HTML 08/08

08 August 2024 09:09 AM

- HTML
  - W3C, ECMA
- Basics of CSS
  - Selectors
  - Element
  - #id
  - .class
  - Universal
  - Grouping
- Java Script
  - DOM
  - Lang basics
  - Functions
  - Classes and objects
  - Arrow Functions (similar to lambda)
  - Sync/Asyn
  - Callback
  - Promise Async/await
  - DOM Manipulation

- AJAX
- HTTP REQUEST
- fetch()

- Browser is an application of a window
- W3C is the organisation which defines html and css specs and it is then followed by all browser service providers

- ! In html gets the html signature

- Inline and block elements

    - So, whenever a new h or p tag is created, it spans to the next line, this is example of block.
    - Img tag is inline, as it does not go to the next line on its own.
    - Div is a block element
    - Span is inline container

- Style should be inside head and outside body
- In style if we select a tag, then it is element selector
- But if we select a tag with a specific id, it is attribute selector
- If styling for class, start with .
- If styling for id, start with #
- TO create a link, use anchor tag <a href=""></a>

- Form
    - REQUIRED INSIDE INPUT TAG
    - Form header should have method as post or else it will show the data filled in the input fields in the url once we submit it
    - If method is set to get or method is not used then default is get and hence it shows the value in the url upon submission
- Table
    - Thead, tbody, th, tr, td
- Border makes separate boxes for each entity
    - So we use border-collapse: collapse; to make it unified
- Radio input type can have default but make sure to name all the radio tags with same name or else radio will turn into a multiple options
- Datalist also accepts value which is not present in the options

JS 08/08

08 August 2024 09:17 AM

```
<button onclick="greet();">Click</button>
```

-
  ```
  <script>
      function greet(){
          alert("Welcome to js!!!!!!");
      }
  </script>
  ```

- JS runs inside the browser
- NODEJs runs outside the browser
- we need not specify the type of any attribute
  - Just use var
- Function functionname(parameters){
  }

Callback:

```
function testadd(){
        let res = add(10,20, function(res){ // callback
        console.log(res)})
    }
    function add(a,b,callback){
        //alert(a+b)
        //console.log(a+b)
        callback (a+b)
    }
```

- Callback is asynchronous and return is synchronous
- Settimeout
- JS does not support access modifier
- There is promise class in js which makes a function asynch
- It uses resolve, which is basically callback
- Learn callback, promise and creating async or sync functions
- Resolve and reject are not keywords, we can write anything in that place

12-08

12 August 2024 09:08 AM

- Fetch is used to get data in AJAX in JS
- Questions on servlet
  - Http response
  - Http request
- MVC pattern
  - Model
  - View
  - Controller
- What are REST endpoints?

- TDD (Test Driven Development) and Observability
  - Junit
  - Assertions
  - @Test
  - @BeforeEach
  - @AfterEach
  - @AfterAll
  - @DisplayName
  - @Disable
  - @Tag
  - Test Suites
- Observability with Splunk
- Declarative dependency management tool in MAVEN

- Test first approach
  - We do not test business objects but the services and DAO
  - We use Junit for testing
  - Here we have assertions which are assumptions about how a method or service should function

- Assertions is a class inside junit.jupiter.api
  - Assertequals for anything from string to numbers
  - AssertSame checks if two objects are same or not
  - AssertIterableEquals for checking two collections
  - AssertArrayEquals for comparing arrays
  - AssertThrows(exception name.class,()->{ body}); for excepetions
- All test methods should be of void type
- To disable a test case, use @Disabled before the method name
- If we want to have inputs and initialization of data, we can create a static method and use @BeforeAll before the method and initialize all the data inside it
- There is @AfterAll as well which could be the releasing point for data
- There is @BeforeEach and @AfterEach where the method does not have to be static and this initializes the data before each test method is run

- Splunk :
  - Stores logs

    - Forwarder helps us in forwarding log files from one system to another

2240872807

14-08

14 August 2024 09:27 AM

- Log4j2, resources, Log4j2.properties
  - Root log
- Module-info.java

- Spring framework, Dependency Injection and Application Context then Spring Boot

- Switch Cases:
  - Case 1,2,3 ->"something";
  - Default->"something";

Switch expresstion - https://docs.oracle.com/en/java/javase/17/language/switch-expressions.html

Pattern Matching with instanceof Opertor - https://docs.oracle.com/en/java/javase/17/language/pattern-matching.html

