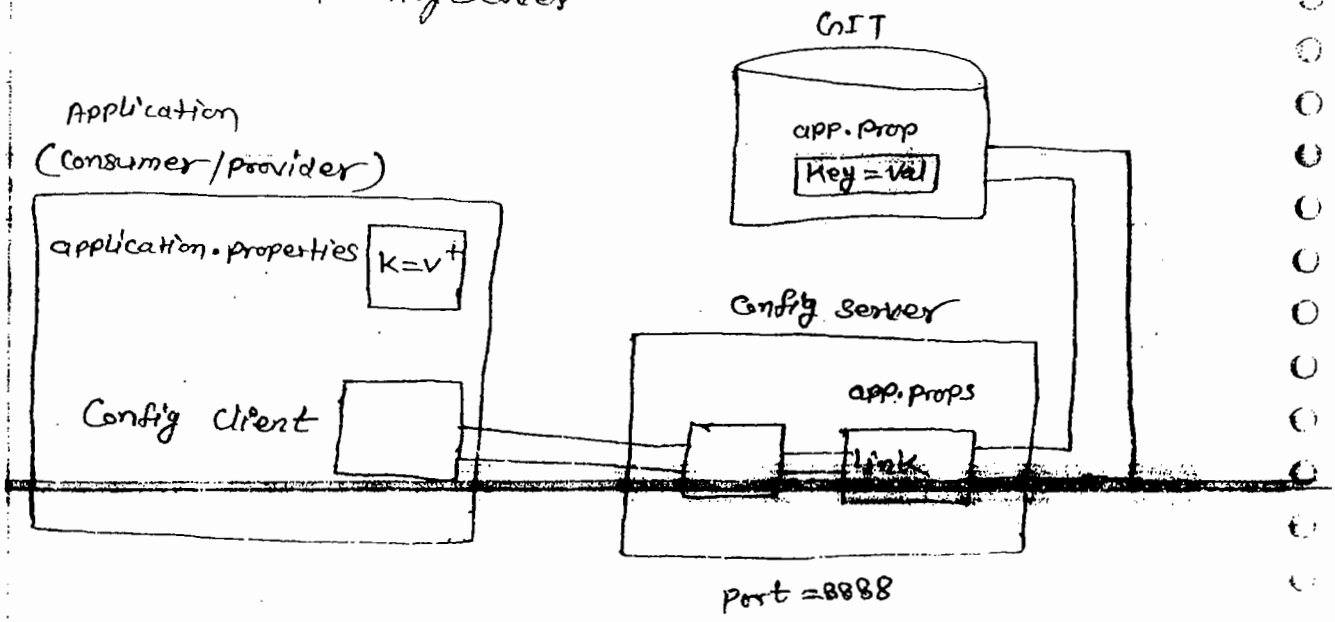
*) In Consumer / Producer project we should add Config Client dependency which gets Config server details at runtime.

*) Config Server runs at default port = 8888

#1 (Native config server)
Local file config server



#2 CNIT Config Server
External Config Server



17/04/2019

Steps to implement Spring Cloud Config Server - Native & External

Step #1 create spring Boot starter project for "ConfigServer" with Dependency: Config Server.

```
<dependency>
    <groupId> org.springframework.cloud </groupId>
    <artifactId> spring-cloud-config-server </artifactId>
</dependency>
```

Step #2 Provide Key values in properties files

Case #a) native

application.properties

```
server.port = 8888
spring.profiles.active = native
spring.cloud.config.server.native.search-locations = classpath:/myapp-config
```

Case #b External

application.properties

```
server.port = 8888
spring.cloud.config.server.git.uri = https://github.com/javabyraghu/
```

Step #3 Create sub folder "myapp-config" in src/main/resources folder, create file application.properties inside this folder. It behaves like source Having esc Key!

eureka.client.serviceUrl.defaultZone = http://localhost:8761/eureka

(or)

Create cld project "configserverex" and create application.properties inside this. place above key and save file.

Step #4 provide include resource code in pom.xml to consider properties files to be loaded into memory.

```
<resources>
  <resource>
    <filtering> true </filtering>
    <directory> src/main/resources </directory>
    <includes>
      <include> *.properties </include>
      <include> myapp-config </include>
    </includes>
  </resource>
</resources>
```

Step #5 At starter class of configServer APP, add Annotation:

@EnableConfigServer

Step #6 In consumer/provider projects pom.xml file ~~add~~ add dependency: config client (or copy below dependency)

```
<dependency>
```

<groupId> org.springframework.cloud </groupId>

<artifactId> spring-cloud-starter-config </artifactId>

</dependency>

---- Execution order ----

> Eureka server > config server

> Producer (Provider) > Consumer

--- Code get Email ---

Step#1 Eureka Server project Dependency: Eureka server

-> Write application.properties

-> Add @EnableEurekaServer annotation

Step#2 Define config server project Dependency: config server

-> write application.properties

-> Add @EnableConfigServer annotation

Step#3 Create Provider Project (Order) dependency: web, eureka, discovery, config client

-> write application.properties

-> Add @EnableDiscoveryClient annotation

-> writer Provider (RestController)

Step#4 Create Consumer Project (Invoice)

Dependency : web, Eureka Discovery, config client, Feign

→ Write application.properties

→ Add @EnableFeignClients annotation

→ Write consumer [Feign client] and Invoce provider (RestController)

Step's to configure External Config server (only modifications)

step#1 in config server project, delete folder myapp-config

step#2 in Config Server Project modify application.properties with Git URI

---

server.port = 8888

spring.cloud.config.server.git.uri = https://github.com/Jeevaraghu/configserverexe

---

step#3 in config server project, in pom.xml delete ~~the~~ line <include>myapp-config</include>

step#4 Create git account and create configserverexe repository. under this create file application.properties (same in ---------- application.properties ---------- (in

eureka.client.serviceUrl.defaultZone = http://localhost:8761/eureka

19/04/2019

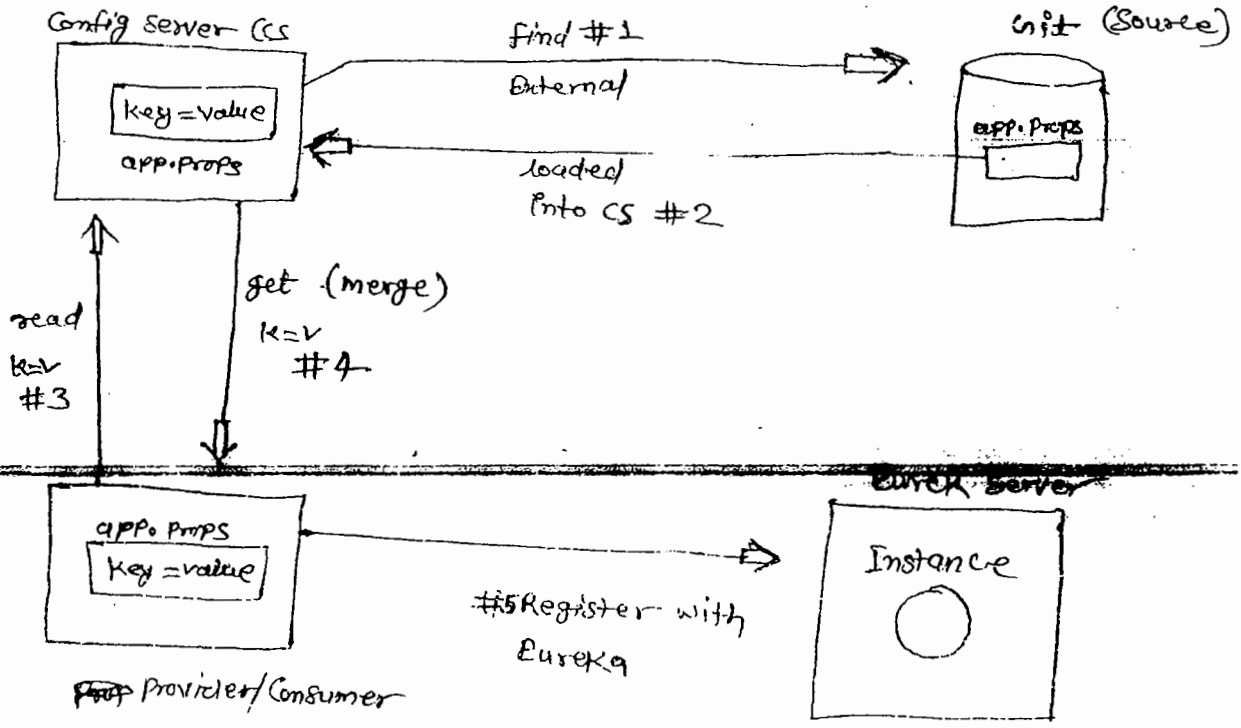
Config Server with Provider/Consumer Execution Flow [External Source] :—

→ On startup ConfigServer (CS) Application it will goto External Source (Git) and read —.properties (or) —.yml file having key=value pairs.

→ Then Provider/Consumer Application (on startup) will try to read K=V from Config Server and marge with our Application properties.

→ If same key is found in both Config server and Provider/Consumer App, then 1st ~~Provider~~ priority is: Config Server

→ After fetching all required Props then Provider gets registered with Eureka Server.



Question:—

*) What is bootstrap.properties in Spring Boot (cloud) Programming?

Answer:—

our Project (child project) contain input keys in application.properties, In same way Parent Project also maintains one properties file named as: bootstrap.properties ~~with~~ which will be loaded before our application.properties.

Execution Order:-

Parent project – loads – bootstrap.properties
our Project – loads – application.properties

→ we can override this file in our project to provide key-values to be loaded by Parent project.

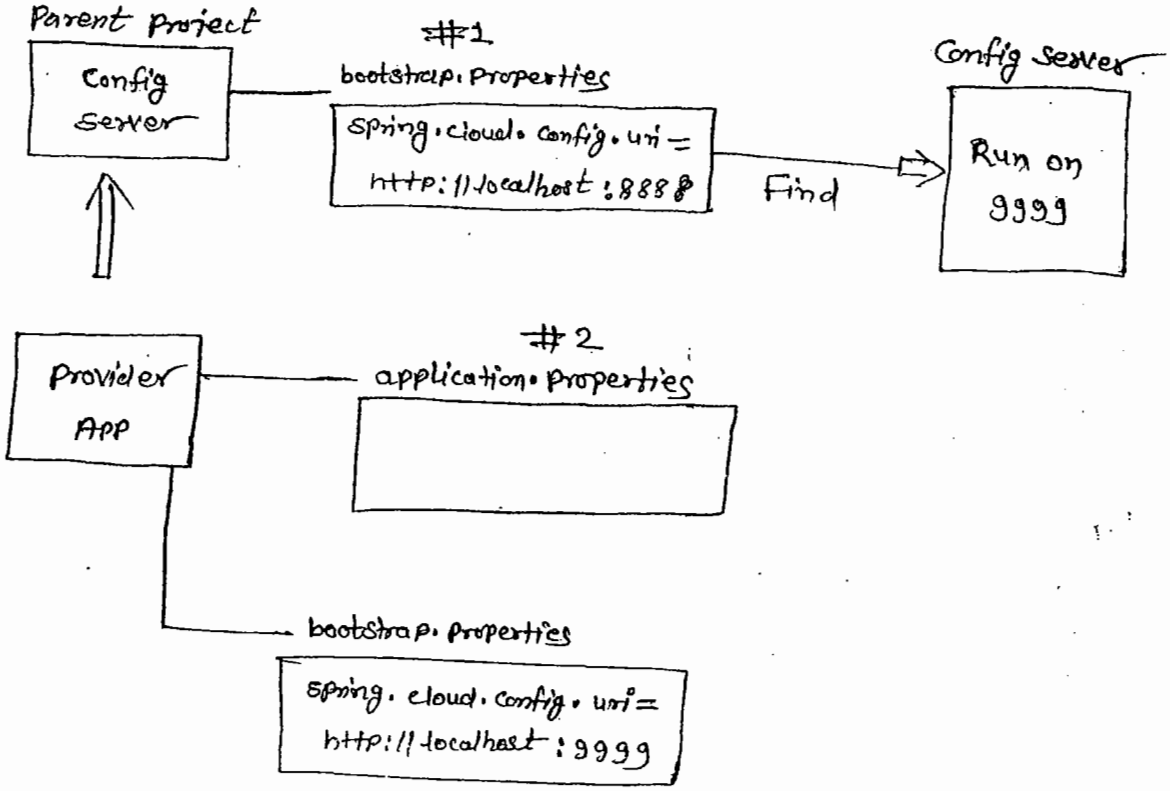
*** by default config server runs on http://localhost:8888 even config client also takes this as default location.

*** To modify this IP/PORT use bootstrap.properties file in our Project.

*** In this bootstrap.properties file ~~order key~~:

override key: spring.cloud.config.uri to any other location where config server is running.

(Config client)



Coding changes for Config Server and Provider/Consumer Apps :—

Step#1 Open application.properties file in config Server and modify port number

server.port=9999

Step#2 In provider/Consumer project create file bootstrap.properties under src/main/resources

Step#3 Add Key=value in bootstrap.properties file

spring.cloud.config.uri=http://localhost:9999

Question :-

Which class will load bootstrap.properties file for config server URI fetching ?

Answer :-

ConfigServicePropertySourceLocator will read config server input by default from http://localhost:8888.

It is only first execution step in provider/consumer project.

20/04/2019

Fault Tolerance API :-

If any microservice is contineously throwing Exception's, then logic must not be executed every time also must be finished with smooth termination. Such process is called as Fault Tolerance.

