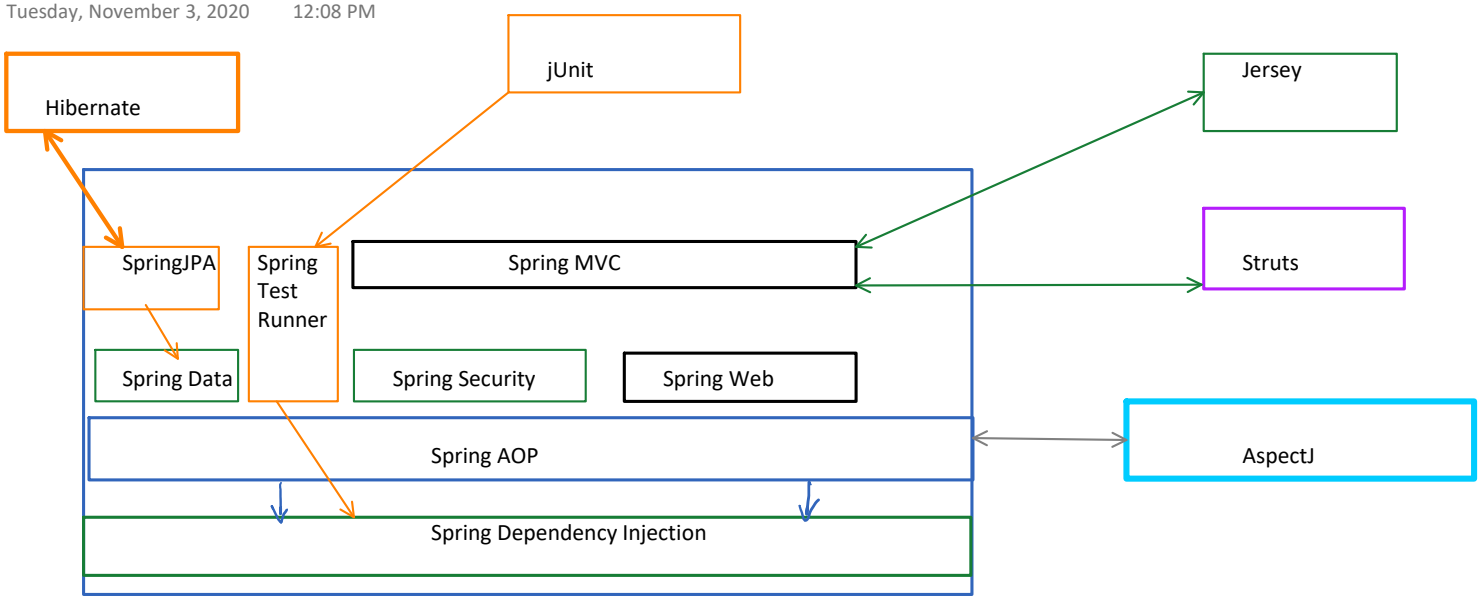
Spring Framework

Tuesday, November 3, 2020 12:08 PM



What Makes Spring Great

- It is a collection of large independently usable Project
  - Most of them depdends of Spring Core
    - Dependency Injection
    - Spring AOP
- Different Projects can provide different functionalities
  - Spring MVC competes with Struts
  - Spring Data makes working JDBC better

- Spring doesn't aim at killing other Projects
  - Its goal is not to replace Junit, or hibernate or AspectJ
  - Its goal is to make your life easier while you use any framework
- You can use hibernate without Spring
  - Hibernate works better when you use Spring
- You can use jUNit without Spring
  - Spring improves your experience with jUnit
- You want to use Struts instead of Spring MVC?
  - No Problem
  - You can still use Spring Dependency Injection, AOP and Spring Data while using struts
- Aspect J is much more versatile framework compared to Spring AOP
  - Spring AOP doesn't compete with AspectJ
  - It provides a simpler way to implement only 50% features
  - Those 50% features are used 95% time

AOP (Aspect Oriented Programming)

Thursday, November 5, 2020 3:26 PM

What is an Aspect?

- An Aspect is one concern or one functionality of our program
- Often we have multiple concerns in our program

Problem

- Very often when you solve a particular task, you often do additional side works
- These side works are important to complete my taks
    - But they are not core activity for my task

Example 1

- In bank account withdraw,
    - the main concern is
        - Do we have sufficient balance
        - Deduct the amount if balance is sufficient
    - Side concerns
        - Is user authenticated?
        - Is amount value given correct?
        - Opening closing database

Example 2

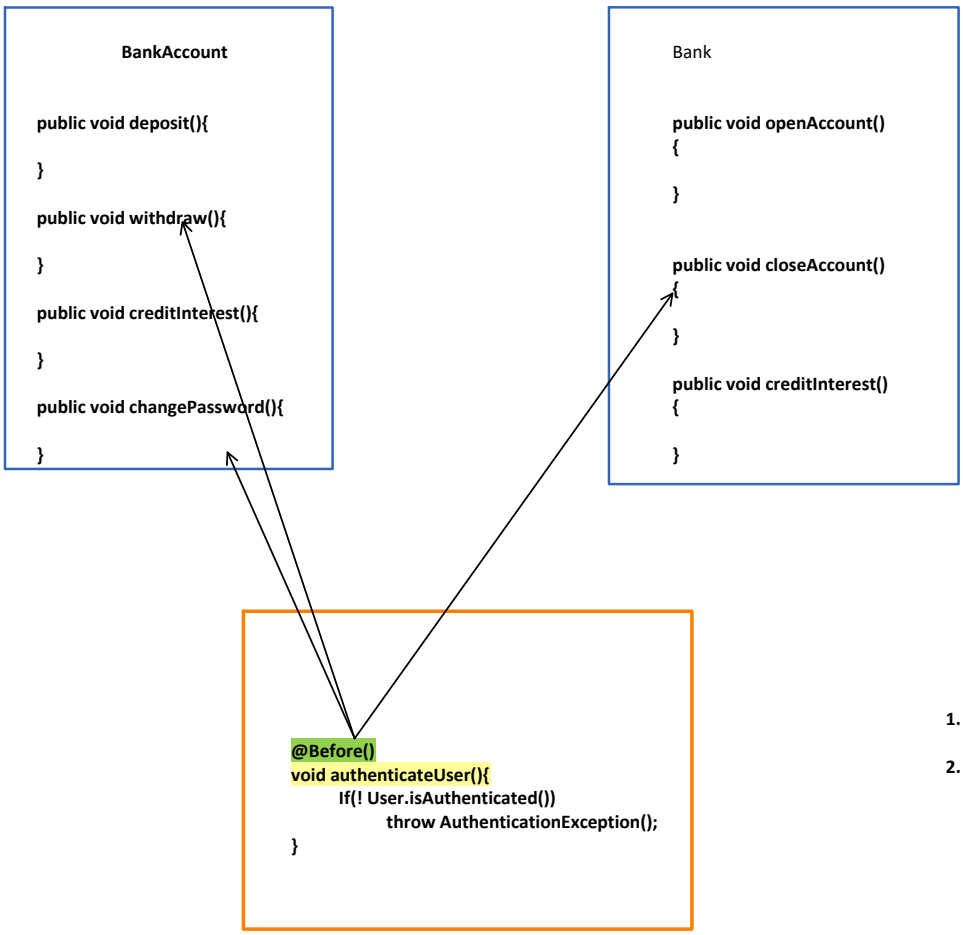
- In Jdbc Operations
    - The Main Concern
        - Fire insert/update/delete queries
        - Get the data from the ResultSet
    - Side Concerns
        - Open Connection
        - Close Connection
        - Handle Exception

Problem Analysis

1. Code Tangling
    - Unrelated codes are tangled (mixed) together
        - Business logic
        - Data access logic
        - Exception handling logic
2. Code Scattering
    - The Side concerns are repeated in every code block
        - Exception hanlding
        - Connection open close
        - Transaction — commit — rollback
        - Logging
    - This leads to redundant programming

Aspect Oriented Programming Components

- Separate Different Aspects (Side Concerns) as a separate functionality
- The point in core functionality where this aspect is required can be called **JoinPoint**
- An aspect may be applied on multiple JoinPoint. A set of all those JoinPoint is called PointCuts
- You apply a functionality on the point cut. This Functionality is called **Advise**



1. **Advise** — What do you want to do
2. **PointCut** — where you want to apply the advise
   a. Before certain selected methods
      - Each method is a JoinPoint

ASPECT

Spring JPA

Thursday, November 5, 2020 5:31 PM

Evolution of Database Access

Generation 1 — Plain JDBC Calls

- User Makes JDBC calls using Jdbc and Drivers
  - Connection
  - Statement
  - ResultSet
- Developer need to generate queries based on User defined Objects
- Developer need to get the value from Resultset and created Object from those values
  - You need to match column to field
  - You need to convert data type
  - You need to plan for Different type of Object

```
if(account instanceof OverDraftAccount) {
    OverDraftAccount od=(OverDraftAccount) account;
    odLimit=od.getOdLimit();
}

final String qry=String.format("insert into bankaccounts(account_type,name,password,balance,odLimit) "
                    + "values('%s','%s','%s',%f,%f)",
                      account.getClass().getName(),
                      account.getName(),
                      account.getEncryptedPassword(),
                      account.getBalance(),
                      odLimit
                     );

//return manager.execute( statement -> statement.executeUpdate(qry) );

return manager.executeUpdate(qry);
```

```
private BankAccount _createAccount(ResultSet rs) throws SQLException {
    BankAccount account=null;
    String accountType=rs.getString("account_type");
    int accountNumber=rs.getInt("account_number");
    String name=rs.getString("name");
    String password=rs.getString("password");
    double balance= rs.getDouble("balance");
    double odLimit= rs.getDouble("odLimit");

    switch(rs.getString("account_type")) {
        default:case "in.conceptarchitect.banking.core.SavingsAccount":
            account=new SavingsAccount(name,"", balance); break;
        case "in.conceptarchitect.banking.core.CurrentAccount":
            account=new CurrentAccount(name,"", balance); break;
        case "in.conceptarchitect.banking.core.OverDraftAccount":
            account=new OverDraftAccount(name,"", balance); break;
    }
    account.setAccountNumber(accountNumber);
    account.setInternalPassword(password);
    if(account instanceof OverDraftAccount)
        ((OverDraftAccount) account).setOdLimit(odLimit);
    return account;
}
```

Can These two steps be Automated?