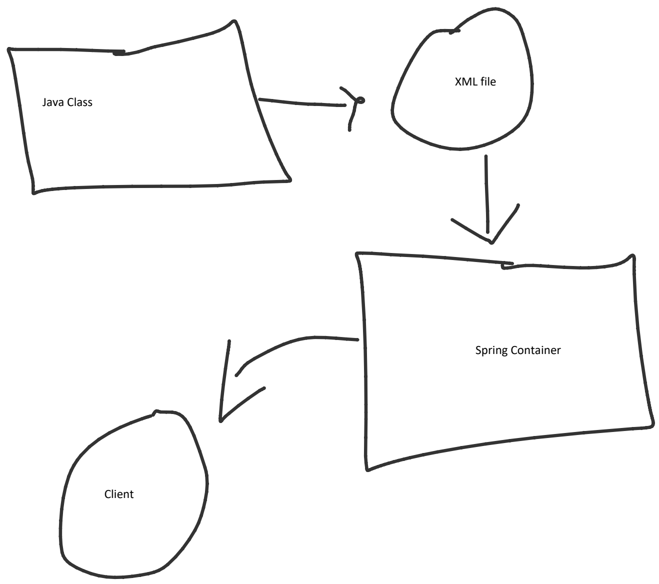
Virtual Thread - https://docs.oracle.com/en/java/javase/21/core/virtual-threads.html#GUID-15BDB995-028A-45A7-B6E2-9BA15C2E0501

- Virtual thread:
  - Thread t = Thread.ofVirtual().start()(new ClassName());
    t.join();

- GO lang

- **Spring Framework - it is a light weight container framework**
- **What is a framework?**
  - **A framework is like a structure that provides a base for the application development process.**

- It helps us in avoiding boilerplate rode i.e., repetitive code.
- Most of the things in Spring are declarative in nature.
- Java+SuperPower -> Spring..... Spring+SuperPower-> Springboot
- So, in spring, we do not have to write a lot of things from scratch like in case of Java
  - Example, to connect to jdbc, we do not have to write code for connecting in spring, it does it on its own
- IOC IN Spring



- Thought-> Object creation is complex
- IOC and DI in spring
  - Inversion of Control and Dependency Injection with Spring
- **IOC**
  - **C - control - it basically talks about create and destroy**
    - **Create - new keyword**
    - **Destroy - make it null**
  - **Spring says that the above is wrong. Control should be on lifecycle and not on an object**
  - **So, new keyword is like a crime in spring**
  - **The solution to above problem is I - Inverse**
    - **In inverse, we do not use the new keywrod**
    - **Instead we delegate the work to other classes**
    - **So, all the services are declared to be part of a container**
    - **Then we can use container class to create or invoke methods of other classes without using the new keyword explicitly**
      - **Though new is still used but not directly by us**
  - **Containers' job is to manage the dependencies and not to take care of the business logic. It wires the services or classes or methods.**
  - **This in hand reduces the tight coupling**
- **A class to be available inside a container it has to be a bean**
  - **To do so we should declare the class inside an xml file and then let the container know that it should read or load the xml file**
- **A container is an object in the main class**

```
ClassPathXmlApplicationContext container =
        new ClassPathXmlApplicationContext("springconfig.xml");
GreetingService gs = container.getBean("greetingSerivce",GreetingService.class);
```

- If we try to print gs, it then prints a reference of gs.
- To make it print value of some function, sout(gs.methodname());
- Now if we create multiple GreetingService instance, it won't create new objects
  - By default scope of bean is singleton
  - To create new instances

        - We should adding scope as prototype in xml
- Every bean or service may have to acquire or release resources (API Calls, Connections etc)
    - To acquire we add init-method="setup" in xml and setup method should be there in the service class as well.
    - To release we need to add destroy-method="cleanup" in xml and cleanup method in the service class.
- Now if we check, the setup is called by container on its own but cleanup is not called.
    - If scope is prototype then setup is called as many times as we have created object of bean
    - To call cleanup we should write this in mainapp
        - container.registerShutDownHook();
        - But then scope in xml should be singleton
- We should never wire a business object but if we still want to do so then in the xml file:

```
<bean id="customer"
     class="com.hsbc.firstspringapp.model.Customer"
>
<property name="name" value="John"/>
   <property name="cell" value="235346534654"/>
</bean>
```