→ Fault Tolerance is achieved using FallBack method and ~~CircuitBreaker~~ CircuitBreaker.

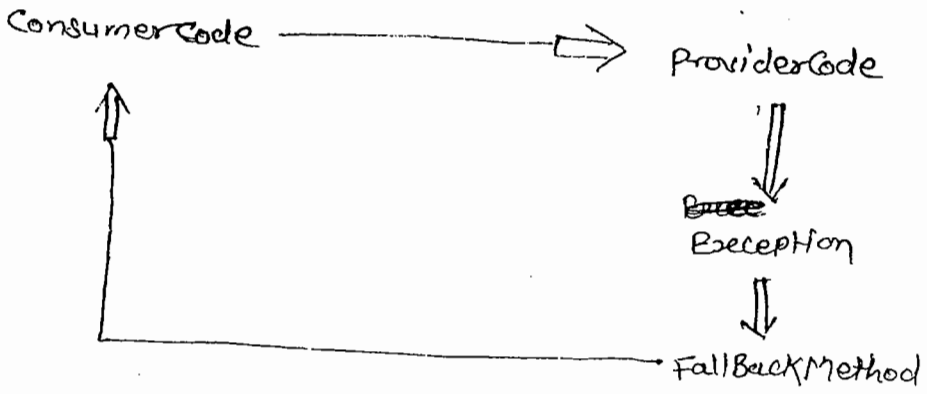
a. FallBack method :-

If microservice is throwing exception then service method execution is redirected to another suppotive method (FallBack method) which gives dummy output and "alerts to DashBoard" (~~to~~ Dev, MS, Admin teams).

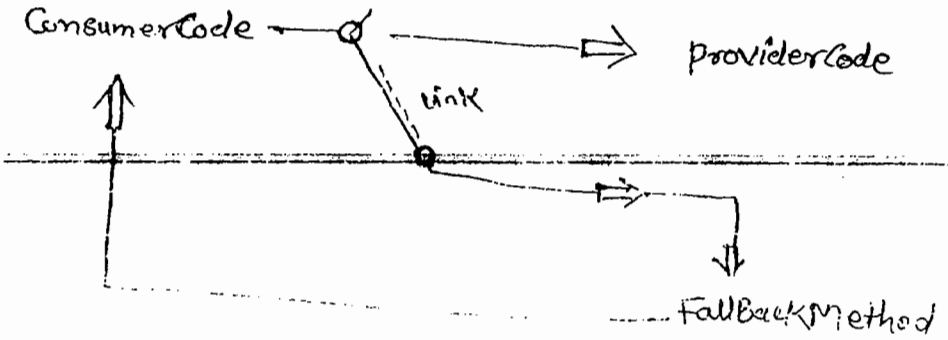
b. CircuitBreaker :—

If Service is throwing exception's continuously the ~~Exception~~ Execution flow directly linked to Fallback method. After some time gap (or) no. of request again ~~Re-hook~~ Recheck once, still same continue to Fallback method else execute microservice.

*) FallBack Process :—



*) CircuitBreaker :—



Hystrix :—

It is a API (set of classes and interfaces) given by Netflix to handle proper execution and Avoid repeated exception logic (Fault Tolerance API) in microservices programming.

→ It is mainly used in Production Environment not in DEV Env.

→ Hystrix supports FallBack and CircuitBreaker process.

→ It provides Dashboard for UI. (View problems and other details in Services)

---working With Hystrix ----------

Step#1 in pom.xml add Netflix Hystrix Dependency

```
<dependency>
    <groupId> org.springframework.cloud </groupId>
    <artifactId> spring-cloud-starter-netflix-hystrix </artifactId>
</dependency>
```

Step#2 At Starter class level apply annotation

@EnableCircuitBreaker (or)

@EnableHystrix

@EnableCircuitBreaker will find concept at runtime using pom.xml dependency ex: Hystrix, Turbin ... etc, where as @EnableHystrix will execute only Hystrix CircuitBreaker

Step#3 Define one service method and apply annotation: @HystrixCommand with details like fallbackmethod, commandkey....

-----Example----------

Step#1 Create spring starter project with Dependencies: web, Eureka discovery, Hystrix

GroupId : org.sathyatech

ArtifactID : OrderProvider

Version : 1.0

> finish

Step#2 Apply below annotations at starter class (both)

@EnableDiscoveryClient

@EnableHystrix

Step#3 application.properties file

server.port=9800

spring.application.name = ORDER_PROVIDER

eureka.client.serviceUrl.defaultZone = http://localhost:8761/eureka

Step#4 Define RestController with FallbackMethod

package com.app.provider;

//ctrl + shift +O

```
@RestController
public class OrderServiceProvider
 {
  @GetMapping("/show")
  @HystrixCommand(fallbackMethod = "showFallBack")
  public String showMsg()
  {
   sopln("from service");
   if (new Random().nextInt(15) <= 10)
   {
    throw new RuntimeException("DUMMY");
   }
   return "Hello from provider";
  }

//--- Fallback method ------
  public String showFallBack()
  {
   sopln("from fallback");
   return "from fallBack method";
  }
}
```

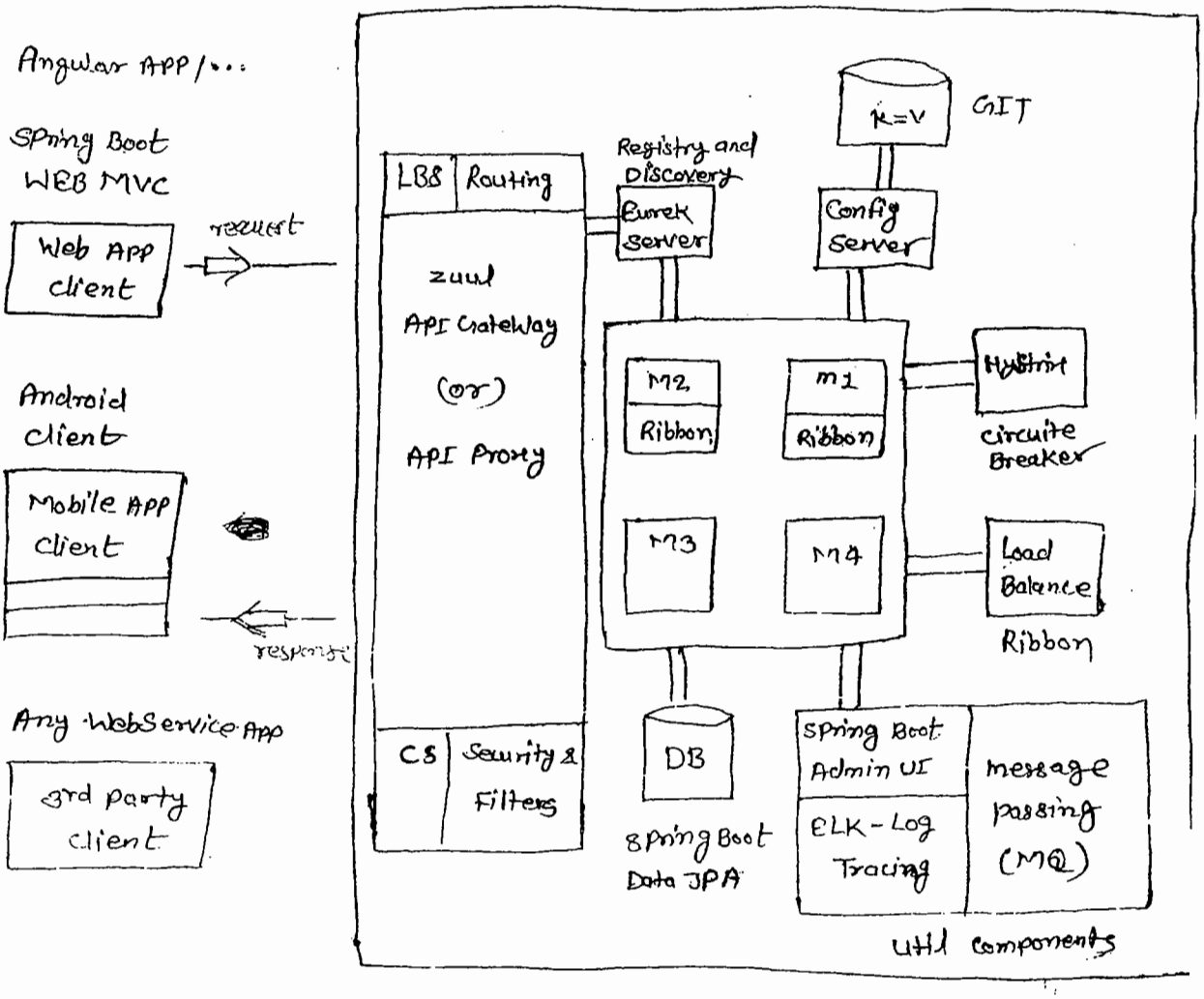
Step #5 Run Eureka server and Order Provider

Step #6 go to Eureka and execute order provider enter with path : /show

*) Note: FallBack method returnType must be same as service method return type.

22/04/2019

Spring Cloud Netflix Microservices Design :-



~~CS Security & F~~

MQ = Message Queues

CS = Config Server

LBS = Load Balancing Server

ELK = Elastic Search -- Logstash - kibana

API PROXY / API GATEWAY :—

In one Application There will be multiple microservices running in different Servers and ports.

— Accessing these by client will be completed.

So, all are accessed through one entry point which makes

→ Single Entry and Exit

→ One time Authentication (SSO = Single sign on)

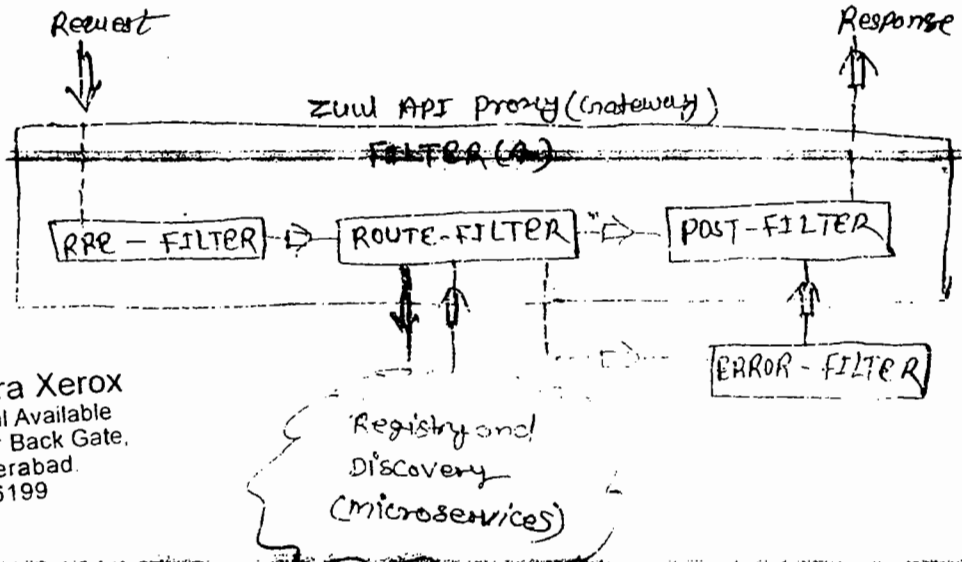
→ Executing of Routing (Find Execution path b/co multiple microservices)

→ Avoid Direct URL to client (Avoid CROS origin Request/response Format).

→ Supports Filtering.

*) Spring cloud netflix zuul behaves as API PROXY for microservices Application which supports "Integration with any type of client Component-" (web, mobile, 3rd party webservices--etc)

Zuul (API PROXY) working Flow :—



23/04/2019

* Zuul API (Proxy-Server) GATEWay :-

Zuul server is a Netflix component used to configure "Routing for microservices" using a key like:

zuul.routes.<module>.path=

zuul.router.<module>.service_id=

**) Before creating zuul server. we should have already created.

(a) Eureka Server Project.

(b) microservices (Ex: Product-Service,
Customer-services, Student-services ....)
(with load balance implemented)

Step #1 Create Spring Starter project as:

zuul-server.

with dependencies:

Web, zuul, eureka-discovery;

Step #2 application.properties should have below details like:

server.port = 8558

eureka.client.service-url.default-zone = http://localhost:8761/eureka
(or) eureka.client.serviceUrl.defaultZone = " " " "
spring.application.name = zuul-Proxy

zuul.routes.<module>.path = /<module>-api/**

zuul.routes.<module>.service-id = (service-Id)

Step #3 In zuul server project at starter class level add:

Annotation: @EnableZuulProxy

Step #4 Implement Filters like : PRE, Route, POST, Error using one abstract class.

ZuulFilter (AC) and use FilterConstants (C)

* Zuul Example Services :—

→ Here Zuul Server behaves as entry and exist point.

→ It hides all details like Eureka server and services instance from client.

→ Zuul provides only Zuul-URL and paths of services only.