- Can we automatically generate Query by looking into Object?
- Can we automatically create object and get the values from Result Set?

Solution

- We can do it by using Reflection
- Generating the Query
  - We can find all the fields of a class
  - We can generate a insert or update query based on the field names
  - This is assuming that the field name and the column name are exactly same
    - What if they are not same
- Fetching the records from ResultSet
  - We can find all fields of a class
  - We can search for a matching column in Resultset
  - We can get the data from the those columns.

Generation 2 — ORM and Hibernate

- ORM is a broad conception to represent Object-Relationship-Mapping
- We need a way to simplify database interaction by providing functionalities like
  - Auto generate tables
  - Auto insert/update/delete Objects into the table
  - Automatically fetch Object from the table
- But world of Object Oriented Programming is different from the world of database

Features	OO Programming	Database	Remark
Fundamental Elements	Object and Class	Table	
Data is stored as	Objects and properties	Rows and colums	
Object Identity	hashCode Not a field of the class	Primary key	

Relationship	Author has many books
	
	class Author{
	String name
	List<Book> books;
	
	}
	
	class Book{
	
	String title;
	Author author;
	}
		Tables have relationship
		
		table Authors
		ID PRIMARY KEY
		NAME VARCHAR(255)
		
		Table Books
		ID PK
		TITLE VARCHAR(255)
		AUTHORID FK
			• No ID required in Author Class
			• No way to represent List of books in BookTable
			○ Represented by foregin key in Book table
			
			• No primary key or foreign key required in java class

Class Hierarcy	• Inhertiance • Interface	Not supported	

- More features
  - Performance Optimization
  - Automatic change detection
  - Caching of records already pulled from the database

Hibernate

- Hiberante is one of the most popular ORM framework available for Java Applications
- It is developed as an open source by third party company — not java not spring
- Provides extensive support for Object Oriented interaction with database
  - You can Map Java class to Database
  - Automatically generates Query
  - Provides a special Object Oriented Query **HQL (Hiberanate Query Language)**
    - Provides query syntax based on Java classes and other features
  - Support for 1:1, 1:M, M:M relationship
  - Support for mapping inheritance
  - Support for different databases
  - Facility to cache the already pulled records
  - Automatic change detection in object
- As a developer you don't execute insert query
  - You save an object
- You query for objects and get a list of Object

Other ORMs

- There are several other lesser popular ORMs available

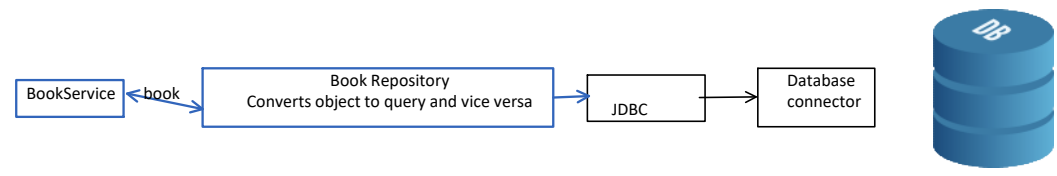
Generation 3 — JPA

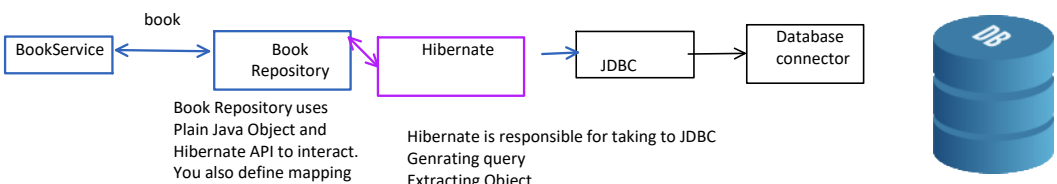
- Java created a standard abstraction on the top of ORM frameworks
- Java standard is known as **Java Persistence API (JPA)**
- You can think of JPA as an standardization of ORM.
- Inspired greatly by Hibhernate
- Provides its own query syntax call **JPQL —> Java Persistence Query Language!**
- Provides standard set of @Annotations and interface to describe how you should interact with any JPA framework
- Hibernate supports JPA
- Hibernate can be viewed as JPA+ framework
  - It supports everything JPA + Additional features

Generation 4 — Spring JPA

- It is an additional layer on the the top of JPA.
- It tries to make the JPA conifuguration and use lot simpler by defining its own set of classes and auto generators
- You often need to write no code to get the data in and out of database

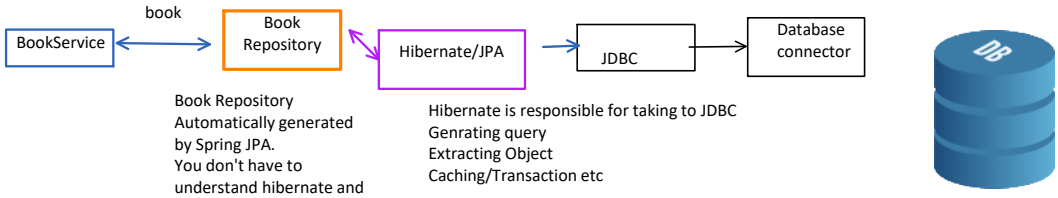
- Note
  - Spring JPA doesn't aim at replacing hibernate
  - In fact it will use along with hibernate
  - Actual JPA interaction and heavy weight lifting is done by Hibernate or other JPA providers
  - Spring simply provides its own functionality to consume JPA and provide easy access to them





and configuration

Extracting Object
Caching/Transaction etc



Book Repository
Automatically generated by Spring JPA.
You don't have to understand hibernate and very little JPA to make the call. You just define the interface suggesting what data you need. Spring will automatically implement the required logic.

Hibernate is responsible for taking to JDBC
Genrating query
Extracting Object
Caching/Transaction etc

Spring JPA Requirement

```
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-jpa</artifactId>
    <version>2.3.5.RELEASE</version>
</dependency>
```

Assignment 14

Thursday, November 5, 2020 5:43 PM

- Create a Library to
    1. Automatically generate insert/update code based on a object
    2. Automatically create an Object getting the value from a ResultSet

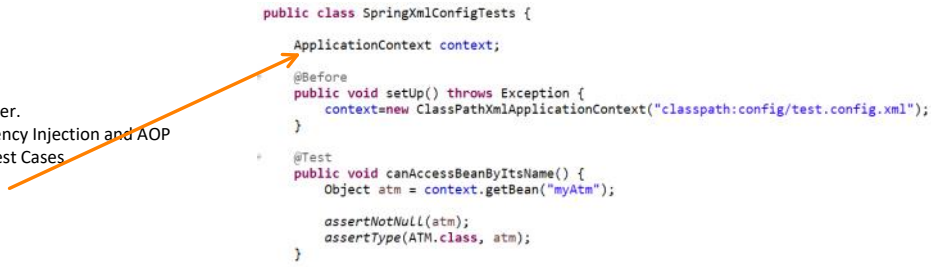
Spring Unit Testing

Friday, November 6, 2020 9:43 AM

- jUnit is the most popular Unit Testing Framework in Java Echo System.
- It is flexible and scalable via third party
- It's main components are
  - Test Runner
  - Test Reporter
  - Test Assertions
- It is possible in jUnit to extend or replace any part of it
  - We created our own Assertions.

Springify jUnit

- Spring doesn't aim at replacing jUnit, it aims at making it better.
- Even Unit Testing can leverage the benefit of Spring Dependency Injection and AOP
- We can use Spring to dependency inject component in our Test Cases
- We can create the context in Spring Test and Use it



```
public class SpringXmlConfigTests {

    ApplicationContext context;

    @Before
    public void setUp() throws Exception {
        context=new ClassPathXmlApplicationContext("classpath:config/test.config.xml");
    }

    @Test
    public void canAccessBeanByItsName() {
        Object atm = context.getBean("myAtm");

        assertNotNull(atm);
        assertType(ATM.class, atm);
    }
```

Spring Support for jUnit

- Spring Provides its own TestRunner for jUnit
- The advantage is the SpringTestRunner automatically injects dependency to a jUnit Tests using standard @AutoWired
- You don't need to create the contexts

```
@TestRunner( SpringTestRunner.class)
@Component
@Confiugration( MyTestConfig.class)
class MySpringifiedAccountRepositoryTest{

    @Autowrited AccountRepository repository;
}
```

Important

- Test Support is not present in the core spring package you need to import another Spring Project