- Container creation is **eager loading** as the moment we create a container, it loads all the beans or methods defined in the xml file
- If we want container to be lazy, then inside the bean for a method in xml, we should add a property named lazy-init=true

- <constructor-arg name="name" value="John"/>

16-08

16 August 2024 09:20 AM

POJO is plain old java object

```
<bean id="emailConfig"
      class="com.hsbc.firstspringapp.service.EmailConfig">
    <constructor-arg index="0" value="192.168.12.2"/>
    <constructor-arg index="1" value="John"/>
    <constructor-arg index="2" value="P@ssword"/>

    <!-- collaborators and configuration for this bean go here -->
  </bean>
```

Index helps in making sure the correct attribute is getting the data

```
<bean id="emailService"
      class="com.hsbc.firstspringapp.service.EmailService">
    <constructor-arg ref="emailConfig"/>
    <!-- collaborators and configuration for this bean go here -->
  </bean>
```

Here we are passing ref in constructor-arg

- There can be two beans of the same class but ID should be different
    - Or inside the bean we can declare primary="true"

- Like classpathxmlapplicationcontext
    - There is FileSystemXMLApplication
    - The only difference is that in case of file we should pass the complete path
    - In classpath we only pass the reference

- We can have multiple container xml files and we can use two ways to call them
    - Either pass multiple arguments in the ClassapathXMLApplication
    - Or make multiple containers and in the main container, use
        - <import resource="reference of another xml/>

- Annotation Configuration
    - We do not need a xml file for container instead
        - We can have a java class and add @Configuration before the class declaration
        - This tells spring that the particular class is a container

```
@Configuration
public class SpringConfig {
  @Bean
```

```
    //@Bean("someIDname")
    @Scope("singleton")
    public GreetingService greetingService(){
      return new GreetingService("John");
    }
}
```

- Inside the main, below is used to create container
  AnnotationConfigApplicationContext container =
  new AnnotationConfigApplicationContext(SpringConfig.class);

- Instead of declaring @Bean in config file, we can go to the business class or service class
  - Before the class definition we can add annotation
    - @Service("optional to add a name or id")
    - Then inside the config file we should add
      - ComponentScan(basePackages={"package reference like com.hsbc.something"})
- Use @Service for services and given that we have a source code, for DAO use @Respository and otherwise @Component
- In case of wiring and if we are using @service or repo or compo
  - Inside the class we should mention @Autowired
- Init-method in java is @PostConstruct and destruct method is @PreDestruct
  - To use the above two annotations we should add javaxannotation dependency

- AOP - Aspect Oriented Programming
  - It has specific concerns and cross-cutting concerns
  - Here java proxy files are used indirectly

19-08

19 August 2024 09:08 AM

- Generally we apply log to a target object
- Applying log before or after is called advice
- Advice is a cross cutting concern -> before and after (CHECK MORE ADVICES)ђ
    - There is around which does before and after
    - In case of around we should pass argument ProceedingJoinPoint pj in the aspect class method
    - There is AfterReturning
    - AfterThrowing

AOP:

- CHECK SPRINGAOP PROJECT
- We need to use @EnableAspectJAutoProxy in springconfig file
- There is join point and point cut in spring
    - We can use aspect to directly make spring execute logaspect without changing the code in orderservice or any other service

```
@Component
@Aspect
public class LogAspect {
  @Before(("execution(* com.hsbc.springaop.service.OrderService.placeOrder(..))"))

  public void logBefore(){
    System.out.println("Before.."+ LocalDateTime.now());
  }
}
```

@Before(("execution(*com.hsbc.springaop.service.*.*(..))")) for all classes and methods

- The above expression is called point cut
- And wherever the expression matches a class or method, that point is called join point

Aspect helps in applying concerns to class files also

```
  @Around("execution(* com.hsbc.springaop.service.*.*(..))")
  public void logBefore(ProceedingJoinPoint pj) throws Throwable{

    System.out.println("Before.."+ LocalDateTime.now());
    System.out.println(pj.getSignature());
    pj.proceed(); // this makes the class execute
    System.out.println("After.."+LocalDateTime.now());


  }
```

- ProceedingJoinPoint has multiple methods

  - .proceed(); -> makes the class execute
  - .getSignature() ->returns the name of the class which will execute or is executing

- SPRING JDBC
  - In spring, we
  - need jdbctemplate or jdbcclient and it executes all CRUD operations
  - We need DriverManagerDataSource in the SpringConfig
  - We can't use @Service in this case as we only have access to the class file
  - REFER TO SPRINGJDBC