→ Zuul takes care of client (Request) load balancing even

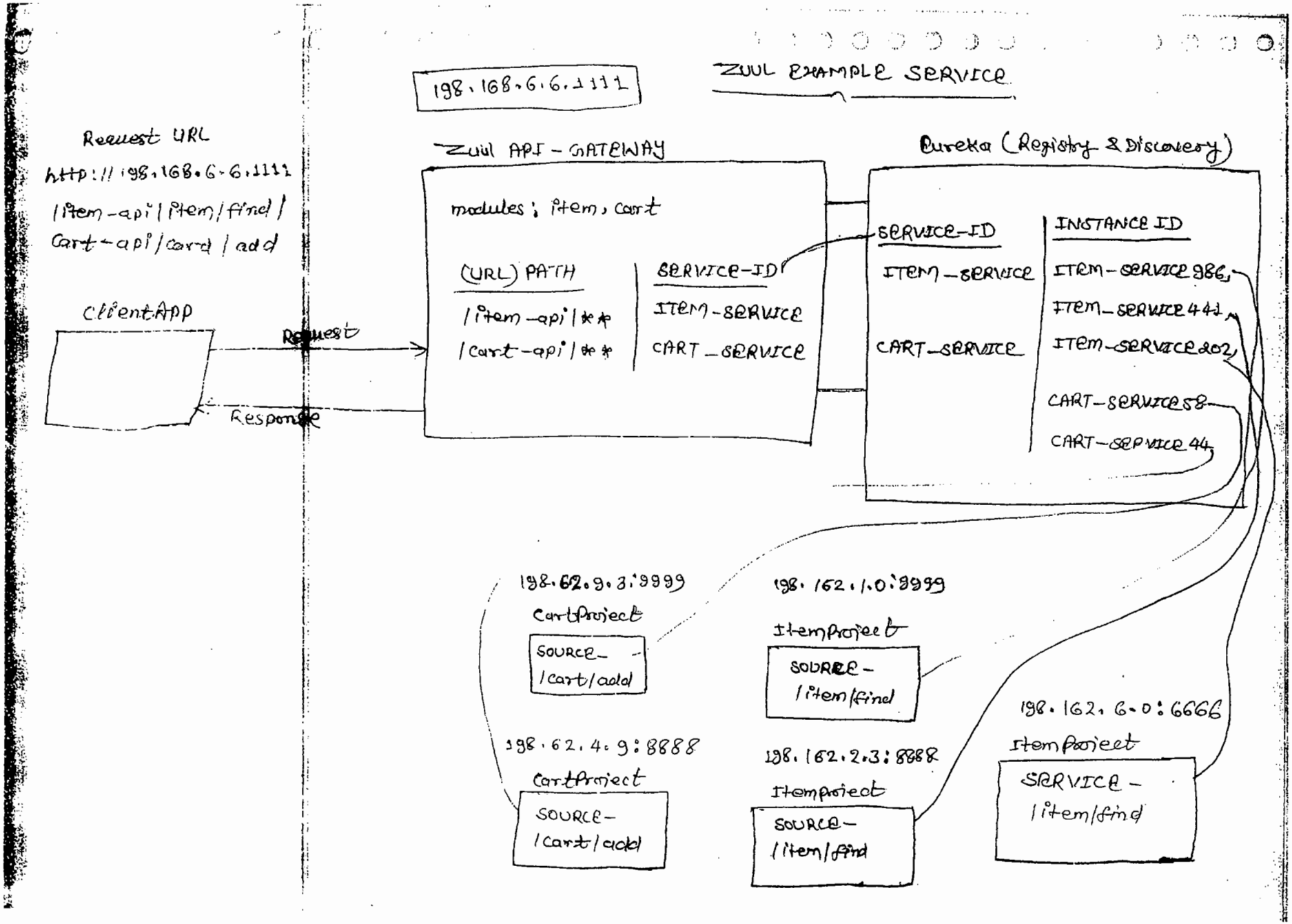
→ Provides Routing based on API (Path/url).

→ Zuul must be registered with Eureka server.

→ In zuul server project, we should provides module details like paths, services-id using application.properties (or .yml)

→ Example application.properties

```
zuul.routes.item-path = /item-api/**
zuul.routes.item.service-id = ITEM-SERVICE
zuul.routes.cart-path = /cart-api/**
zuul.routes.cart.service-id = CART-SERVICE
```


198.168.6.6.1111
ZUUL EXAMPLE SERVICE
Request URL
http://198.168.6.6.1111
/item-api/item/find/
Cart-api/cart/add
Zuul API-GATEWAY
modules: item, cart
(URL) PATH
SERVICE-ID
/item-api/**
ITEM-SERVICE
/cart-api/**
CART_SERVICE
Eureka (Registry & Discovery)
SERVICE-ID
INSTANCE ID
ITEM-SERVICE
ITEM-SERVICE 986
ITEM-SERVICE 441
CART-SERVICE
ITEM-SERVICE 202
CART-SERVICE 58
CART-SERVICE 44
clientApp
Request
Response
198.62.9.3:9999
CartProject
SOURCE-
/cart/add
198.162.1.0:9999
ItemProject
SOURCE-
/item/find
198.62.4.9:8888
CartProject
SOURCE-
/cart/add
198.162.2.3:8888
ItemProject
SOURCE-
/item/find
198.162.6.0:6666
ItemProject
SERVICE-
/item/find

⇒ Equal application.yml code :-

```
zuul :
    routes :
        item :
            path : /item-api/**
            service-id : ITEM-SERVICE
        cart :
            path : /cart-api/**
            service-id : CART_SERVICE
```

24/04/2019

* If Modules are provided in zuul then only module name get changed in keys. Consider below Example :-

MODULE	PATH	SERVICE-ID
Product -	/path-api/**	PROD-SERVICE
student -	/std-api/**	STD-SERVICE

= application.properties

zuul.routes.product.path = /prod-api/**

zuul.routes.product.service-id = ITEM-SERVICE

zuul.routes.student.path = /std-api/**

zuul.routes.student.service-id = STD-SERVICE

application.yml

```
zuul:
  routes:
    Product:
      path: /item-api/**
      service-id: ITEM-SERVICE
    Student:
      path: /std-api/**
      service-id: STD-SERVICE
```


*) Working with ZUUL Filters :-

Filters are used to validate request and construct valid response.

In simple we can also call as "PRE-POST" processing logic.

ZUUL filter provides even extra types like ROUTE FILTERS and ERROR FILTERS

→ When client made request to ZUUL Then .PreFilter. gets called automatically.

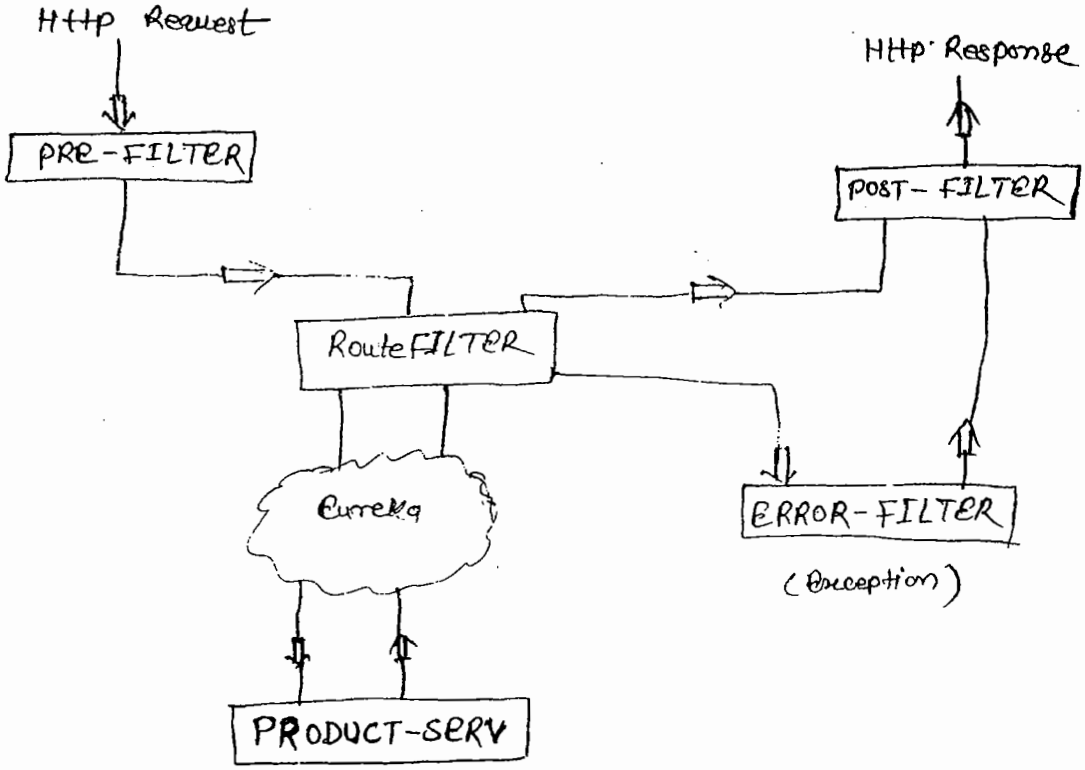
→ After validating, request is dispatched to Route Filter.

→ Route Filter is like 2nd level validation at Required @SERVICE level.

→ Route Filter will dispatch Request to one microservice based on Service-id.

→ If microservice is not executed properly (ie throwing exception) Then Error Filter is called.

→ Finally POST Filter works on HTTP Response (adds Headers, Encode data, provide info to client ...etc) in case of either Success or Failure.



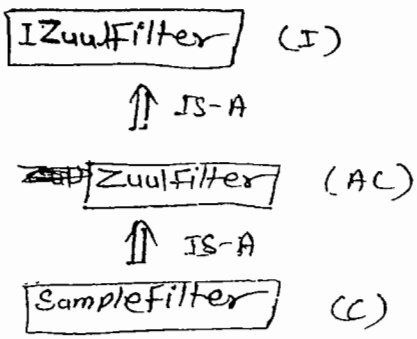
→ Here, Filter is a class must extends one abstract class "ZuulFilter" provided by netflix API.

→ We can defines multiple Filters in one application. Writing Filters are optional.

→ While creating Filter class we must provide Filter order (0, 1, 2, ...) and FilterType ("pre", "route", "error", "post")

→ Two Filter of same type can have same order which indicates any Execution order is valid.

→ We can enable and disable Filters using its flags (true/false)



→ To indicate Filter Type, we use its equal constants (public static final String variables), provided as:

Type	Constant
Pre	PRE_TYPE
route	ROUTE_TYPE
error	ERROR_TYPE
Post	POST_TYPE

→ All above constants are defined in FilterConstants (c) as global variables (public static final String)

→ Write one class in Zuul Server and extend ZuulFilter Abstract class, override below method (4) in your Filter class.

a) shouldFilter() — must be set to 'true' if value is set to false then filter will not be executed.

b) run() — Container Filter Logic Executed once then Filter is called.

c) filterType() — Provide Filter Constant must be one Type (pre, post, route, error)

d) FilterOrder() — Provider order for Filter. Any int Type number like: 0, 56, 98

25/04/2019

Project Lombok :——

This is open source JAVA API used to avoid writing (or generating) common code for Bean/model/entity classes.

That is like:

1. Setters and Getters
2. toString() method
3. Default and parameterized Constructor

4. hashCode() and equals() methods.

→ Programmer can write these method manually or generate using IDE. But if any modification(s) are done in those classes then again generate set/get methods also delete and write code for new : toString, hashCode, equals and param const (it is like repeated task)

→ By using Lombok API which reduces writing code or generating task for Beans. Just apply annotations, it is done.

→ To use Lombok, while creating spring Boot project choose Dependency : Lombok (or) Add below dependency in pom.xml

(For Spring Boot project : Do not provide version provided by spring boot parent only.)

```
<dependency>
    <groupId>org.projectLombok</groupId>
    <artifactId>lombok</artifactId>
    <scope>provided</scope>
</dependency>
```

(for Non spring Boot projects)

```
<dependency>
    <groupId>org.projectLombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.6</version>
</dependency>
```

---- Install Lombok in IDE ----

#1 open STS/Eclipse (any workspace)

#2 create spring Boot Project with lombok Dependency (or) maven Project with above Lombok Dependency.

#3 update maven project (Alt+F5)

#4 close STS.

#5 Go to Lombok JAR Location

ex: C:\Users\<userName>\.m2\repository\org\projectlombok\lombok\1.18.6

#6 open Command Prompt Here

> shift + Mouse Right click

> choose "open Command Window Here"

> type cmd given below

```
java -jar lombok-1.18.6.jar
```

> wait for few minutes (IDEs are detected)

> Click on Install/update

> finish

#7 open STS / Eclipse and start coding...

--- Example Application ---

#1 Create spring Boot starter project

> File > new > spring starter project > Enter Details:

GroupId : com.app

ArtifactId : SpringBootLombokEx

Version : 1.0

> choose dependency : lombok (only)

#2 Create model class which lombok annotations

```
Package com.app.model;
// Ctrl+shift+O
import lombok.*;
@Getter // generates get methods
@Setter // generates set methods
@ToString // overide tostring method
@NoArgsConstructor // generate def. const
@RequiredArgsConstructor // Param const
@EqualsAndHashCode // override hashcode, equals
public class Employee
{
 @NonNull
 Private Integer empId,
```

```
@NonNull
private String empName;
private Double empsal;
}
```

** To use @RequiredArgsConstructor which generates constructor using variables annotated with @NonNull. if no variable found having @NonNull, then it is equal to generating "Default constructor" only.

--- Runner code ---

```
package com.app.Runner;
//ctrl+shift+o
@Component
public class MyAppRunner implements CommandLineRunner
{
public void run(String ...args) throws Exception
{
Employee e1 = new Employee();
e1.setEmpId(10);
e1.setEmpName("AA");
e1.setEmpSal(1.5);

Employee e2 = new Employee();
e2.setEmpId(10);
e2.setEmpName("AA");
e2.setEmpSal(2.3);
```

```
        Sopln(e2.equals(e1));
    }
}
```

**NOTE: **

(1) Apply @Data (Package : lombok.Data) over Bean/model which generates set, get, tostring, equals, hashCode and Required Args Constructor.