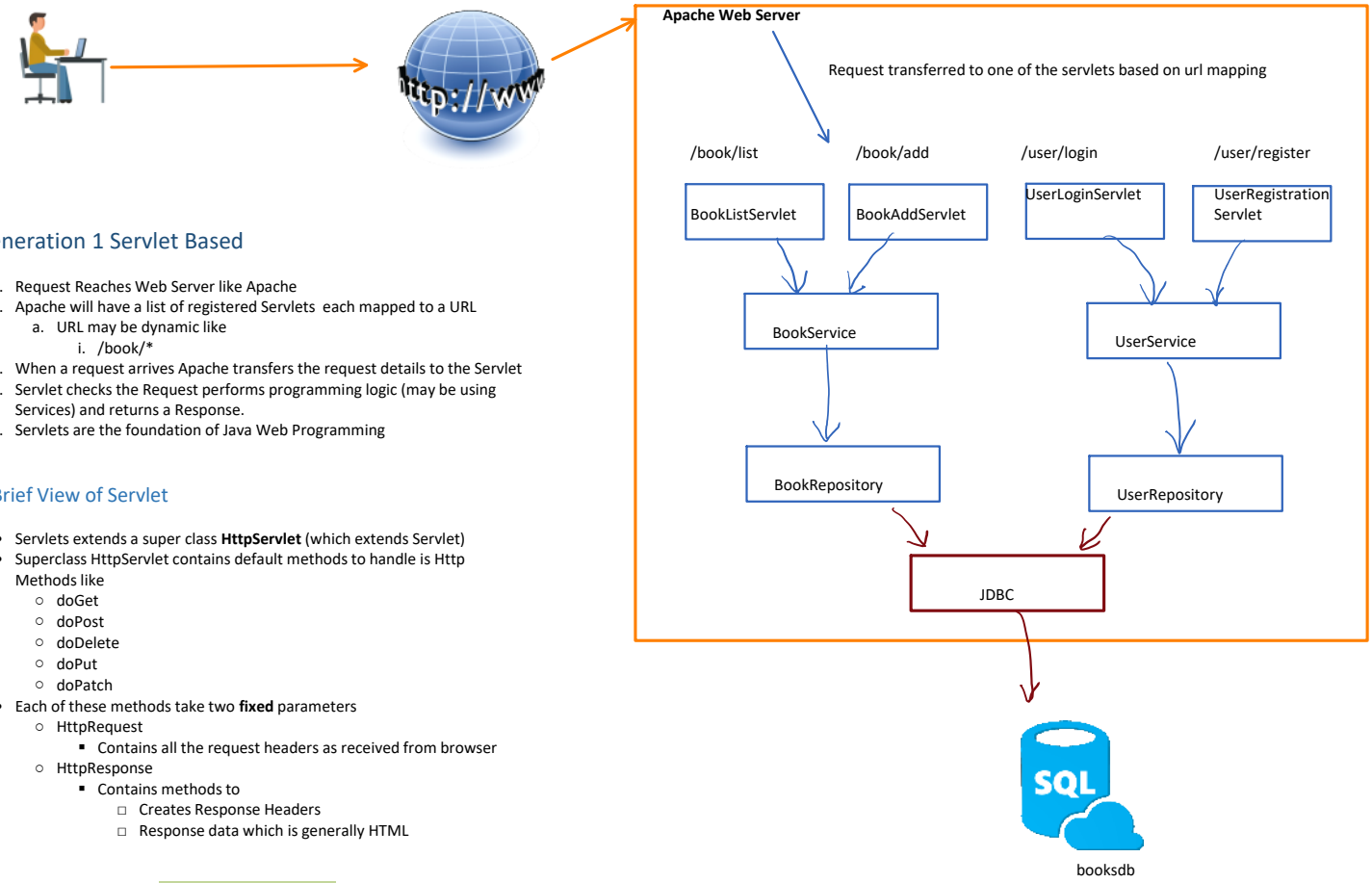
```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
    <version>${spring.version}</version>
    <scope>test</scope>
</dependency>
```

Web Application

Friday, November 6, 2020 10:19 AM

A web server doesn't know how to interact with a database

http://booksweb.org/books/by/jeffrey-archer



Generation 1 Servlet Based

1. Request Reaches Web Server like Apache
2. Apache will have a list of registered Servlets each mapped to a URL
    a. URL may be dynamic like
        i. /book/*
3. When a request arrives Apache transfers the request details to the Servlet
4. Servlet checks the Request performs programming logic (may be using Services) and returns a Response.
5. Servlets are the foundation of Java Web Programming

A Brief View of Servlet

- Servlets extends a super class **HttpServlet** (which extends Servlet)
- Superclass HttpServlet contains default methods to handle is Http Methods like
    - doGet
    - doPost
    - doDelete
    - doPut
    - doPatch
- Each of these methods take two **fixed** parameters
    - HttpRequest
        - Contains all the request headers as received from browser
    - HttpResponse
        - Contains methods to
            - Creates Response Headers
            - Response data which is generally HTML

```
class BookByServlet extends HttpServlet{

    @Override
    public void doGet(HttpRequest request, HttpResponse
    resp){

        //1. get user request
        String author= req.getQueryString("author")

        //2. get data from the server

        List<Book>
        books=bookService.getAllBooksBy(author);

        //3. create html for the browser
        String html= htmlGenerator.buildHtmlFor(books)

        //4. send response to the browser
        resp.getWriter().write(html);
    }

}
```

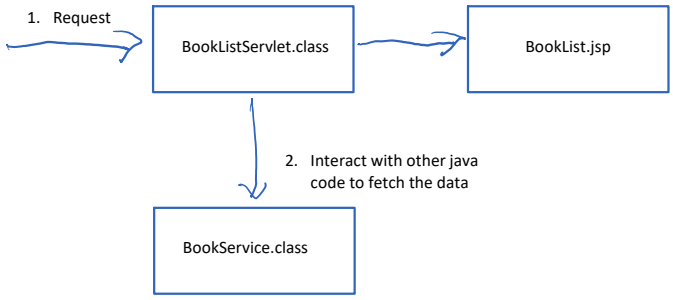
Problems

1. We need One Servlet Per URL
    - A servlet can serve wild card url
    - But generally we design different servlet for different funcionality
        - Web don't want a single BookServlet doing everything about book
        - It needs to check url, do a switch case
        - It will increase the code
2. We need to generate HTML programmatically using String manipulation that will become complex for large UI
3. There is no support for dependency injection of services and repository here.

Generation 2 — JSP and Servlet

- Java introduced JSP or Java Server Pages
- They are like HTML pages where you can embed Java code or data from a Java Program
- Internally a JSP will be translated and compiled as a Servlet
- This approach solves One Major Problem —>
    - we don't need to write HTML inside Java
- But we can't replace Servlet with JSP
    - They we need to write java logic in html like page
- So We use a combination mechanism

3. Servlet will dispatch request with data to JSP



4. JSP will create the HTML output based on data it received from the server

```
class BookByServlet extends HttpServlet{

    @Override
    public void doGet(HttpRequest request, HttpResponse resp){

        //1. get user request
        String author= req.getQueryString("author")

        //2. get data from the server

        List<Book> books=bookService.getAllBooksBy(author);

        //3. create html for the browser
        req.addAttribute("books", books);
        req.getDispatcher("booklist.jsp").dispatch(req,resp)
    }

}
```

Problems

We still have our previous unsolved problems

1. No dependency injection
2. Too many Servlets created one for each functionality

Generation 3 — Spring MVC (Controller Based)

- Spring Introduces its own framework to deal with web programming
- **Spring provides its own Servlet that would catch all the URLs for a web application**
  - **Every URL will reach Spring's pre-created Servlet called DispatcherServlet**
- **We will not create a single servlet in our project**
- When any request is made, the web server will transfer the request to SpringServlet
- Once Spring Receives the request it
  1. Checks the URL and HTTP Method
  2. Based on URL and HTTP Methods, it transfers the control to one of the user defined functions present in a controller

Controller

- A Controller is a plain Java class with @Controller annotation
  - @Control is also an stereotype for @Component
- This class doesn't extend any other class
- It can have different user defined methods to handle different URLS
- User defined methods can take parameters based on their needs
  - It may be something from
    - Http request
    - HTTP header
    - Spring context
- Controllers returns are processed by Spring Dispatcher Servlet to produce the final response

Controller vs Servlet code

Functionality	Servlet	Controller

Get All Books	`class BookListServlet extends HttpServlet{`
	
	`public doGet(HttpRequest req, HttpResponse resp){`
	
	`}`
	`}`
		`@Controller`
		`public class BookController{`
		
		`@GetMapping("/book/list")`
		`public String getAllBooks(){`
		`return "list.jsp";`
		`}`
		
		`@GetMapping("/books/by/{id}")`
		`public String getBookById(String id){`
		`return "details.jsp"`
		`}`

Get Book By Id	`class BookByIdServlet extends HttpServlet{`
	
	`public doGet(HttpRequest req, HttpResponse resp){`
	`String id=req.getUrl().substring(req.getUrl.lastIndexOf("/")+1);`
	`…`
	`}`
	`}`	

AddBook	`class BookAddServlet extends HttpServlet{`	`@GetMapping("/books/add")`

```
    …
  }
}
```

AddBook

```
class BookAddServlet extends HttpServlet{

  public doGet(HttpRequest req, HttpResponse resp){
    //Show Book Create Form
    …
  }
  public doPost(HttpRequest req, HttpResponse resp){
    //Handle Form submission and database insert
    …
  }

}
```

```
        return "details.jsp"
    }


    @GetMapping("/books/add")
    public String showBookForm(){
        return "bookform.jsp";
    }

    @PostMapping("/book/add")
    public String addBook(Book book){
        //add the book
        return "list.jsp";
    }

}
```

