Date : 29/03/2021
Spring Boot 9AM
Mr. RAGHU

Connection Pooling:-

*) Pool is a group of Objects which are of same type.
Ex: Admin Pool = Admin Objects
    Spring Constant Pool = String Objects
    Product Pool = Product Objects
    ..etc

   Connection Pool = Database Connection
=> ie Makes our no. SQL operations increases for parallel operations.
=> By default Spring Boot uses HikariConfig that comes with default
   auto-configuration.

=> When we add Data JPA (or) JDBC API then by default below dependency
    is added and comes with auto-configuration.

```
  <dependency>
      <groupId>com.zaxxer</groupId>
      <artifactId>HikariCP</artifactId>
      <version>3.4.5</version>
      <scope>compile</scope>
    </dependency>
```

=> Spring Boot 1.x Was supporting Tomcat Connection Pooling,
   now it is moved to HikariCP (Spring Boot 2.x).

=> Tomcat CP is not recomanded. It is Server based.
   If we move from one server to another it will not work.

--API Details---
HikariConfig(C) that comes with all default values for Configuration
HikariDataSource(C) it is a impl class for DataSource(I) [javax.sql]
   Here DataSource means Database Connection.

========================Configuration Properties=============

1. Provide a name to Connection Pool Object
```
spring.datasource.hikari.pool-name=my-hikari-cp
```

2. By Default CP started with 10 Connection and later it takes
   our configuration. ie Default Pool size.
   (DEFAULT_POOL_SIZE = 10)

3. Max No.of Connections created at Pool (int number)
  It can not be <1.
```
spring.datasource.hikari.maximum-pool-size=20
```

4. Max no.of non-used/no work connections to be ketp in Pool
```
   spring.datasource.hikari.minimum-idle=15
```

5. Time in MillSec, for a Connection Timeout of A SQL query execution
```
spring.datasource.hikari.connection-timeout=180000
```

```
6. (Not a recomanded value)
  it will delete existed connection from
  pool after given time reached from connection
  creation time.

spring.datasource.hikari.max-lifetime=2500000

7. For Idle connection detection time to be considered (in Mill Sec)
 spring.datasource.hikari.idle-timeout=600000

*) Default IDLE_TIMEOUT out : IDLE_TIMEOUT = 10 Mins
*) MAX LIFETIME of a Connection : 30 min
*) CONNECTION TIMEOUT : 30 sec
*) Connection validation time out : VALIDATION_TIMEOUT  5 sec


Spring Config: DBCP Apache Data Base Connection Pooling 2.x
https://commons.apache.org/proper/commons-
dbcp/apidocs/org/apache/commons/dbcp2/BasicDataSource.html

Hibernate c3p0 (See Three Pee Ooo)
=================================================================
EHCache/JBoss Cahce (Hibernate )
Spring :
1. Hazelcast-cache (very slow proces/basic cache)
2. Redis Cache
https://www.youtube.com/watch?v=HBmlNMGh9O0
https://www.youtube.com/watch?v=IwYEdZOmY6g

#)Multiple Database Connections
https://www.youtube.com/watch?v=nzszxQbQ5WU

=========Add below configuration=====================
# Connection Pooling Details
spring.datasource.hikari.pool-name=my-hikari-cp
# Default started with 10 Connection
spring.datasource.hikari.minimum-idle=15
spring.datasource.hikari.max-lifetime=2500000
spring.datasource.hikari.idle-timeout=600000
spring.datasource.hikari.connection-timeout=180000
spring.datasource.hikari.maximum-pool-size=20

--------------------------------------------------------
RestTemplate(C) :-
 This is used to Make HTTP call to any webservice Application
 (java and non-java apps).

 That reads final response into ResponseEntity<T> or a direct time.
 It supports auto-type conversion of Input/Output into GlobalFormat
 (ie Object<-->JSON/XML).

 We can read even Header Information, Body, Status code...etc
 RestTemplate needs major input ie URL.

=> All Endpoint details must be given finally.
  Endpoint - URL, Http Method, Input, Ouput
```

```
             Supported Media Type details..etc

  Details required to make a HTTP call to a service is called
  Endpoint details.

=> RestTemplate supporta all Http Method Types (GET, POST...etc)

==========Provider App================================
Name : SpringBoot2RestProvider
Dep  : Web , lombok, devtools


*) RestController class
package in.nareshit.raghu.rest;

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

@RestController
@RequestMapping("/std")
public class StudentRestController {

        @GetMapping("/data")
        public ResponseEntity<String> showMsg() {
                return ResponseEntity.ok("Hello");
        }
}
============Consumer App============================
Name: SpringBoot2RestConsumerApp
Dep: Lombok, Web

*) AppConfig: Spring Boot never provides Auto-Configuration for
    RestTemplate, we should configure it manually when we are
    writing consumer application.

package in.nareshit.raghu.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.client.RestTemplate;

@Configuration
public class AppConfig {

        @Bean
        public RestTemplate rt() {
                return  new RestTemplate();
        }
}

*) Consumer code #1

package in.nareshit.raghu.runner;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Component;
import org.springframework.web.client.RestTemplate;

@Component
public class RestConsumerOne
        implements CommandLineRunner
{

        private static final Logger LOG =
LoggerFactory.getLogger(RestConsumerOne.class);

        @Autowired
        private RestTemplate rt;

        public void run(String... args) throws Exception {
                //1. Define URL of Provider
                String url = "http://localhost:8080/std/data";

                //2. Create RestTemplate object
                //RestTemplate rt = new RestTemplate();

                //3. Make call and get Response
                ResponseEntity<String> resp = rt.getForEntity(url,
String.class);

                //4. print details
                LOG.info("Status ID {}", resp.getStatusCodeValue());
                LOG.info("Status CODE {}",
resp.getStatusCode().name());
                LOG.info("Response Body {}", resp.getBody());
                LOG.info("Response Headers {}", resp.getHeaders());

                //5. Stop server manually
                System.exit(0);
        }
}
```