Ex:

```
@Data
public class Employee{.....}
```

26/04/2019

Spring Boot Message Queues (MQ) :-

In case of ~~realtime~~ realtime application large data needs to be transfered and processed.

Like Google server to Amazon, facebook, ... etc.

→ Data (Message) Transfer can be done using Message Queues.

→ Message Queues are used for

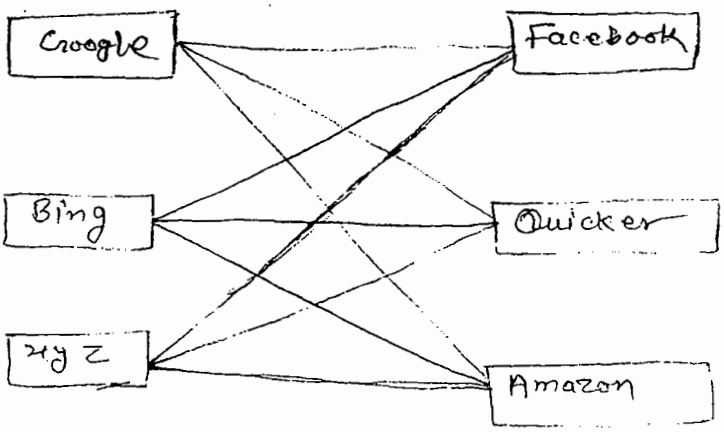
(a) Log aggrigation (Read Large log files from server and send ~~to another place~~)

(b) Web Activities (Know what users are searching and provide related links and ADs in client websites)

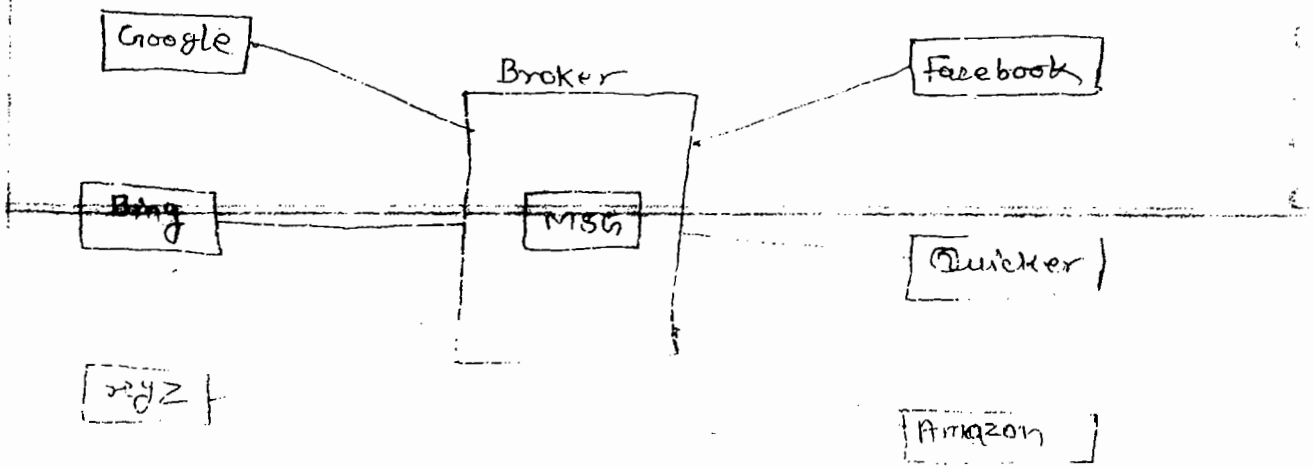
© Command Instruction (Send message to Printer / Email one Invoice on receive data ... etc).

ⓓ Data Streaming (Read Large Data from files, Networks, Databases continuously) etc...

→ In case of multiple clients share data without MQ, may look like this:



A Design with Message Queues (MQ) :-



→ MQ can be implemented using Language based Technologies (or API)

Ex: JMS (Java message Service)

→ Global MQ (between any type of client Advanced Message Queuing Protocal (AMQP)

→ Apachee ActiveMQ, Rabbit MQ, Apache Atrims are different Service Providers (Broker Softwares) for JMS.

→ JMS Kafka is a service provider (Broker software) for AMQP

[redacted]

Spring Boot with Apache ActiveMQ :—

[redacted] Java JMS is simplified using spring Boot which reduces writting basic configuration for connectionfactory, Connection, Session, Destination creation, Send/Receive message etc...

Java supports 2 types of clients. Those are:

(a) Producer (client) : Sends message

(b) Consumer (client) : Read Message

message exchanged using message Broker called as MOM (Message Oriented Middleware). Apache ActiveMQ is a MOM Software.

Types of communications in MQS :—

1. P2P (peer-To-peer) :—

   Sending 1 message to one Consumer.

2. Pub/sub (Publish and Subscribe) :—

   Sending 1 message (same copy) to multiple consumers.

*** JMS supports two types of communications

Note:

(a) Destination is a special memory created in MOM which holds message

(b) Here Queue Destination is used for P2P Topic Destination is used for Pub/Sub.

Limitation of JMS :—

ⓐ used between Java Applications only.

ⓑ message (Data) size should be ~~stn~~ Smaller.

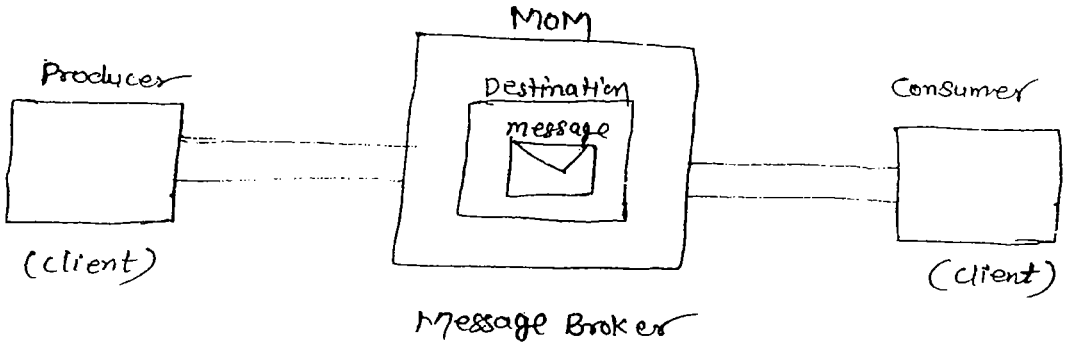
ⓒ only ~~use~~ one MOM (one Intance) run at a time.

ⓓ Large data takes lot of time to Process.

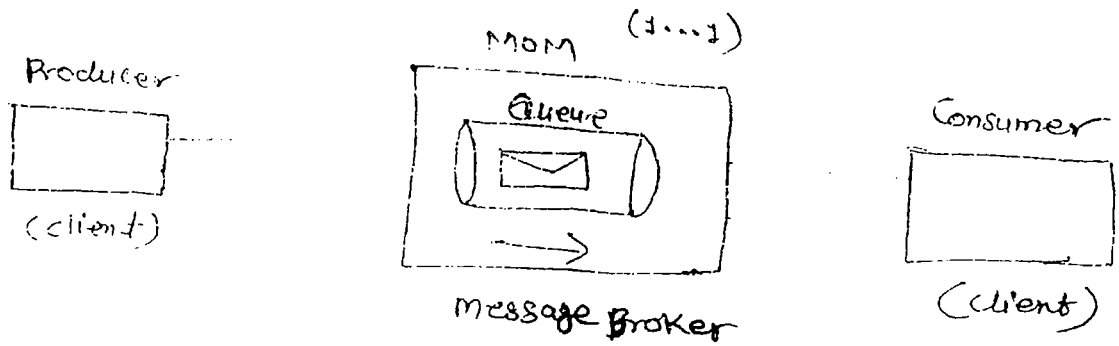
ⓔ If pub/sub model is implemented with more consumers then process will be very slow.

ⓕ Data ~~many~~ may not be delivered (Data lose) in case of mom Stops ~~are~~ or Restart

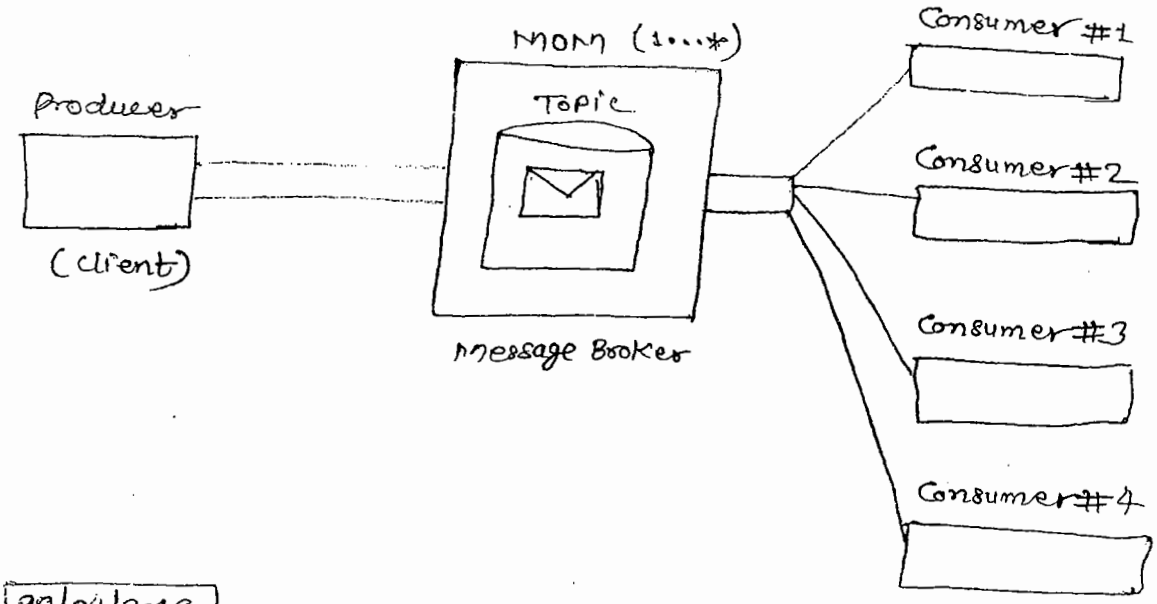
Generic Design :-



P2P : peer-To-peer : (Queue)



Pub/Sub : publish and subscribe (Topic)



29/04/2019

1. * Steps to implement ActiveMQ application in Spring Boot :-

Step #1 : Create one spring boot starter application using dependencies:

ActiveMQ (JMS)

(or add below dependency in pom.xml)

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

Step#2: In application.properties file provide common keys like MQ brokerUrl, user, password.

→ If we do not specify any type then it behaves as P2P (Queue). To make it as pub/sub (Topic).

---application.properties---

```
spring.activemq.broker-url = tcp://localhost:61616
spring.activemq.user = admin
spring.activemq.password = admin
spring.jms.pub-sub-domain = false
```

Step #3 If Application is producer type then use JmsTemplate object and call send() which will send message to MOM.

Step #4 If Application is Consumer type then define one listener class using destination.

Use code;

```
@JmsListener(destination = "___")
```

It must be enabled using code;

```
@EnableJms
```

→ In case of JmsTemplate(), @EnableJms is not required.

Download and setup for Activemq:

Download link:

→ Go to below location

https://activemq.apache.org/components/classic/download/

→ Click on : apache-activemq-5.15.9-bin.zip

→ Extract to one folder

→ Go to ....\apache-activemq-5.15.9\bin\win64

→ Double click on activemq.bat file

→ Go to Browser and type url:

http://localhost:8161/admin

→ To create multiple consumer applications in same or different workspaces add below

spring.jms.pub-sub-domain = true

in application.properties file (or yml)

(download code FB)

30/04/2018

~~Apache Kafka~~

~~Apache Kafka~~ ?

Kafka

~~Spring Boot Apache Kafka~~ Integration :—

Apache Kafka is used for

a. Distributed Messaging System, which works for multiple Destinations of any type(any Language + plugin required), to send or receive messages.

b. Supports Realtime Data Streaming. It means read continous an large data from External sources like Flat Files, Database, networks, ... etc.

c. Message Brokers will persist the message (Save into their memory). to avoid data lose in case of consumer non-available or Broker is down.