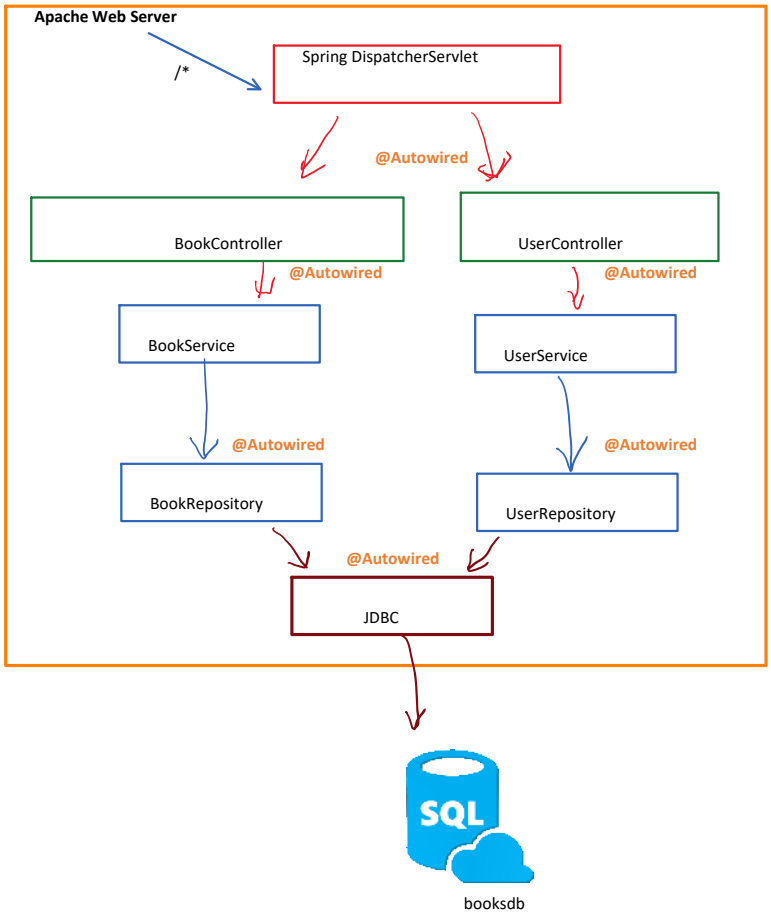
Controller Advantage over Servlets

- Compared to Servlets, Controllers are simple classes
  - They don't need to extends any superclass
  - They are generally not required to directly interact with request and response objects

- A controller can process multiple functionalonlity or URL in different user defined functions
  - This functions can have more meanigful names
- A controller can use all the Spring Framework features like
  - Dependency Injection
  - JPA support
  - AOP support
- A controller is based on simple functions which allows
  - Fewer classes
    - Instead of creating a class for each functionality you create multiple functions in same class
  - It is easier to unit test
    - It takes simple parameters and returns simple parameters
  - No need to manually process the request and response objects
-

Summary!

- A controller provides a much shorter, simpler and Springfield web programming which is more testable and scalable
- It allows you to leverage other features of spring
  - DI
  - AOP
  - JPA
  - Security
  - Testing



Ntier Application Architecture and Components

Friday, November 6, 2020 12:27 PM

There are two main forms of Interactions

1 User (Human) Interaction with system (B2C)

- Requires a human oriented output
  - Colors
  - Images
  - Tables
  - Starts
- Can be created using
  - HTML,CSS,Images etc for browser
  - Mobile Application design using Mobile development kit
    - This kit my be in any programming language like
      - Java (Android)
      - Dart (Cross Platform)
      - Objective C (Apple)
      - Swift (Apple)

2. Interaction between two computer system (business)

a. Use Cases

i. Interaction between two different business
   1) Ecommerce Website and Payment Gatways
   2) BookMyShow and PVR
   3) MakeMyTrip and AirIndia

ii. Interaction between client and server application
   1) A mobile app displaying data received from the server

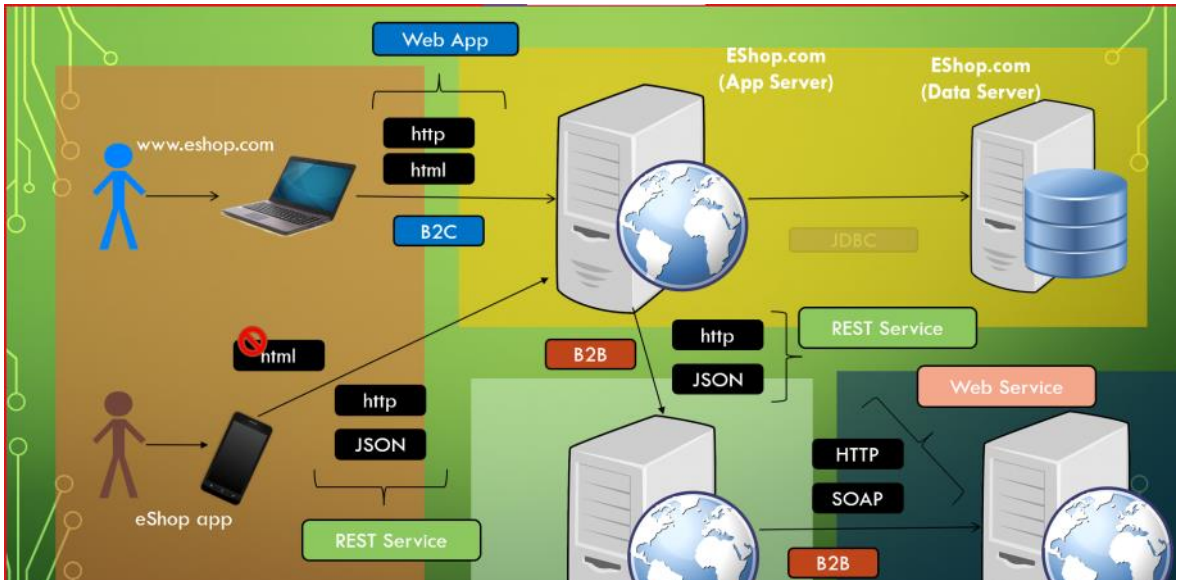
b. **Data Requirement**

- We need data that is
  1) Fast to search
  2) Small to transfer
  3) Easy to manipulate