WHAT IS IMPEDENCE MISMATCH

- BeanPropert
```
class ContactRowMapper implements RowMapper<Contact>{
   @Override
   public Contact mapRow(ResultSet rs, int rowNum) throws SQLException {
         return  new Contact(rs.getInt("id"),
               rs.getString("name"),
               rs.getString("email"));
   }
}
```
- RowMapper

SPRING BOOT:

- It is RAD - rapid application development
- Spring+ Starter POM(RAD) + Auto Configuration + Version Compatibility = Spring Boot
- Every springboot application has annotation @SpringBootApplication before the class name
- CommandLineRunner is a mini main method
  - We use this because we cannot access non-static methods directly from main method
- As in case of springboot we do not have a springconfig file
  - Every springboot has a resources folder and inside that there is applicaton.properties
  - That is where we declare the url username and password for mysql
  - spring.datasource.url=jdbc:mysql//localhost/hsbcdb

20-08

20 August 2024 09:18 AM

AMEYA JOSHI

- The moment we declare @Controller
  - The method gets servlet
  - This is also known as front controller pattern
  - So it gets all the request initially and then it decides where that request will go
    - For that to happen, servlet should be aware of all the methods
  - Spring makes it easier by just asking to send request URI and then dispatcher
  - searches for that controller with value present in the request URI
  - URI - Uniform Resource Identifier
- REST - Representational State Transfer
- HTTP request is a protocol and HTTP servlet is java specific
- Every annotation is an interface
- @RestController is equivalent to using @Controller @Request and @Response
- So URI is done by using @RequestMapping
  - Inside requestmapping we declare the value which is the name and method like request get put post delete etc
  - Get is the default method for requestmapping
- Now if we are declaring the requestmethod inside requestmapping
  - Instead we get the option of using the following
    - @PostMapping
    - @GetMapping
    - @Put….
    - @Delete…
- CHECK PARAMS IN REQUESTMAPPING

- REST:
  - It is architectural pattern
  - It uses http protocols
    - There are methods like GET POST PUT DELETE etc
    - HTTP URI syntax like paths parameters etc
    - Media types like xml json html plaintext etc
  - There are 6 characteristics that a service should follow to become a RESTful service
  - Only response can have a status and not a request

- Request Body
  - @RequestBody will bind the parameter in a method to the request body
  - @RequestMapping(PATH)
    Public void writeString(@RequestBody String inputvariable){}
    Or it can be @RequestBody SomeObject for classes
  - If we are using RequestController, we need not use the RequestBody as it is already present in controller
- Request has options of .getHeader
- Response has options of .setHeader and .setStatus to set the status or header
  - Or we can use @ResponseStatus(HttpStatus.CREATED) // there are many like CREATED

```
spring.application.name=courseapijdbc

server.port = 9001

spring.datasource.url =jdbc:mysql://localhost:3306/hsbcdb
spring.datasource.username = root
spring.datasource.password = hksharsh11
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.sql.init.mode=always
logging.level.web = trace
```

- REMEMBER THAT once the application is run for the first time, we should comment the init.mode=always as next time it will try to create the already existing table

- Hibernate ORM JPA
  - Object Relational Mapping
  - Hibernate is an ORM solution for java
- JPA is Java Persistance API
  - Is a specification of java
  - Used to persist data b/w java object and relational database
  - JPA Annotations
    - @Entity
    - @Table - when we specify the name
    - @Id - used to map to primary key
    - @Column -
    - @GeneratedValue
    - @GenericGenerator
    - Types of generator:
      - Assigned
      - Auto
      - Increment
      - Sequence
      - Identity
      - Hilo
- Boot provides something named datarepositry
- Rules of datarepositry with example of getByFirstName
  - Method should preferably start with get or find
  - It then looks for next capital letter
  - By will come, so it knows it has to get by
  - Then next capital letter
  - F-first, but no column named First so keep looking further
  - Then it understands it is firstname
- CrudRepositry

- Inter-service communication
  - RestTemplate
    - How to declare using @RestTemplate in the main or the springapplication file
    - It has multiple methods
    - Exchange is one of them which helps in doing both request and response
  - RestClient
    - Get

        - Put
        - Retrieve
        - ToEntity
        - toBodilessEntity
    - Both the above are synchronous and hence blocking in nature