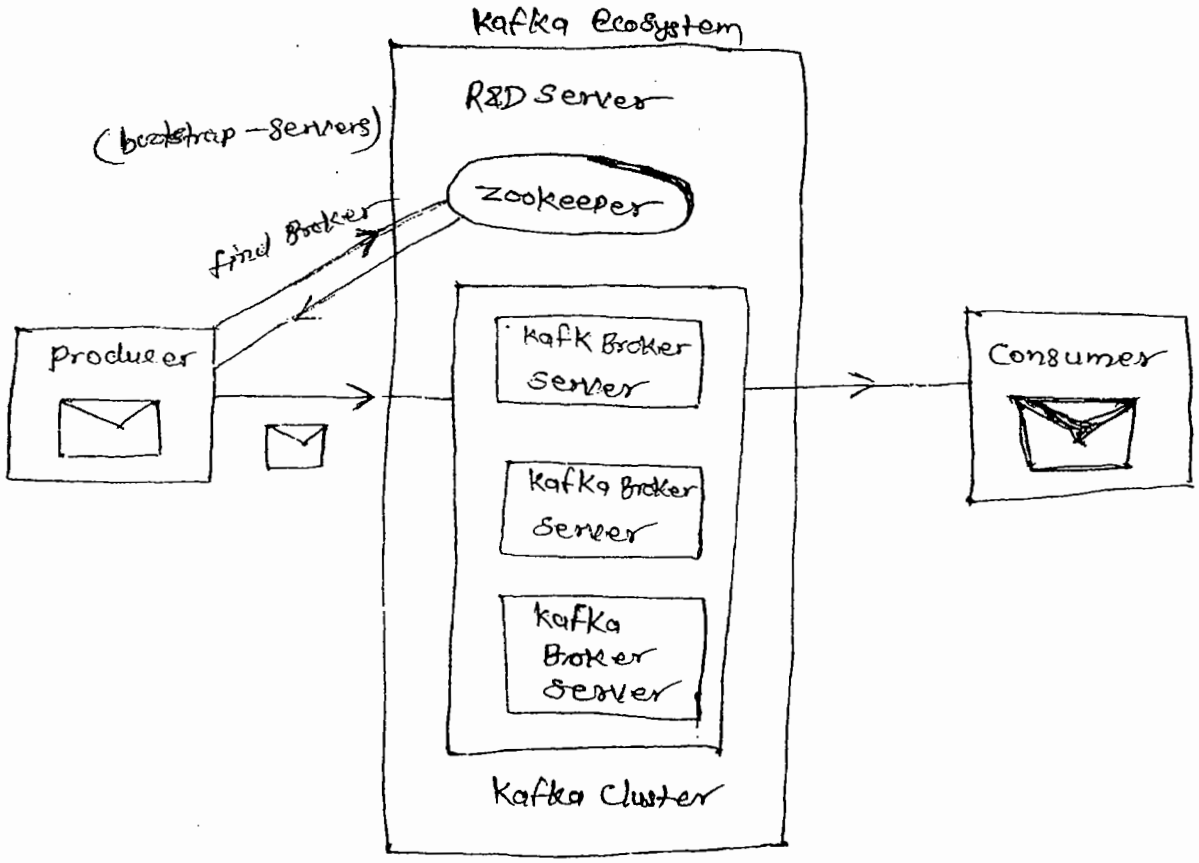
d. Kafka default communication type is pub/sub (Topics).

e. Kafka Supports Load Balancing for Broker softwares to make execution faster. It is known as Kafka Cluster.

f. All These ~~Broker instances~~ must be Registered with Registry and Discovery (R&D) Server. Kafka comes with default Zookeeper R&D Server.

g. This complete kafka software is called as kafka ecosystem (= kafka cluster + R&D server)

h. kafka works as protocal independent. (ie works for tcp, ftp, smtp, http,... etc.)



----Execution Flow ----------

→ Producer Application should get Message Broker details from R&D server (Zookeeper) (Known as bootstrap-server).

→ Producer get's unique-id for message Broker Server and sends message to Broker.

→ Message Broker will send this message to one or multiple Consumers

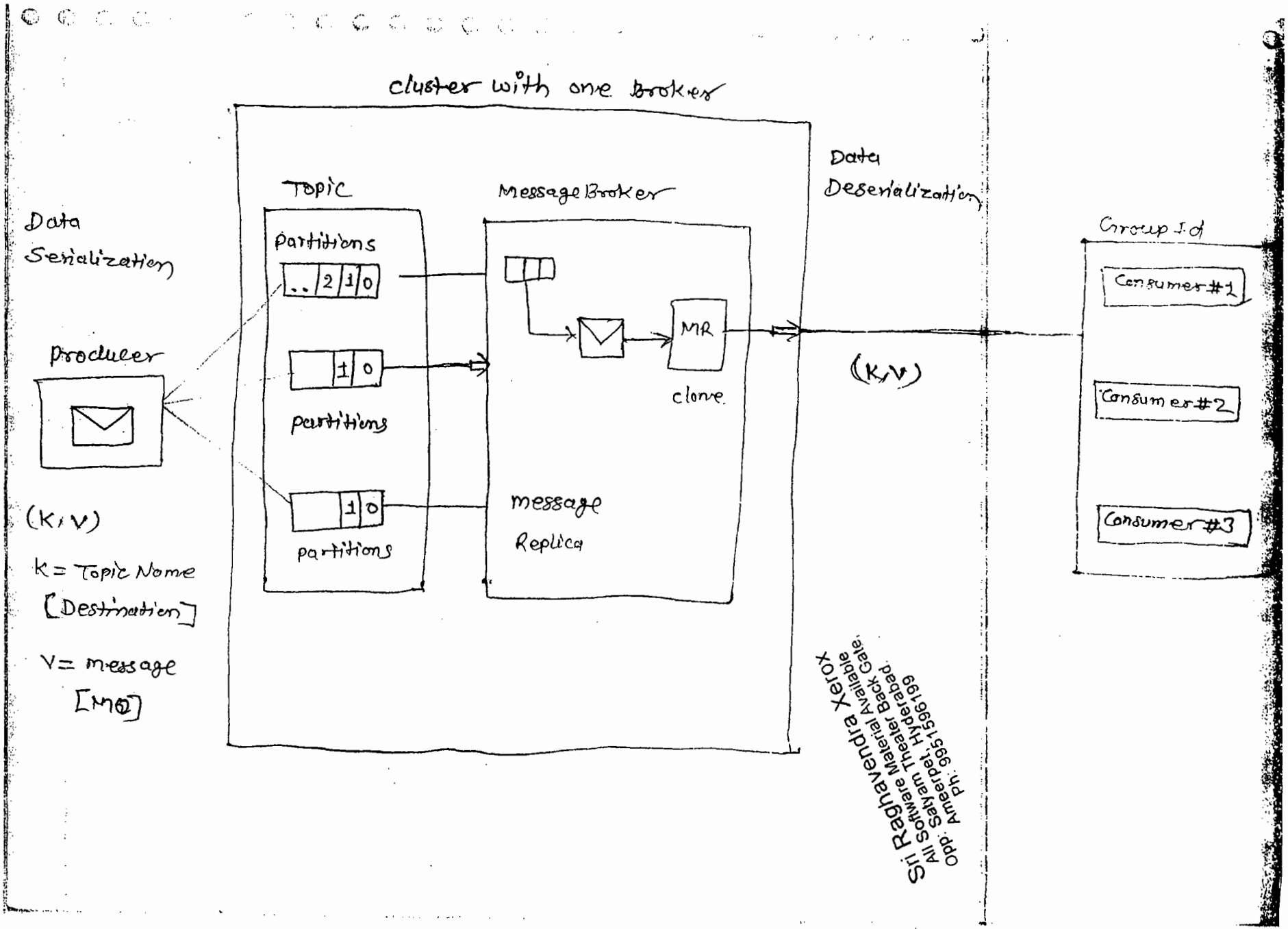
→ Producer sends data <K,V> format in Serialized (converting to binary / characters formats). Here K = Destination (Topic name) and V = message.

→ Every Message will be partitioned into multiple parts in Topic (Destination) to avoid large data sending, by making into small and equal parts (some times size may vary).

→ Broker reads all partitions data and creates its replica (clone / mirror obj) to send message to multiple Consumers based on Topic and Group-Id.

→ At Consumer side Deserialization must be applied on K,V to read data.

Consumer should also be linked with bootstrap-server to know its broker.



*) Partitions are used to breakdown large message into multiple parts and send same to multiple brokers to make data distribution in parallel.

*) Message Replica:

It Creates multiple copies to one message to publish one message to multiple Consumer.

01/05/2019

Kafka Producer and Consumer Setup Details :-

For Producer Application we should details in application.properties (or .yml)

=> Those are :

bootstrap-servers = localhost:9092

key-serializer = StringSerializer

value-serializer = StringSerializer

-> By using this spring boot creates instance of "KafkaTemplate<K,V>" then we can call send(K,V) method which will send data to Consumer.

Here: K = Topic Name , V = Data/message

For Consumer Application we should Provide details in application.properties (or .yml)

=> Those are :

bootstrap.servers = localhost:9092

key-deserializer =

value-deserializer = StringDeserializer

group-id = MyGroupId

→ By using this spring Boot configures the Consumer application, which must be implemented using:

@KafkaListener(topics="__", groupId="__")

*** bat files in Kafka to be Created ***

--cluster.bat--

→ Starts Zookeeper with Kafka cluster design

.\bin\windows\zookeeper-server-start.bat

.\config\zookeeper.properties

--server.bat--

→ Starts Kafka Server (message broker)

.\bin\windows\kafka-server-start.bat

.\config\server.properties

------ Coding Step's ----------

Step#1 Create new Spring Boot Starter project with dependencies:

we, kafka

GroupId : com.app

artifactId : SpringBootKafka

Version : 1.0

Step#2 add key = value pairs in application (.properties/.yml) file

```
server:
  port: 9988

spring:
  kafka:
    producer:
      bootstrap-servers: localhost:9092
      key-serializer: org.apache.kafka.common.serialization.StringSerializer
      value-serializer: org.apache.kafka.common.serialization.StringSerializer
    consumer:
      bootstrap-servers: localhost:9092
      key-deserializer: org.apache.kafka.common.serialization.StringDeserializer
      value-deserializer: org.apache.kafka.common.serialization.StringDeserializer
      group-id: groupId

my:
  app:
    topicname: sampletopic
```

Step #3 Define producer code

```
package com.app.producer;
// ctrl+shift+O
@Component
public String topic
@Component
public class Producer {
@Value("${my.app.topicname}")
private String topic;
@Autowired
private KafkaTemplate<String, String> template;
public void sendMessage(String message) {
    template.send(topic, message);
  }
}
```

Step #4 Define message storage class

```
package com.app.store;
// ctrl+shift+O
@Component
public class MessageStorage {
private List<String> list = new ArrayList<String>();
```

```
    public void put(String message) {
        list.add(message);
    }
    public String getAll() {
        return list.toString();
    }
}
```

Step#5 Define Consumer class

```
package com.app.consumer;
//ctrl+shift+o
@Component
public class Consumer {
    @Autowired
    private MessageStorage storage;

    @KafkaListener(topics = "${my.app.topicname}", groupId = "group1")
    public void consume(String message) {
        storage.put(message);
    }
}
```

Step #6 Define Kafka Rest Controller class

```
package com.app.controller;
//ctrl +shift +o
@RestController
public class KafkaRestController {

  @Autowired
  private Producer producer;

  @Autowired
  private MessageStorage storage;

@RequestMapping("/send")
public String readInMessage(@RequestParam String message) {
    producer.sendMessage(message);
    return "message sent!!";
}
@RequestMapping("/view")
public String viewOutMessage() {
    return storage.getAll();
}
}
```

*) Run Starter class and enter urls

http://localhost:9988/send?message=OK

http://localhost:9988/view

02/05/2019

Spring boot with Apache Camel :—

Routing :—

It is a process of sending large data from Application (Source) to another Application (Destination).

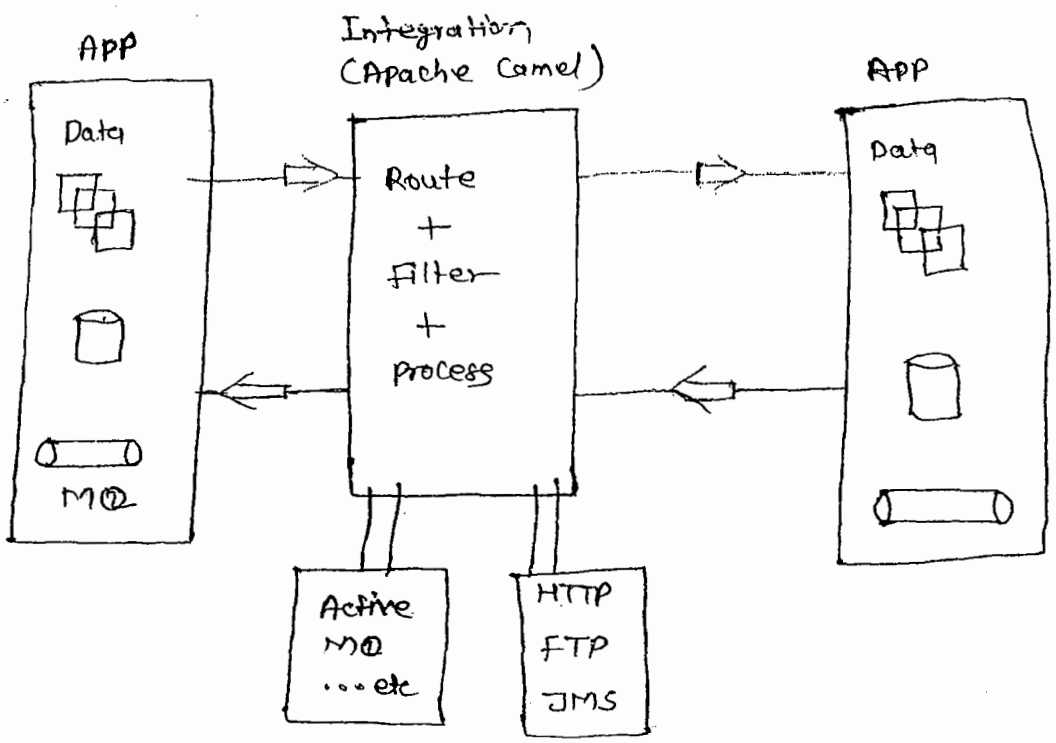
Here data can be File system (.xml, .txt, .csv, .xls=1, .json, ...etc), Database (Oracle DB, MySQL DB) or message Queues using JMS (Active MQ) etc...