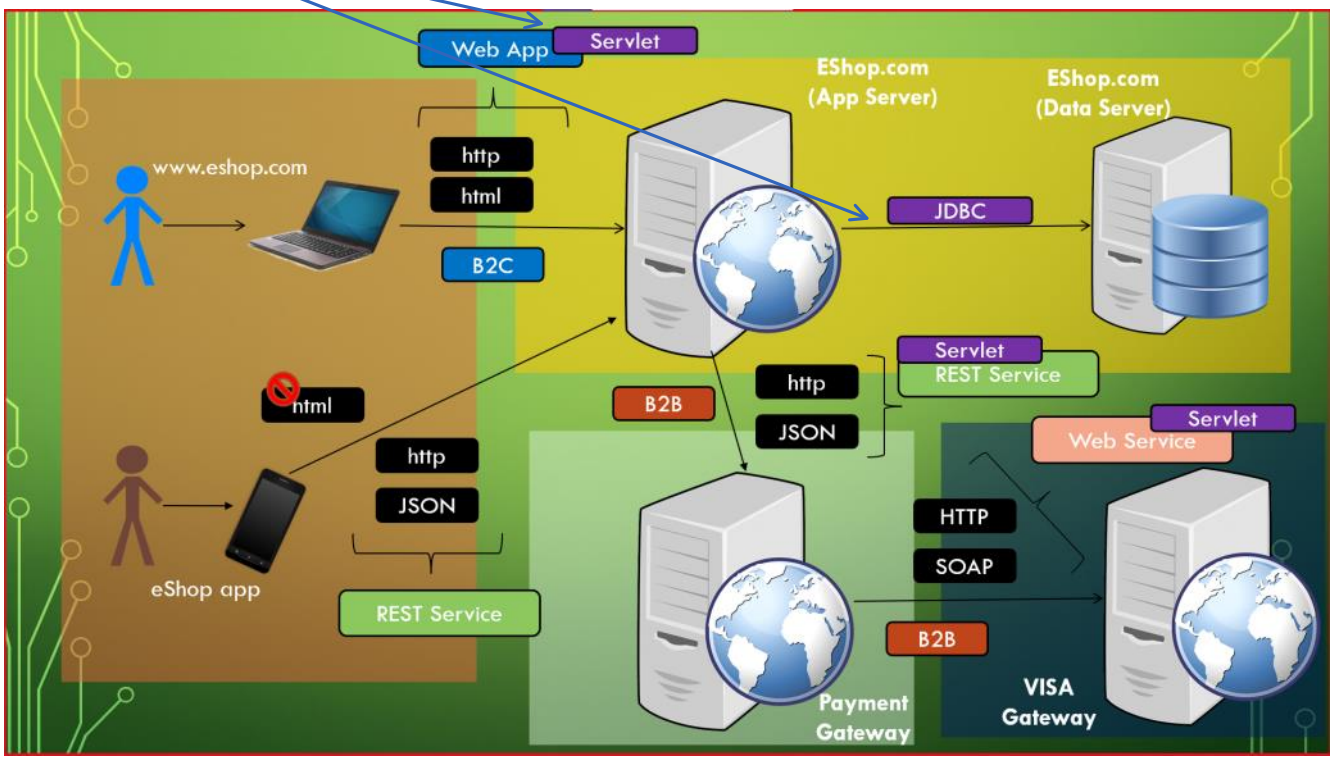
c. **Popular Format**

- **JSON —> REST Service (MORE PREFERRED TODAY)**
- **XML —> SOAP Service**

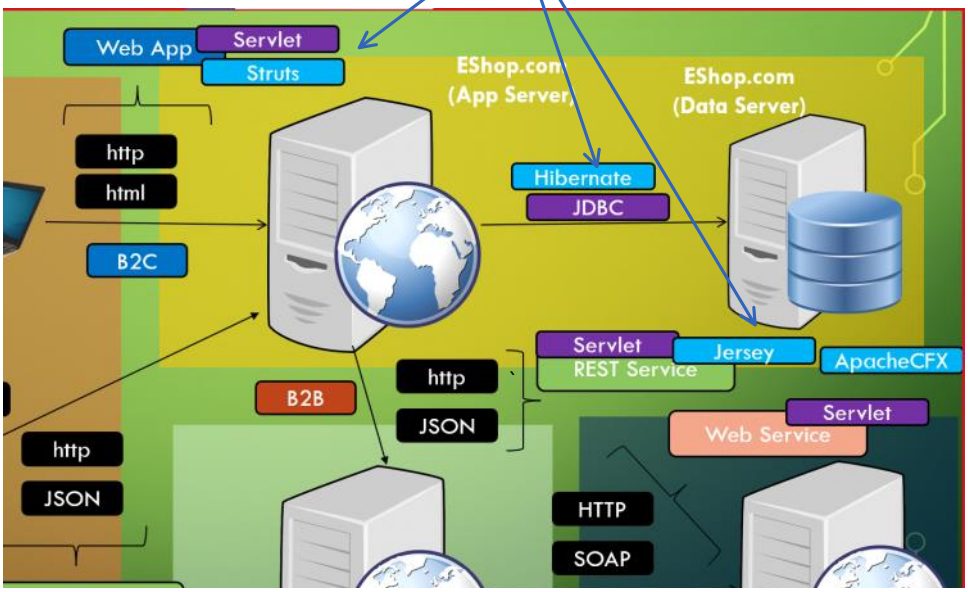
Broad Abstract Architecture

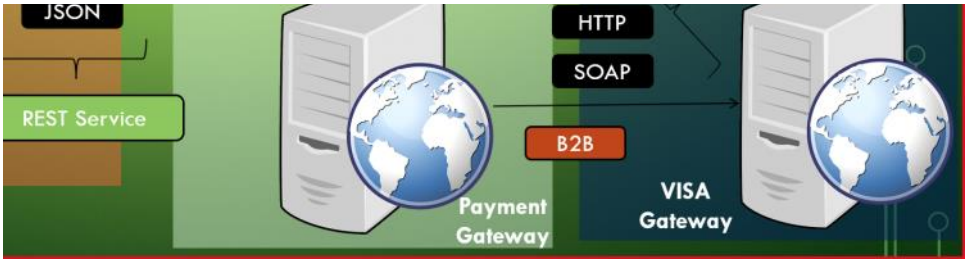


Standard Java Technology Stack (Marked in Purple Boxes)



Multi-tier Architecture with Open Source Frameworks

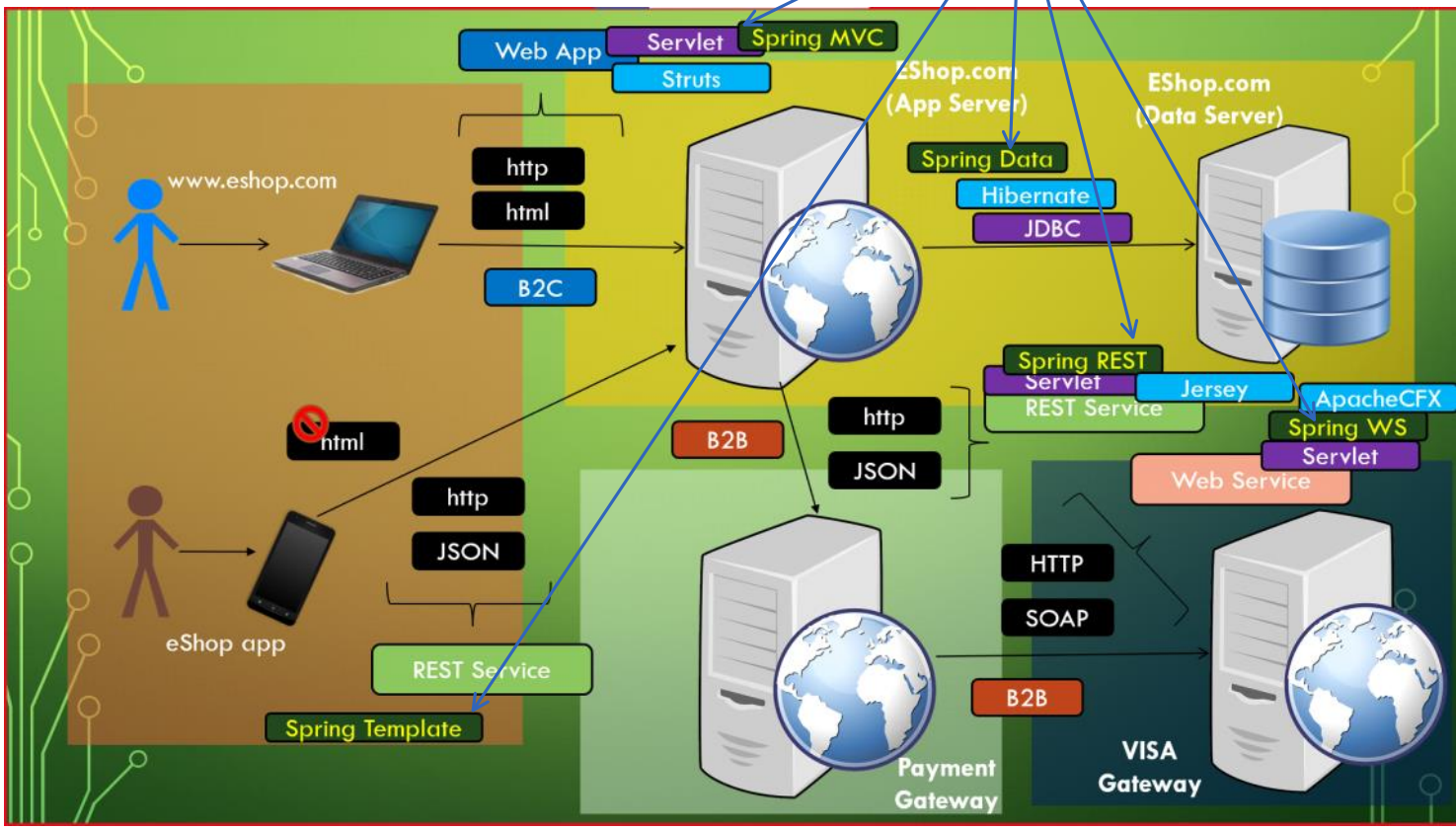




Where Spring Fits In?

Spring is everywhere

- Everywhere



Microservices

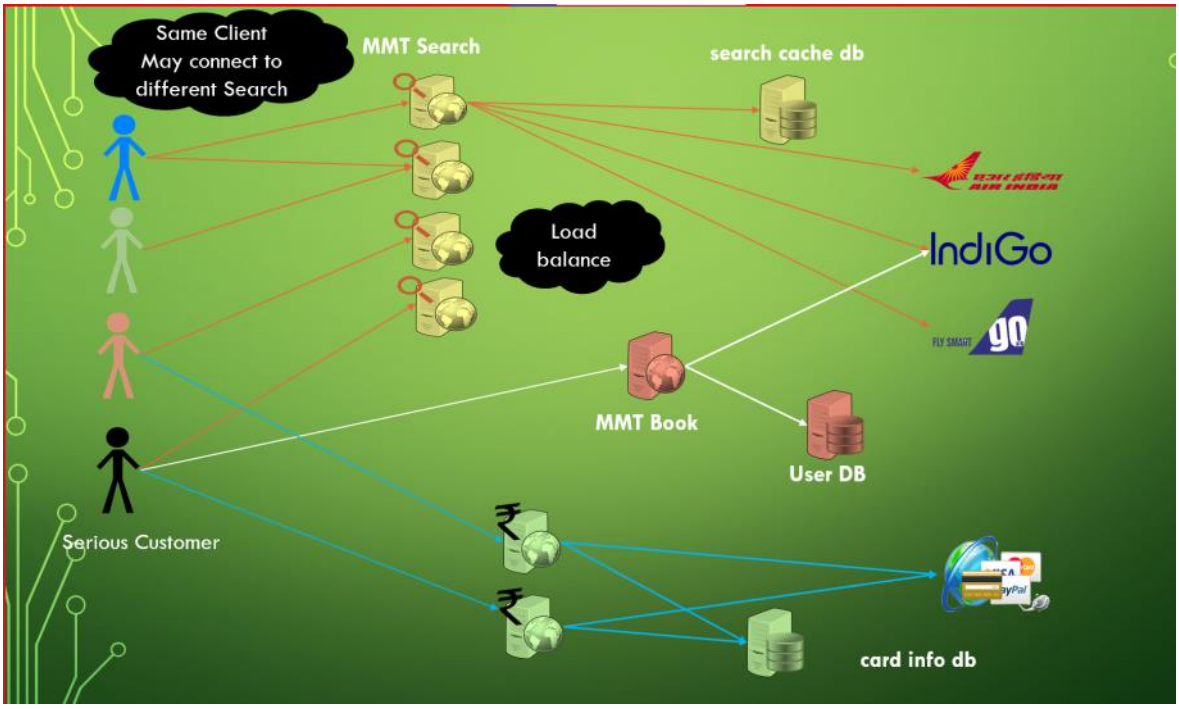
Friday, November 6, 2020 1:05 PM

1. What is a Monolythic Application?

- What?
  - In a monolythic application, Same application provides end to end needs of the client
    - Search
    - Payment
    - Book
    - Retrieval
- How?
  - Application is typically partitioned in multiple layers based on
    - Technlogy
      - Presentation
      - Business
      - Data
    - Physical divisions
      - Client
      - Server
  -
- Problem
  - Large application
  - Runs at one end
  - Difficult to scale (because of size)
  - If application fails, it fails entirely
  - Difficult to move to cloud architecture