Execution Order
1. Provider Starter class
2. Consumer Starter class

Note:

a. getForEntity(String url,Class<T> clz):ResponseEntity<T>
  This method is given by RestTemplate(C) used to make
  HTTP calls using GET type, takes two inputs
  => URL, Expected Response Type
    ** String can hold any type of data
      (int, Double, boolean, JSON, XML...etc)

 => Above method returns data in ResponseEntity<T>
    that holds all response information.

b. System.exit(0); To stop main thread/Server, use this code.

```
c. Do not use same port number for Producer/Consumer App.
    consumer App :  server.port=9898

d. we can get only ResponseBody, not other details
    by using method getForObject(url,classType)

    String body = rt.getForObject(url, String.class);
    LOG.info("Response Body {}", body);

_______________________________________________________________
*) postForEntity(url,httpEntity,responseType,pathVariables)
   that returns ResponseEntity.

   HttpEntity = HttpHeaders + Body (JSON/XML)

  It is also called as request entity while making call using
  POST/PUT method.

========Producer RestController===========
package in.nareshit.raghu.rest;

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

import in.nareshit.raghu.model.Student;

@RestController
@RequestMapping("/std")
public class StudentRestController {


        @PostMapping("/create")
        public ResponseEntity<String> createStudent(
                        @RequestBody Student student
                        )
        {
                return ResponseEntity.ok("Student data is " +
student);
        }
}

===========Consumer RestController============
package in.nareshit.raghu.runner;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpHeaders;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Component;
import org.springframework.web.client.RestTemplate;

@Component
public class RestConsumerPOSTType
implements CommandLineRunner
{

        private static final Logger LOG =
LoggerFactory.getLogger(RestConsumerPOSTType.class);

        @Autowired
        private RestTemplate rt;

        public void run(String... args) throws Exception {
                //1. Define URL
                String url ="http://localhost:8080/std/create";

                //2. HttpEntity=header+body
                String body ="
{\"stdId\":100,\"stdName\":\"A\",\"stdFee\":300.0}";

                HttpHeaders headers = new HttpHeaders();
                headers.setContentType(MediaType.APPLICATION_JSON);

                HttpEntity<String> request = new HttpEntity<String>
(body, headers);

                //3. make request and get response
                // URL, HttpEntity, ResponseType,
pathVariable(optional)
                ResponseEntity<String> resp  = rt.postForEntity(url,
request, String.class);

                //4. print details
                LOG.info("Status ID {}", resp.getStatusCodeValue());
                LOG.info("Status CODE {}",
resp.getStatusCode().name());
                LOG.info("Response Body {}", resp.getBody());
                LOG.info("Response Headers {}", resp.getHeaders());
                //5. Stop server manually
                System.exit(0);
        }
}
===========================================================
```