Apache Camel is open source and light weight "Conditional based Routing Engine" which supports filtering and processing.

Apache Camel also supports different languages like PHP, Python, Javascript .... etc.

Compared to spring Batch Integration tool Apache Camel is a light weight tool.

Camel supports reading data from different source even like HTTP, FTP, JMS protocal Based.



--- Implementing Camel Routing in Spring boot ---

step#1 In pom.xml, we must add below dependency which supports spring boot integration

```
<dependency>
    <groupId> org.apache.camel </groupId>
    <artifactId> camel-spring-boot-starter </artifactId>
    <version> 2.23.2 </version>
</dependency>
```

Step#2 Camel avoid main method (thread) execution control and runs as independent. while working with spring boot, our starter class is main method. so we should add Key=Value in Properties file as.

--- application.properties ---

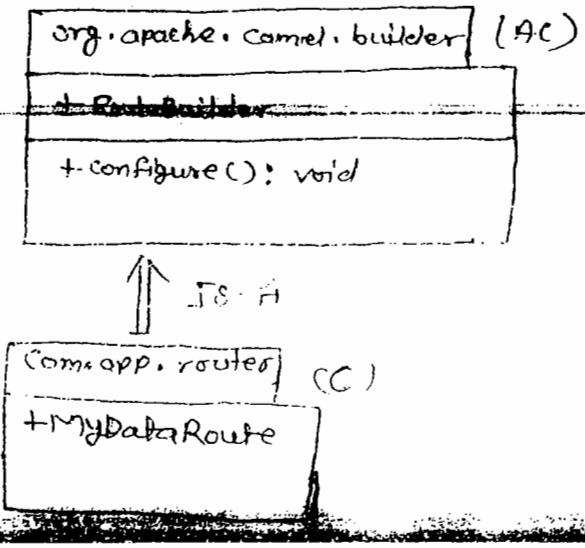
camel.springboot.main-run-controller = true

Step#3 Define one Route Builder class and configure details of routing, filtering and processing.

To implement this we need to write one class using (Abstract class) RouteBuilder provided by Apache Camel having one abstract method: configure() which contains coding format like:

```
from(SourceLocation)
  .[filter].[process].
  to(DestinationLocation)
```


Here Location can be URL/Local file systems, DB, JMS (message Queues) ... etc.



---- Coding Steps ----------

Step#1 Create spring Boot starter application with dependencies:

Apach Camel

GroupId : org.sathyatech

ArtifactId : SpringBootApachCamel

Version : 1.0

Step#2 open application.properties (.yml) and add main-method-Control key as true.

-- application.properties ----

camel.springboot.main-run-controller = true

-- application.yml ----

```
camel:
  springboot:
    main-run-controller = true
```

Step #3 Define Router Builder class with file transfer logic.

```
package com.app.routers;
// ctrl+shift+o
@Component
public class MyFilesRouter extends RouteBuilder
{
```

```
public void configure() throws Exception
{
  // with static location
  from("file:D:\\source").to("file:D:\\desti");
}
}
```

file:D:/source file:D:/desti

Step#4 Create two folder; source and desti in "D: drive".

Step#5 Start Application and place files in "D:/source" which will be copied to "D:/desti".

EIP Patterns by Apache Camel:-

EIP stands for "Enterprise Integration patterns" used to define short form code for

→ Data Routing
→ Data Filter
→ Data Processing

---#1---

```
from("file:source").to("file:desti");
```

It will copy file from source to destination by taking files in backup folder in source with name camel

It supports even same file sending with new data (operation to override program).

---#2----

from("file:source?noop=true").to("file:desti");

To avoid this we should set "No operation to override program" should be set as true.

--#3---

from("{{source}}").to("{{desti}}")

Here {{location}} indicates dynamic location passing using Properties/yml files, System args, Command Line inputs etc...

3/5/2019

*) Example EIPs :—

#1) Dynamic Location :—

Location (Source/Destination) can be passed at runtime using properties/yml files, config server, System arguments .... etc..

⇒ To indicate Location (URL, File System, DB, MQ ...) ~~Change at runtime~~

use format:

{{location}}

Code changes :-

ⓐ application.properties :

```
camel.springboot.main-run-controller=true
my.source=file:D:/Source?noop=true
my.desti=file:D:/desti
```

ⓑ Router class code :

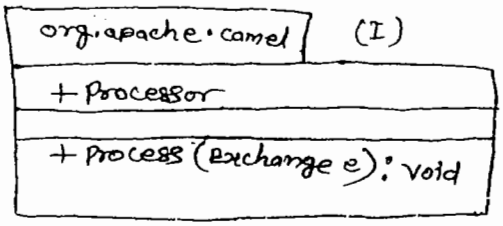
```
from("{{my.source}}")
    .to("{{my.desti}}");
```

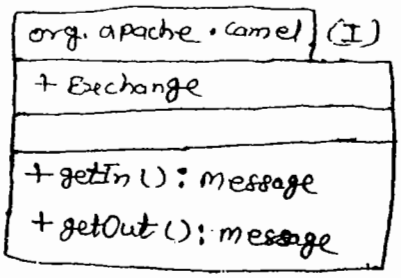
#2. Processing Data :-

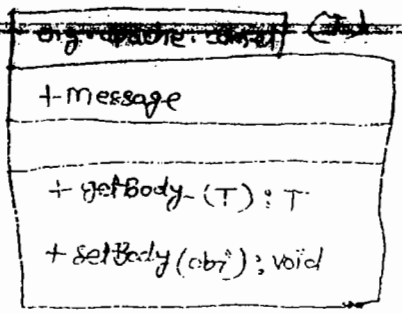
In real time data will be converted or modified based on requirements. Like, XML → JSON, JSON → simple Text, XML → CSV, CSV → SQL format ... etc.

⇒ i.e, Data converted from one format to another format which can be done using 3 supporting interfaces Those are :

(1) Processor
(2) Exchange
(3) Message







Code: write inside Router (Impl) class

```
from("file:D:/source")
 .process(new Processor() {
    public void process(Exchange ex) throws Exception {
 // 1# read input message
    Message m = ex.getIn();
 // 2# read body from message
    String body = m.getBody(String.class);
 // 3# do processing
    body = "modified::"+body;
 // 4# write data to output message
    Message m2 = ex.getOut();
 // 5# set body to the message
    m2.setBody(body);
  }
})
 .to("file:D:/desti?fileName=myfile.txt");
```


*) NOTE:-

Here file name indicates new name for modified message. It can also be passed at runtime.

Using Lambda Expression style:

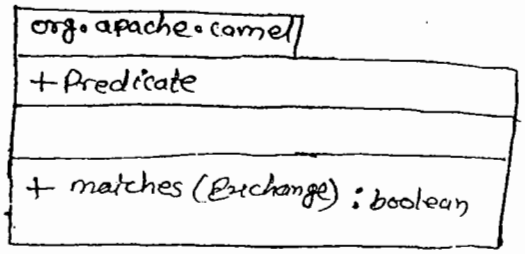
```
from("file:D:/source")
 .process(ex -> {
        String body = ex.getIn().getBody(String.class);
```

```
        body = "New AB modified :: " + body;
        ex.getOut().setBody(body);
    }
)
    .to("file:D:/destination?fileName=myfile.txt");
```

3. Filters using predicates :-

Predicate is a type expression which returns either true or false based on condition checking.

It true next level step are executed else false execution stopped here only.



⇒ ValueBuilder() is used to execute required predicates, using method: contains(), startsWith(), isNull()..... etc.

⇒ We can get ValueBuilder() object using method body(), header()

⇒ header() indicates checking on file name, extension, location ...etc.

⇒ body() indicates file data/content check.

ex#1 : file name having word sample

```
from("file:D:/source")
    .filter(header(Exchange.FILE_NAME)
        .contains("sample"))
    .to("file:D:/destination");
```


ex#2 : file body starts with java

```
from("file:D:/source")
    .filter body().startsWith("java")
    .to("file:D:/desti");
```

04/05/2019

Conditional based Routing :--

[choice - when - otherwise]

-> Filters are used to check Predicates and if true executes the next level process. But it do executes nothing for false case.

-> To handle switch-case concept for data routing use choice when-other.

format looks like :--

```
from("source")
.choice()
.when(condition #1).to("destination #1")
.when(condition #2).to("destination#2")
....
otherwise().to("destination#n")
```

--- Example Router code ----

```
from("file:D:/source")
.choice()
.when(body().startsWith("java"))
    .to("file:D:/desti?fileName=a.txt")
.when(body().startsWith("xml"))
    .to("file:D:/desti?fileName=b.txt")
```

```
.when(body().startsWith("json"))
    .to("file:D:/desti?fileName=c.txt")
.otherwise()
    .to("file:D:/desti?fileName=d.txt");
```

---Apache Camel Integration with Active MQ (JMS)---

patterns used to communicate with MQ using camel is: jms:<type>:<destination> Here type can be queue or topic.

example: jms:queue:info

jms:topic:news ...etc.

For this coding along with ActiveMQ and Camel Dependencies we should add activemq-pool and camel-jms integration dependencies.

Step#1 Create Spring boot starter project with dependencies:

Apache Camel, ActiveMQ

Step#2 add below dependencies in pom.xml file

```
<dependency>
    <groupId>org.apache.activemq</groupId>
    <artifactId>activemq-pool</artifactId>
</dependency>
```

```
<dependency>
    <groupId>org.apache.camel</
    <artifactId>camel-jms</
    <version>2.23.2</
</dependency>
```

Step#3 provide Camel and ActiveMQ properties

---application.properties---

camel.springboot.main-run-controller=true

spring.activemq.broker-url=tcp://localhost:61616

spring.activemq.user=admin

spring.activemq.password=admin

Step#4 Define Routers

--a. Camel To MQ Router --

```
package com.app.router;

@Component
public class CamelMqRouter extends RouteBuilder
{
```

```
    public void configure() throws Exception
    {
     from("file:D:/source").to("jms:queue:outdata");
    }
}
```

--b. MQ to Camel Router --

```
Package com.app.router;
//ctrl+shift+O
@Component
public class MqCamelRouter extends RouteBuilder
{
public void configure() throws Exception
{
~~from("file:D:~~ from("jms:queue:indata")
        .to("file:D:/dest?fileName=info.txt");
}
}
```

Step#5 Run starter class and start ActiveMQ using bat

Step#6 Login to MQ (http://localhost:8161/admin)

-> Click on Queue

→ Enter Queue Name and click Create

[2 queues : outdata, indata]

→ click on "Send To" option on "indata" Queue Enter message and Press Send. File will be copied to destination folder

→ Go to D:\source folder and place text file having any message. MQ reads this from source.

Task#1 : Camel → JDBC [MySQL DB]

Task#2 : ActiveMQ → Camel → JDBC [MySQL DB]

with processor and filters

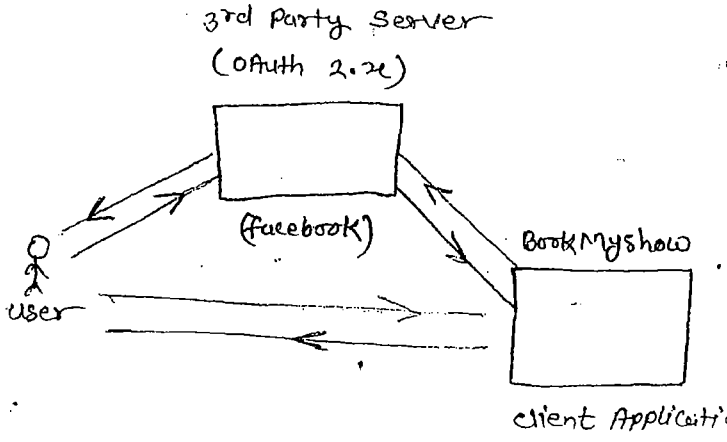
Hint :

jdbc:datasource. [Provide driver class, url, user name and password in application.properties]

06/05/2019

Open Authorization (OAuth 2.x)

OAuth 2.x is standard and framework which provides 3rd party Security services to client application which are registered on behalf of end user.



→ These 3rd party Application are also called as "Authorization and Resource Servers".