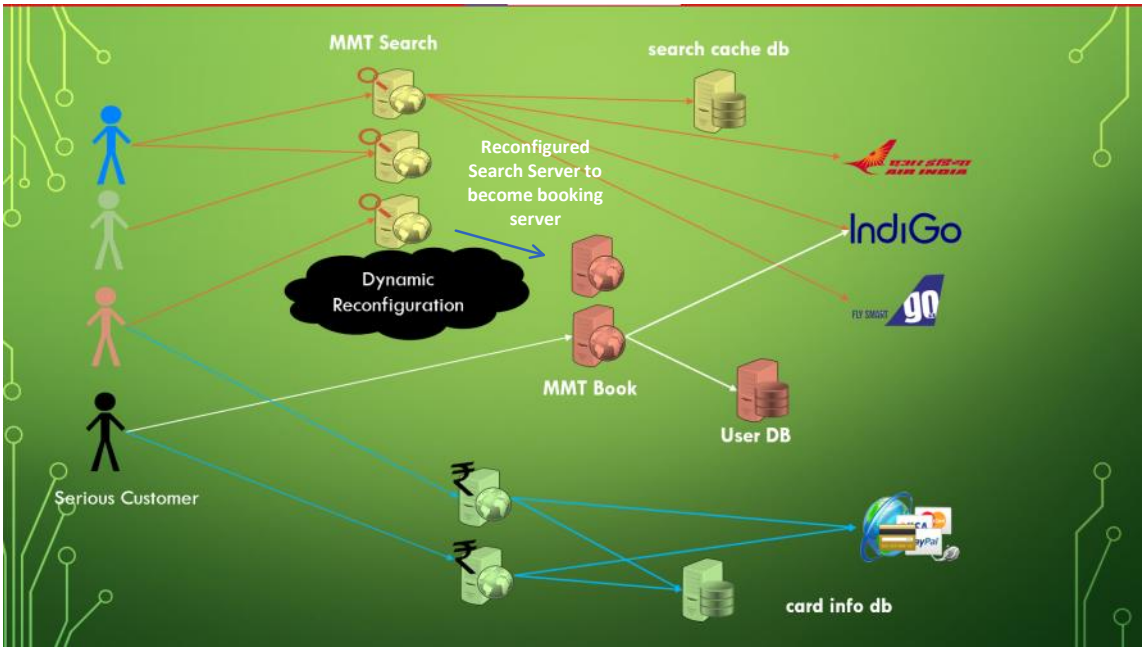
2. Microservices

- What?
  - Your large application partitioned in smaller set of functionalities
    - Search
    - Payment
    - Booking
  - Each functionality is a smaller application
    - They will have their own n-tier
      - Presentation
      - Business
      - Data
- How?
  - Each functionlity is encapsulated as separate application
  - Each application can be deployed on a different server
  - Each application can be deployed on multiple servers
  - The servers can be hosted on cloud
  - Servers can talk to each other using REST Architecture
- Advantage
  - Smaller Application
  - Same application can run on multiple servers
  - Crash of one server doesn't cause crash of other server
  - Easy to deploy and scale up.



Dynamic Configuration

- When requirement changes Server can be reconfigured Dynamically
- Consider a vacation season
  - Do you think the ratio search and book remains unchanged during vaction season?
  - How does it really change?
  - **Now we expect more people to book the ticket**
- We may
  - Rent more servers for a smaller period to scale up the booking capability
  - We may reduce a few search server and use book capabilities with same resource
    - Here we will not need to invest more



Essential Consideration for a Micro Service

Easy Deployment

- Service should be
  - light weight
  - Easy to install
  - All inclusive
    - Doesn't require manual installation of other components
- How?
  1. User containerless server architecture
     i. Instead of running your app in web server, a small web server should run within your app
  2. User application containers like docker to put your entire application need
     i. Can be easily deployed on cloud platforms like AWS, Azure, Google Cloud
  3. User Pre-configured Virtual Machines

Avoid dependency on In memory data

- Servers are going to be light weight and may crash
- Search may be served via multiple different servers
- An information stored in primary memory or one server application will not be available to other server. This includes
  - Session information from traditional web design
    - Authentications are also typically session based
- How?
  - Prefer to use a microservice to store the data in a scalable fast data bases.
  - Use http headers instead of session to store required details.

For more information

- Search for 12 Factors for Microservice
- https://12factor.net/

Challenges and Solutions

1. Service Discovery
   - If multiple instances of a Micro service is running on different servers, How will other Microservice know about its location.
   - Each may have different and dynamic address
   - Since the address of server is not fixed, how would client reach it.
2. Load Balancing
   - If multiple instances of a Micro-service is running, who would ensure that load is evenly distributed across all those microservice?
   - We must avoid that all the loads are going to only one instances and others are sitting idle
3. Session Management
   - How will we persist session data across the servers
4. Configuration Sharing
   - Each server needs some configuration of database and other such elements
   - Where should be store those configuration
5. Health and Performance monitoring
   - How do we check performance

Springified Solution

1. Netflix Eureka Microserver
   a. Created by Netflix
   b. Is a Service Registration and Discovery service
   c. Every service one starts will register itself with the Eureka Server
   d. Anyone who needs the service will contact eureka server to know all available instances of the server

2. Ribbon API
   a. Will perform a client side routing between available Microservices that it can discover from eureka
   b. Suppose Service1 needs to call Service2
      i. First time you call service1 it calls first instance of service1
      ii. Next time service1 will call seconds instance of service2 and so on.
3. Redis Server
   a. Fast NOSQL data server for session management

4. Congiruation Server
   a. Can access configuration data from source controls and repositories like GitHub
5. Built in monitoring service like acutor

Springboot

Friday, November 6, 2020 2:35 PM

Problems that Springboot solves

1. An enterprise application has multiple dependencies (Maven)
    a. Can we reduce the number of dependencies in our pom.xml
2. Each Dependencies have sub-dependencies
3. Often those dependencies may conflict with each other.
4. There are multiple frameworks to carry out a particular task
    a. Examples
        i. MVC design — struts or spring?
        ii. Which JSON parser to use
        iii. Which webserver to use?
    b. We need someone/something to suggest what would be a good choice.
5. How can we make our deployment easier?

What is spring boot?

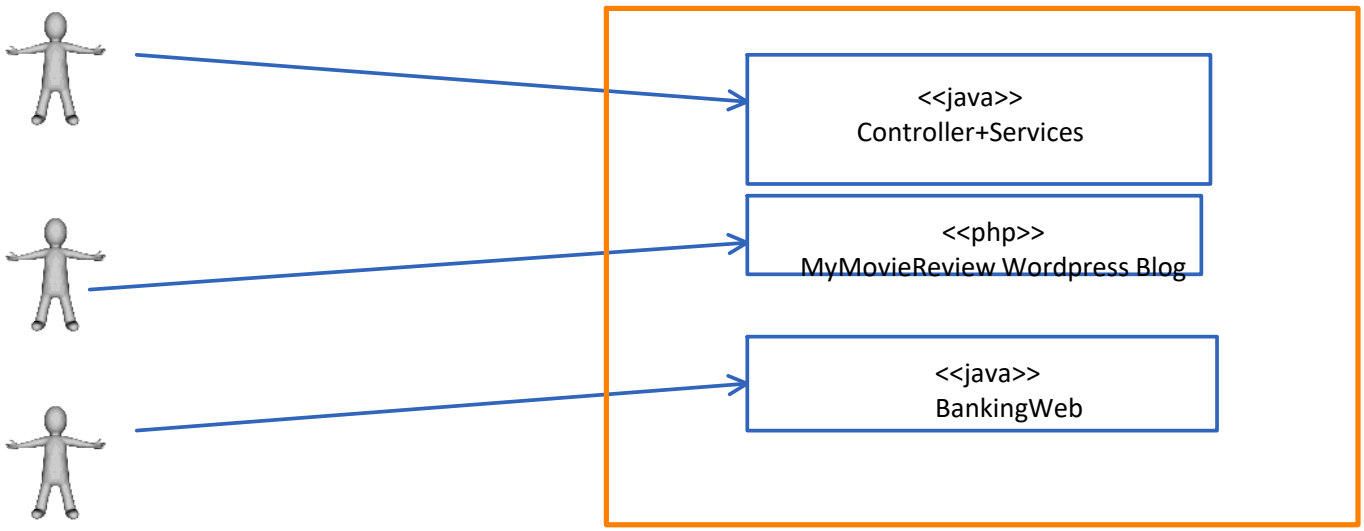
- Spring boot is an Opinionated Framework
    - It provides best practices guidelines for developing Application
    - It recommends tools and libraries that we can use for a good spring application

- It is like a supercharged maven
    - When you need to apply a functionality, Spring boot can provide
        - A pre-configured set of dependencies by include just a single maven dependency
        - A preconfigured set of configurations and beans to get started.
- It provides a containerless web server
    - Your application doesn't run on a webserver, A small webserver runs inside your application.
- A rich set of library and dependencies for different application needs.

Traditional web application runs withing a server

- Traditionally a web server like apache can host multiple web application
- Those application can be written in different programming language.