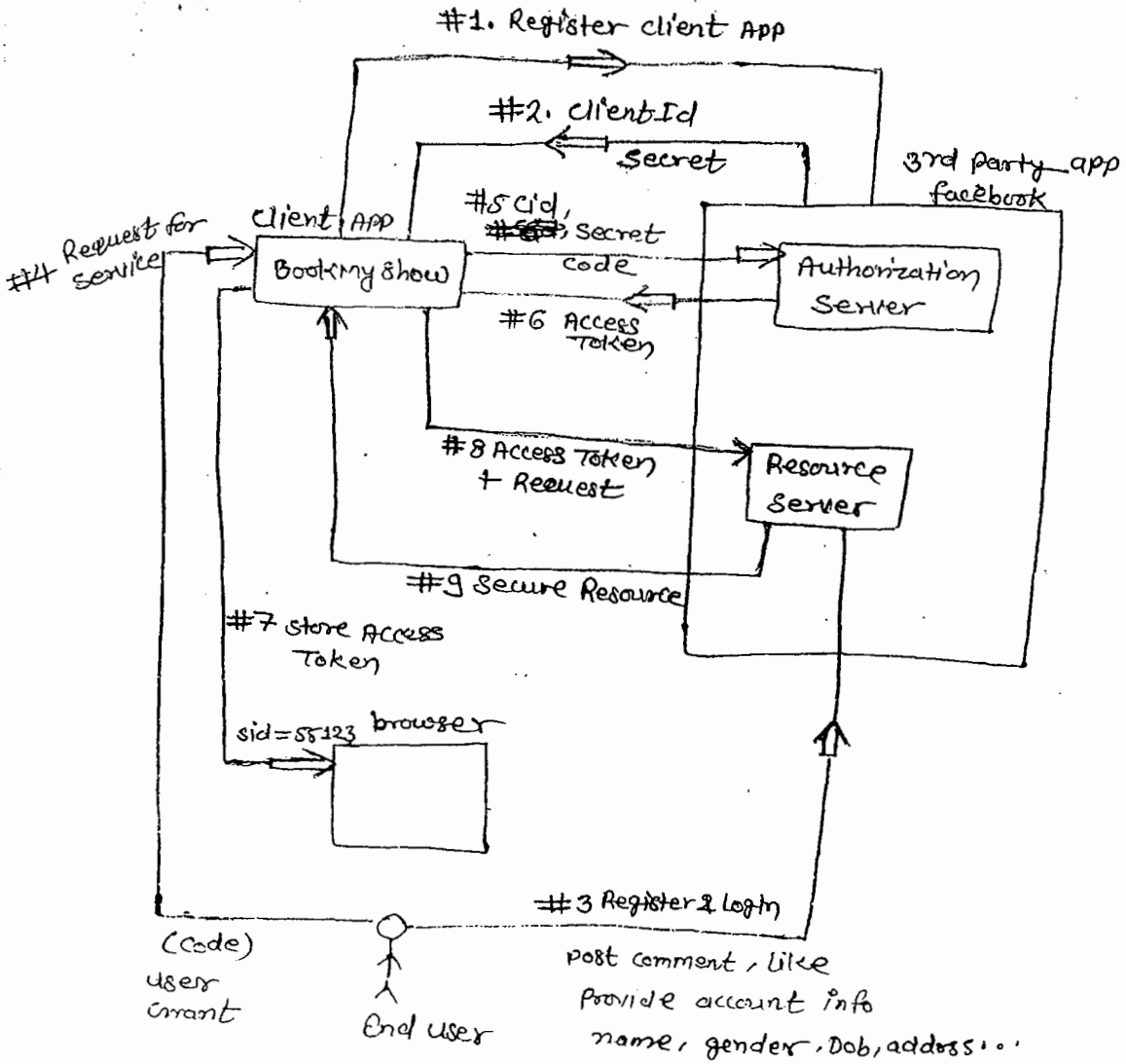
→ OAuth 2.x standard is widely used in small medium scal, daily used, business application.

→ OAuth 2 provide SSO (single sign on) for multiple application acting as a one services. (ex: google Accounts)

→ Example client Application are:
book my show, yatra.com, Quora, redbus, make my trip, netflix, mediafire, avast, carwale, zomato.

→ *** OAuth 2 may not be used for high level and large scale applications (Ex: Banking, Credit card, stock market, etc) these Spring Security ORM is used mostly.

→ few Authorization and Resource Servers are:


#1. Register client App
#2. clientId
Secret
3rd party app
facebook
#5 cid, Secret
code
Client App
#4 Request for service
BookMyShow
Authorization Server
#6 Access Token
#8 Access Token + Request
Resource Server
#9 Secure Resource
#7 store Access Token
sid=58123
browser
#3 Register & Login
(code)
User Grant
End User
post comment, like
Provide account info
name, gender, Dob, address...

07/05/2019

* One time setup for OAuth2 :-

#1 choose any one (or more) 3rd party "Authorization and Resource server".

Ex:- facebook, Google Cloud plateform (GCP), Github, LinkedIn, Twitter, etc...

#2 Here choosing facebook as 3rd party server for open Authorization. Link is : https://developers.facebook.com/

#3 Define one new (client) Application in FB Authorization server which generates ClientId (AppId) and ~~Secret~~ Secret (App Secret)

> Goto facebook developer page

> click on top right corner "MyApps"

> choose "Add New App"

> Provide Display name (ex: BootTest) and email Id : xyz@gmail.com

> Click on "Create AppId".

> Complete Security verification

> Click on "Dashboard"

> ~~Click on "facebook login" Setup button~~

> Click on "Settings" >> "Basic"

> Copy AppId (ClientId)

and App Secret (Client Secret)

#4 Create one SpringBoot Starter app with dependencies "Security" and "Cloud OAuth2".

Ex:- SpringBootOAuth2Ex1

--- pom.xml ---

```
<dependency>
    <groupId> org.springframework.boot </
    <artifactId> spring-boot-starter-security </
</dependency>
<dependency>
    <groupId> org.springframework.cloud </
    <artifactId> spring-cloud-starter-oauth2 </
</dependency>
```

#5 create application.yml file using below key value pairs

--- application.yml ---

```
server:
  port: 9898

security:
  oauth2:
    client:
      clientId: <your clientId Here>
      clientSecret: <your secret here>
```

accessTokenUri :

userAuthorizationUri :

tokenName :

authenticationScheme :

ClientAuthenticationScheme :

Auth Server

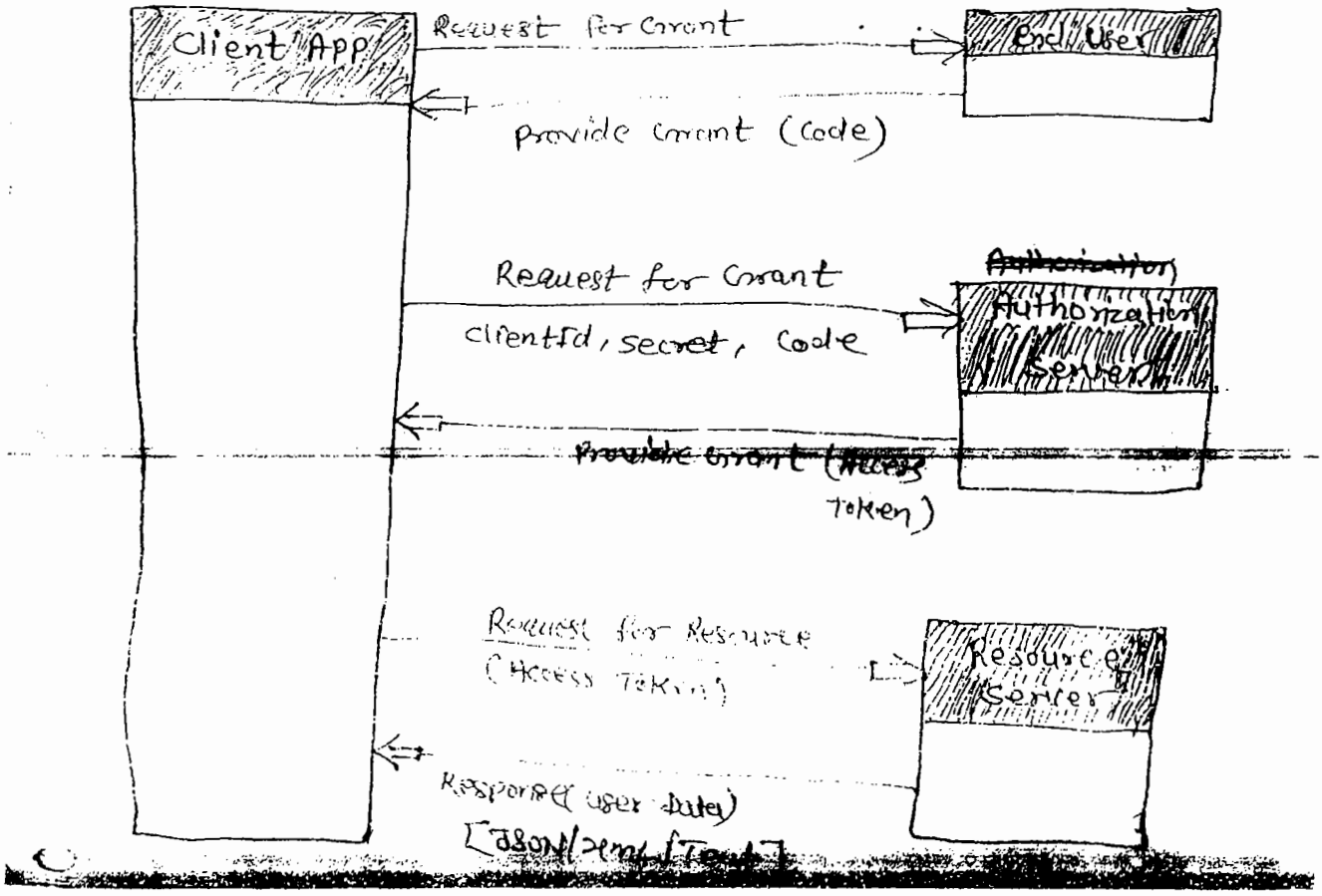
resource :

userInfoUri :

Resource Server

---- Request Execution Flow ----

Components Involved : ClientApp, User (Browser), Auth server (with Token Generator) and Resource server (User Data in xml/JSON)



08/05/2019

Step #6 Define SecurityConfig class (SecurityInit is not required, handled by Spring Boot only)

-> Here Authentication Details not required to Configure as we are using 3rd party Security services.

-> In Authorization config specify which URLs need type "Every one Access" [permitAll]

```
package com.app.config;
//ctrl + shift + o
@Configuration
@EnableOAuth2Sso
Public class SecurityConfig extends WebSecurityConfigurerAdapter
 {
  Protected void Configure (HttpSecurity http) throws Exception
   {
    http.authorizeRequests()
        .antMatchers("/", "/login").permitAll()
        .anyRequest().authenticated();
   }
}
```

Step #7 Define RestController for User (or any)

```
package com.app.controller;
@RestController
public class UserRestController
{
 @RequestMapping("/user")
 public Principal showUser(Principal p)
 {
  return p;
 }
 @RequestMapping("/home")
 public String showData()
 {
  return "Hello";
 }
}
```

Step #8 Define one UI page ex: index.html under src/main/resource in static folder.

--- index.html ---

```
<html><body>
<H1>Welcome to Login page</H1>
<a href="user">facebook</a>
(or)
```

```
<hr/>
<form action="#" method="post">
User : <input type="text">
pwd : <input type="password">
<input type="submit">
</form></body></html>
```

Step #9 Run Application and Enter URL http://localhost:9898

Step #10 Click on Facebook Link and accept name (or) enter details.

Spring Boot PCF Deployment

pivotal cloud foundry is a Cloud Deployment server provides service to Spring Boot and Microservices Applications mainly with all kinds of environment setup like Database, Server, log tracers ... etc.

~~#PCF~~ PFC Account Setup

Step #1 Goto ~~https://localhost~~

~~https://login.run.pivotal.io/login~~

and click on Create new account

Step#2 Enter details name, email, password and confirm Password and Register

Step#3 Go to Email account and Verify PFC Link

Step#4 Login to PCF account (Above URL)

Step#5 Provide Initial details like

→ Company Name

→ Mobile Number and Verify

→ Organization (org) name > finish

Step#6 Download and Setup PCF CLI

> click on "Tools" option

> Choose OS and Bit and Download Software

> Extract zip into Folder

> click on "cf_installer.exee"