A Traditional Apache Server



Installation of Java Application

- Java Application (servelet,controller, pages, libraries) are compressed into an archive
- This archive is a **renamed jar file** known as **war file (web-archive)**
- The war file is copied into apache server and configured to run within the server
    - Web Servers are therefore called **Web container or Application containers**
    - Our application runs on a large web application

Advantage

- Single central server
- Multi-capacity
- Shared computer to host many small Applications

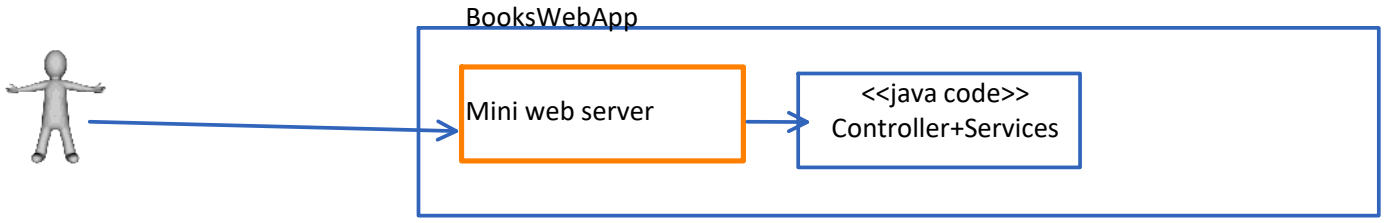
Problem

- Server is heavy weight as it is designed to soupport mulitple technologies
- Complex installation
- Not Scalable
- If Server crashes all the applications running on server will crash and go down.

Spring Boot Microservice Application Design

- A Spring boot is compiled as a stand alone **jar file** rather than a **war file**
    - It can also be compiled as **war file** if required.
- The Application contains a mini version of web server
    - This webserver knows how to serve Java request only
    - It doesn't have other language capability
- The web server is started within and for your application only
- If we run application multiple times, multiple copies of the web server will run
    - Each web server will run a single instance of the service
    - We can run as many instances of a service as we like.
- Your application will run anywhere a java application can run
- The entire application is packed as single jar.

- **java -jar booksweb.jar**

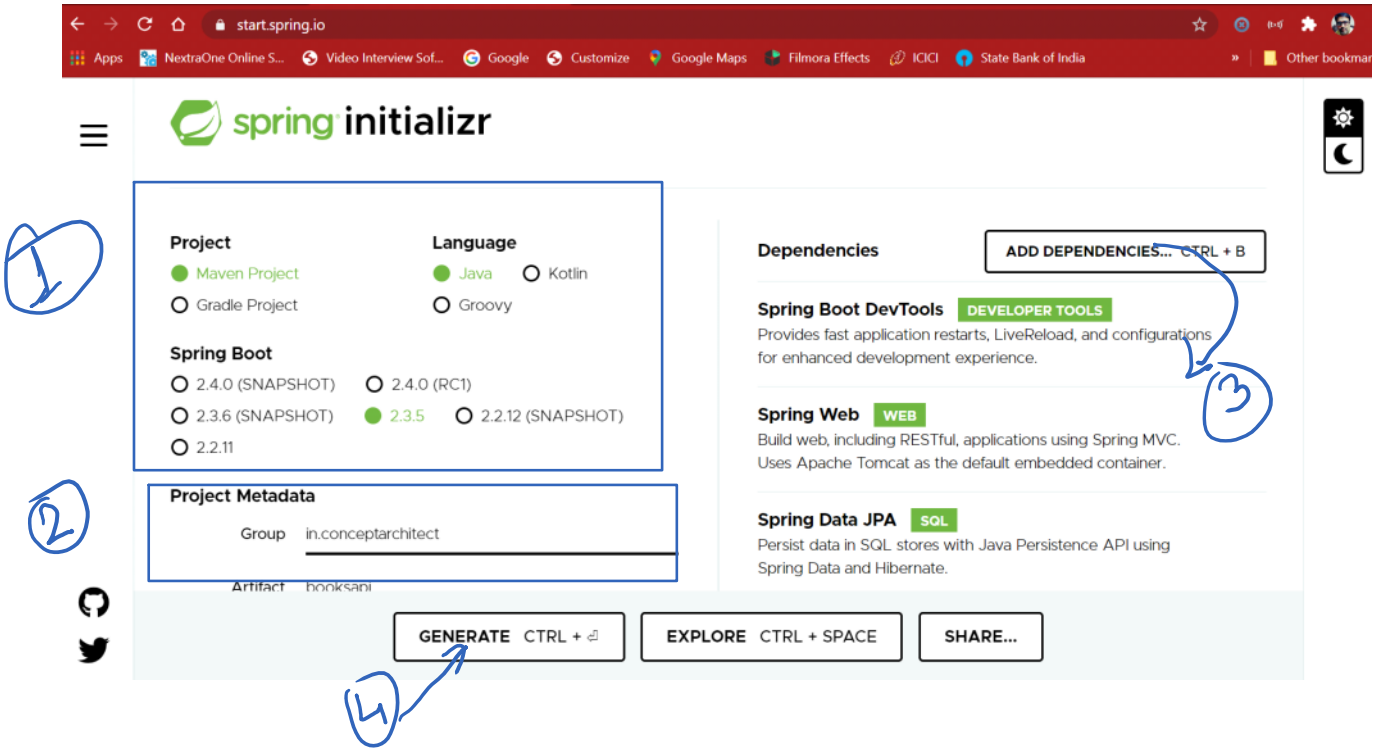


Creating Spring Boot App

Friday, November 6, 2020 3:17 PM

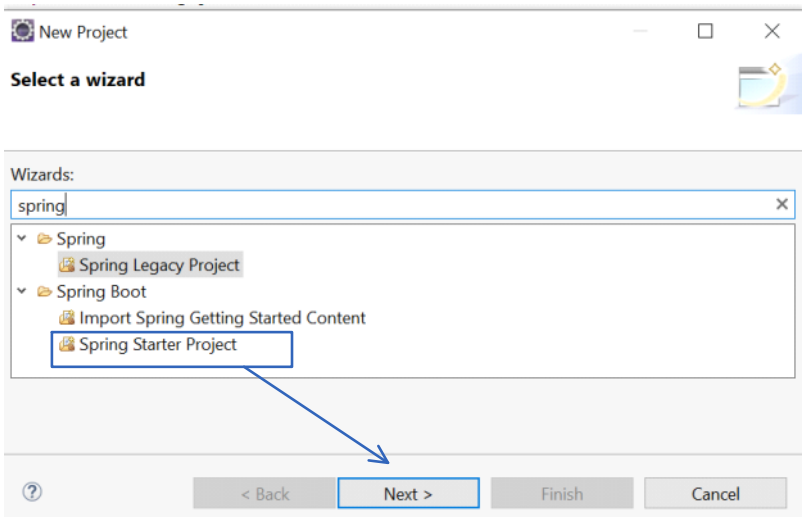
1. Create Initializer Project by visiting http://start.spring.io

1. Fill the basic information about the project
   a. Langauge
   b. Version
   c. Build tool
   d. GroupId
   e. ArtifectId
2. Select the dependencies
3. Generate the zip file



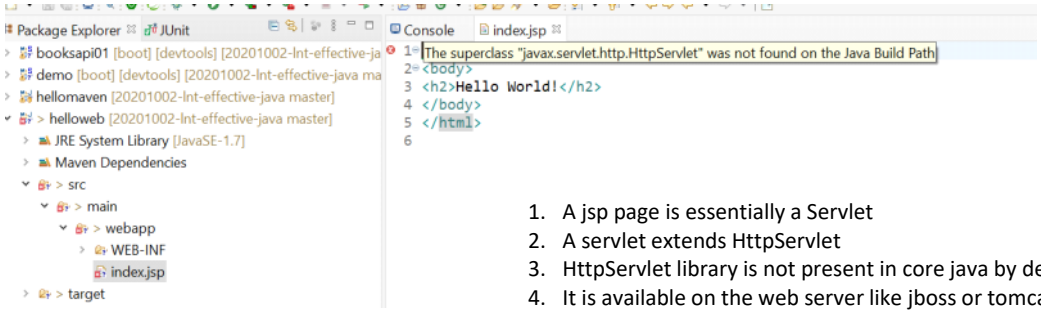
6. Import the project in your IDE as Existing Maven Project

Generate the Project directly in Eclipse using Spring Tool Suite 4+


New Project
Select a wizard
Wizards:
spring
Spring
Spring Legacy Project
Spring Boot
Import Spring Getting Started Content
Spring Starter Project
< Back
Next >
Finish
Cancel

A Simple Web Application (No Spring)