> next > next > finish

Step#7 Login and Logout Commands

> Go to cmd prompt

> Login command is

cf login -a api.run.pivotal.io

Email >

Password >

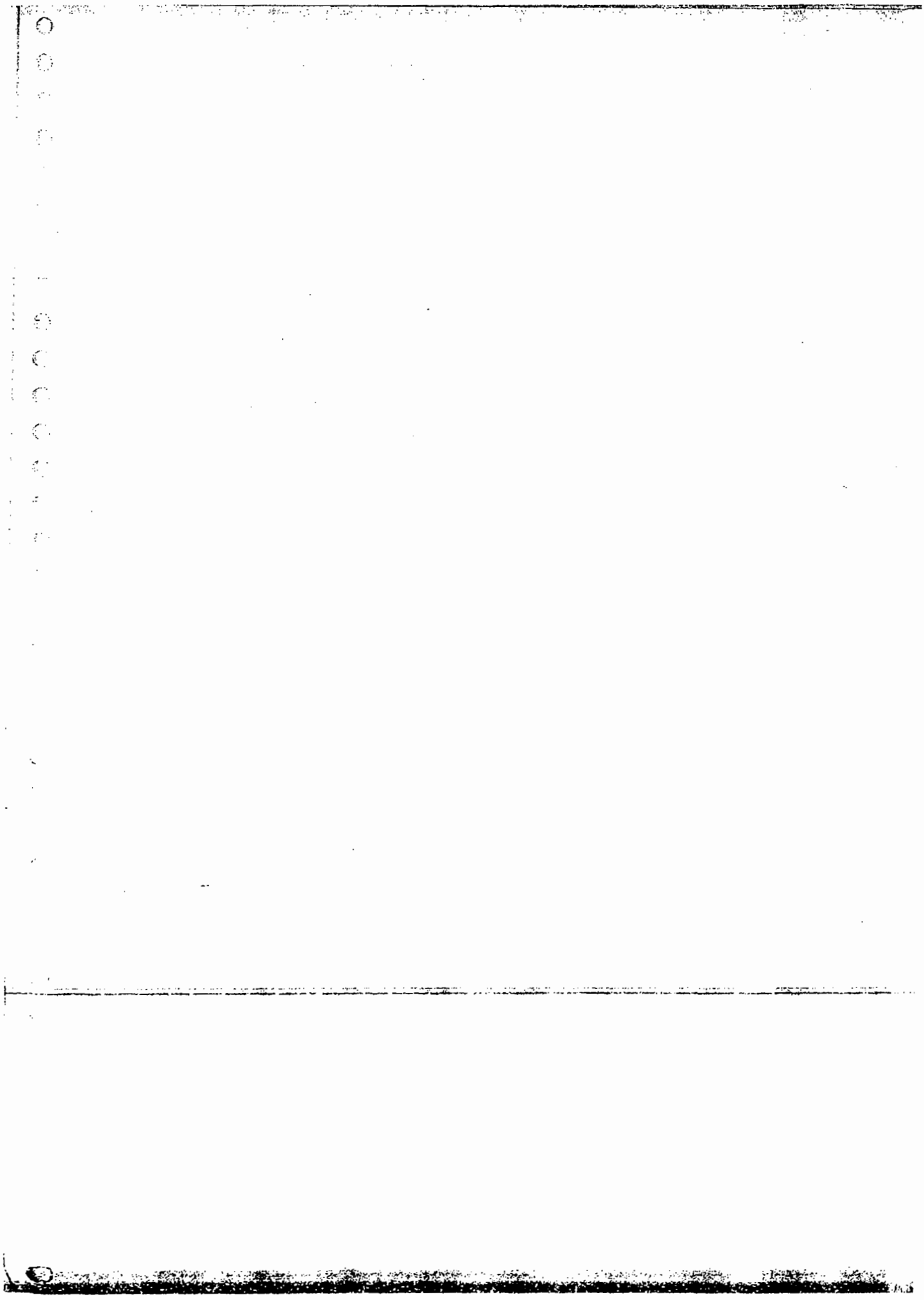
> Logout Command is

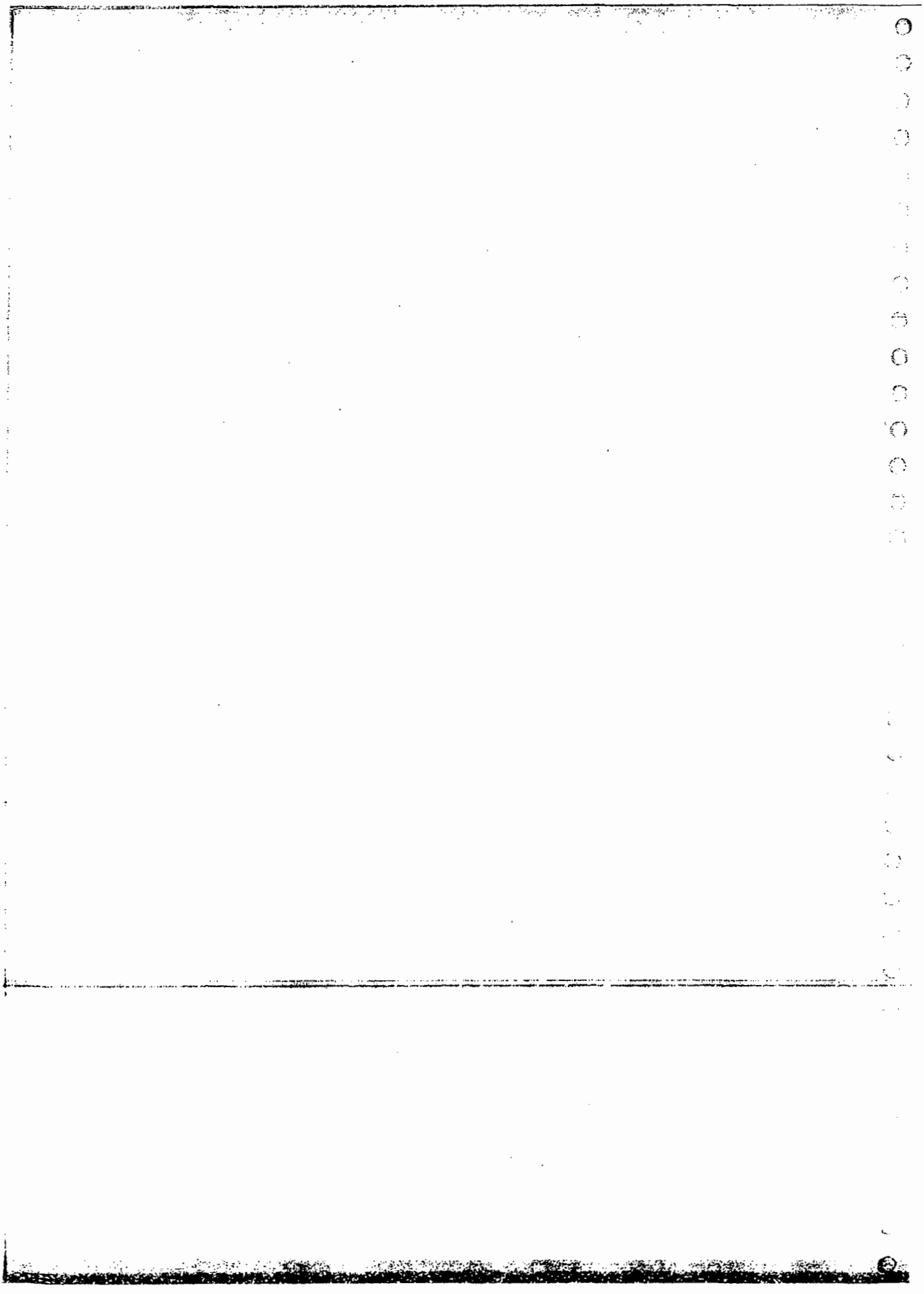
cf logout

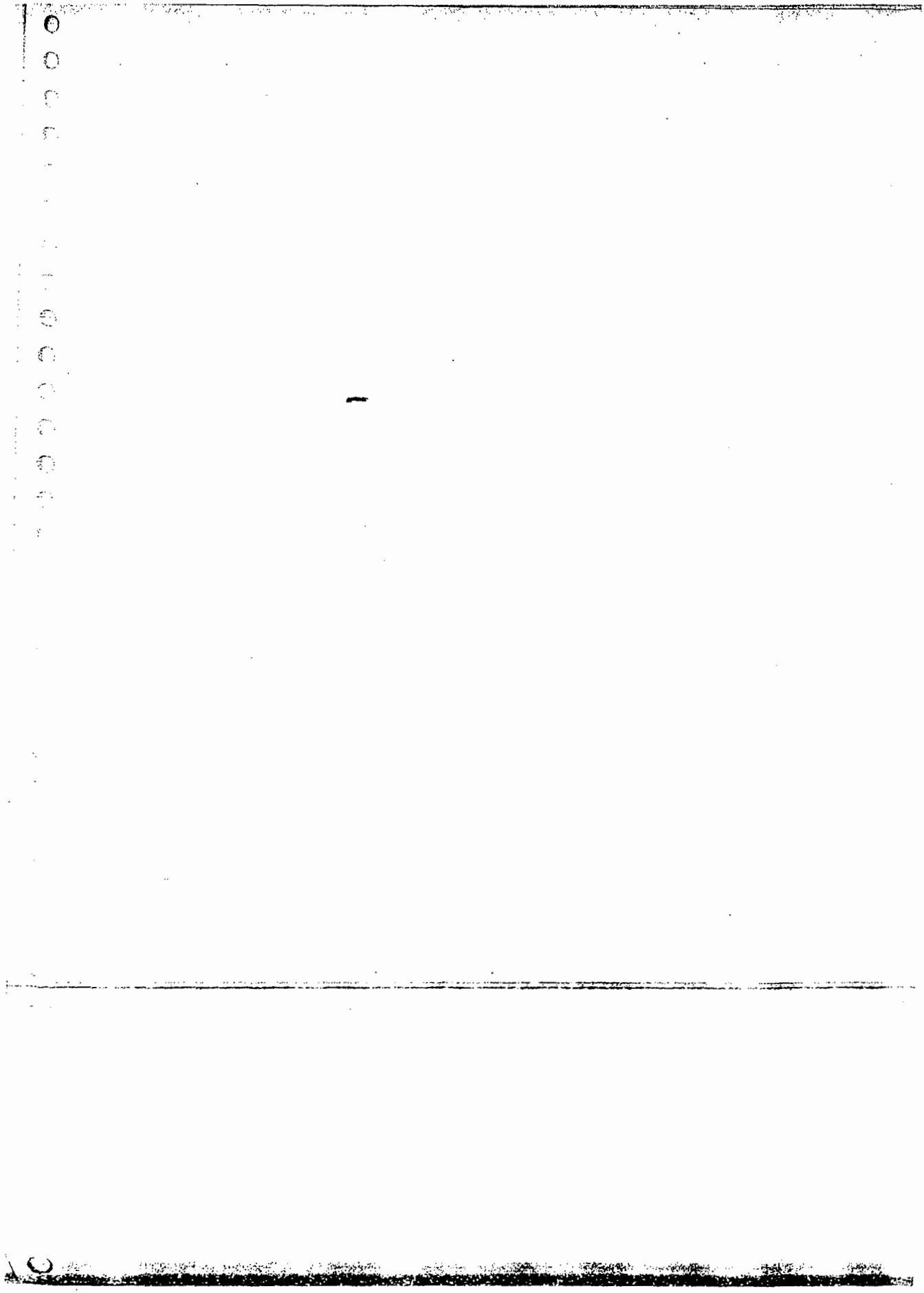
Q

Spring Boot WEB MVC + Thymeleaf + Data JPA + H2 (Curd Operation) + Bootstrap

9/5/2019







10/05/2019

Spring Boot Actuator :-

It Provides "Production Read EndPoints" which are used to give enter information of Production Server.

A server having application running and used in realtime by end customers is known as Production Server.

→ In Case of Production, we need some times information of it like,

ⓐ DB Status

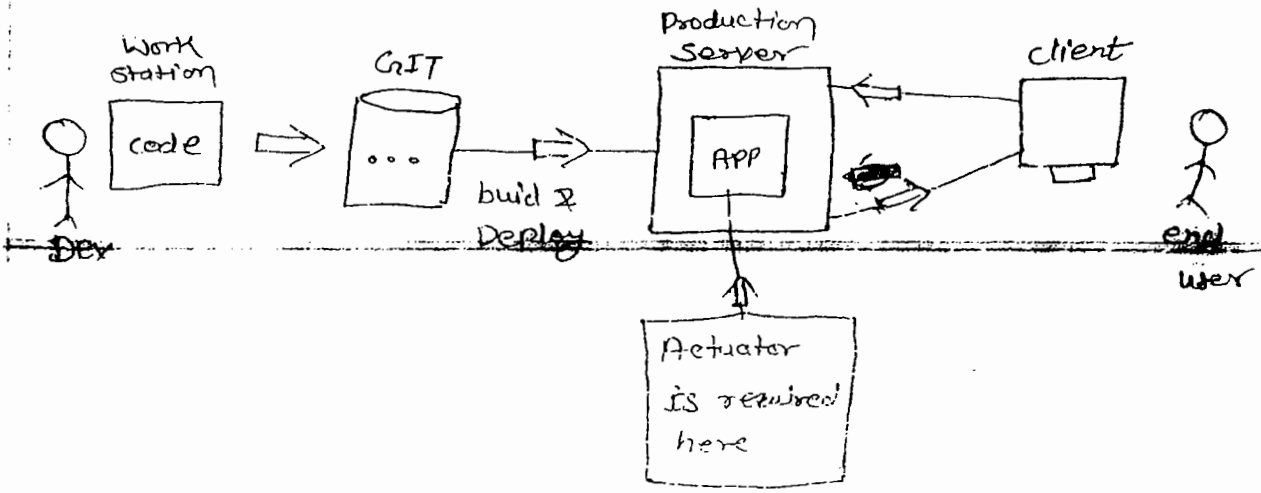
ⓑ Beans (objects) created.

ⓒ Memory (Heap, ThreadDump, ... etc) details

ⓓ Cache Managements...

ⓔ Is App connected to any Remote Networks or tools (Printer, Scanners...)

ⓕ logfiles and log levels updates etc.



*) By default Actuator enabled 2 endpoints those are:

/actuator/health and

/actuator/info

*) To enable all endpoints use code: ~~management.~~

management.endpoints.web.expose.include = *

in application.properties

Spring Admin UI (codecentric) :—

only Actuator if we implements in micro services then all details we should check manually. These all are made "ADMIN-UI Track and Trace" using SpringBoot Admin UI application.

For this every ~~application~~ microservices should have actuator and admin client dependencies and one Application with admin server and ui dependencies.

—Spring Boot Admin Server Set-up———

step#1 Create spring boot starter app with dependencies:

admin server and ui.

```
<dependency>
    <groupId>de.codecentric</
    <artifactId>spring-boot-admin-starter-server</
</dependency>
```

```
<dependency>
    <groupId>de.codecentric</groupId>
    <artifactId>spring-boot-admin-starter-ui</artifactId>
</dependency>
```

Step #2 at starter class level provide annotation

@EnableAdminServer

Step #3 provide details in application.properties

server.port=9999

-- Spring Boot (Any microservice) client App --

Step #1 create spring boot starter app with dependencies:

web, admin client, actuator.

Step #2 provide server details actuator endpoints in application.properties file.

server.port=8888

spring.boot.admin.client.url=http://localhost:9999

management.endpoints.web.exposure.include=*

step #3 Define one RestController (even servie, DAL...)

Run Adminserver and then clientApps

Enter URL', http://localhost:9999

Sri Raghavendra Xerox
All Software Material Available
Opp: Satyam Theater Back Gate,
Ameerpet, Hyderabad.
Ph: 9951596199