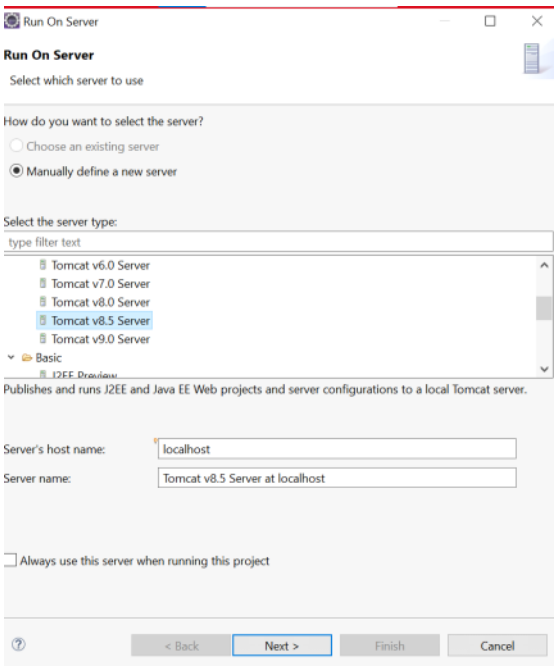
Friday, November 6, 2020 3:31 PM

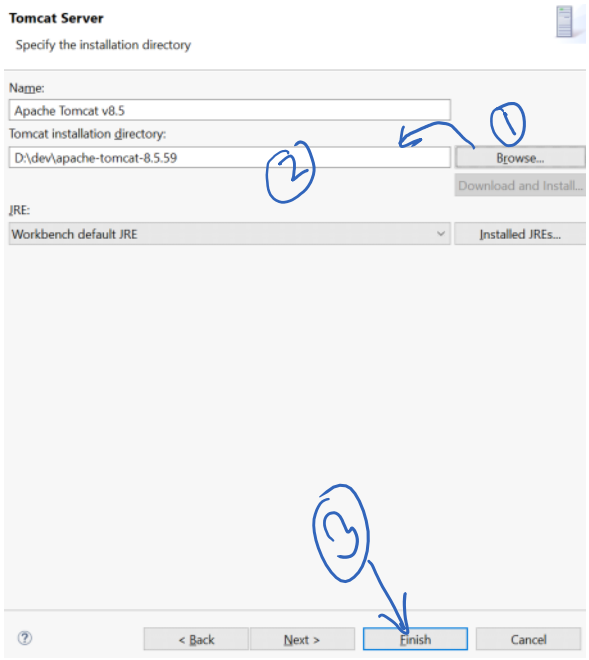


1. A jsp page is essentially a Servlet
2. A servlet extends HttpServlet
3. HttpServlet library is not present in core java by default
4. It is available on the web server like jboss or tomcat.
5. Eclipse fails to find it here
6. There would be no problem in running the code
    a. But it would be an error at compile time.

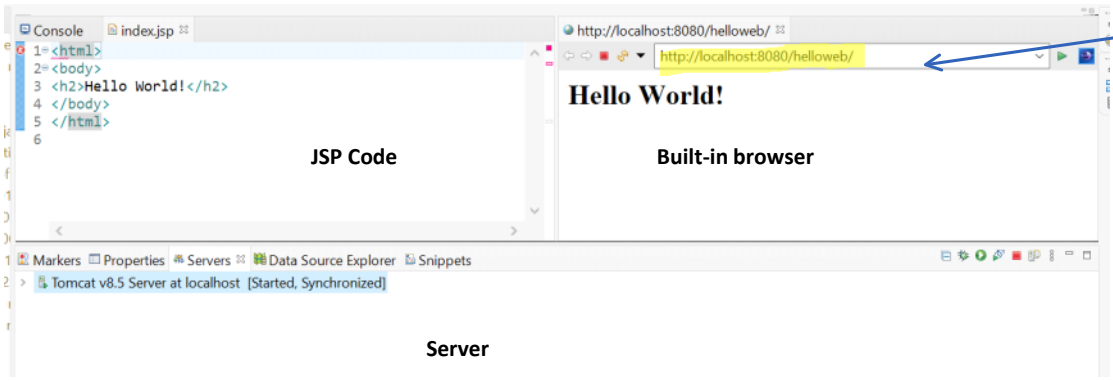
Running The Application

1. We can configure the apache or other web server from within eclipse
2. You may have to configure it once every workspace





Running the Application



URL includes application name

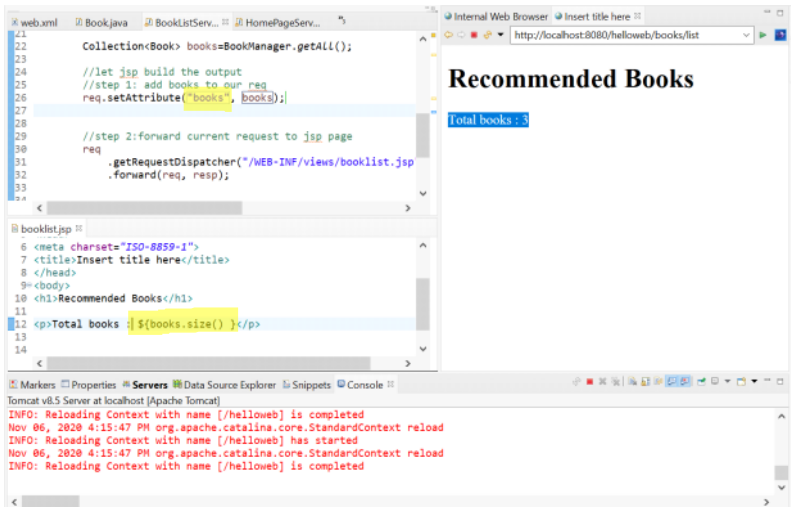
To work with servlets, we need servlet api available at runtime also

- We can include them using a maven dependency
- Scope:provided says, the dependency would be available at runtime
    - We need not include it in our project **war file**

```
<dependency>
   <groupId>javax.servlet</groupId>
   <artifactId>javax.servlet-api</artifactId>
   <version>3.1.0</version>
   <scope>provided</scope>
</dependency>
```

Web.xml configuration of you Servlet

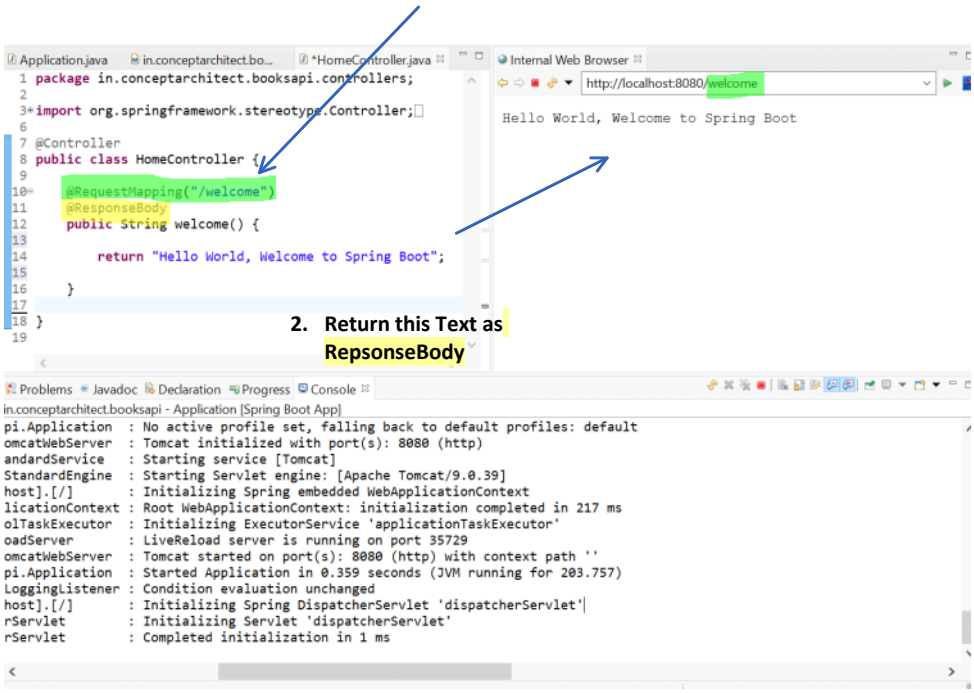
Passing Request between Servlet and Jsp



Adding a controller to spring boot

Friday, November 6, 2020 4:55 PM

1. On this URL



- @ResponseBody suggest that the current content is the actual data to be sent to the server
- Default setting is current result is the name of the jsp page that should be returned.

A Spring Boot Project by default is not configured to return HTML Output

- Microservices are generally RESTful in design
- They generate and return JSON data
- In modern web applications, generally HTML pages are created using front end technologies like Angular or React
- JSP based pages are not used regularly
- As a good practice Spring Boot is not configured for JSP page and trying to create a JSP page may result in error.
- **Spring boot can be configured to return JSP based Views also.**

How you configue Spring Boot App for JSP View rendering

1. Include following dependencies in your maven

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

</dependency>
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
</dependency>
```

2. Create your jsp files under special location

- src/main/resources/META-INF/resources/WEB-INF

Adding JPA to our Project

Friday, November 6, 2020 5:37 PM

1. Add JPA starter
2. Add the JDBC Connector
3. Create the Entity
4. Configure JPA in application.properties

Book.java | application.properties | data.sql | Application.java

```
1
2
3 spring.datasource.url=jdbc:mysql://localhost/booksdb?useSSL=false
4 spring.datasource.username=root
5 spring.datasource.password=@DM1n.
6
7 spring.jpa.hibernate.ddl-auto = create
8
9
10 spring.datasource.initialization-mode=always
```

- ddl-auto
  - Controls how the table would be created
  - Values
    - Create
      - Create everytime context starts
      - If exists drop and recreate
      - Good for development and testing
      - **NEVER USE IT IN PRODUCTION**
    - Update
      - Create if not exists
      - Update only if there is change
    - None
      - Never create
      - You have to create manually

- If data.sql is present use it to popular the database
- Good for testing with initial seed data
- Options
  - Always
    - Whenever database is created or updated
  - Never
    - Don't use the data.sql

Adding Autormatic JPARepository method by naming conventions

Friday, November 6, 2020 5:58 PM

1. JPA repository is automatically implemented by Spring
   a. You don't need to write class or query to generate them
2. You can write your additional methods and based on name convention they will be auto generated

MethodName syntax includes

- findBy
- fieldName
- Additional criteria like
  - Containing
  - IgnoreCase
  - Between
  - LessThan
  - Equals
  - NotEquals
  - NotNull
  - IsNull
  - And
  - Or
  - Like
  - Between
- If Field doesn't exist it gives an error
- If method name is well formed Spring automatically generates query based on method name

More Queries

```
List<User> findByName(String name)
List<User> findTop3ByAge()
List<User> findByNameIs(String name);
List<User> findByNameEquals(String name);
List<User> findByNameIsNot(String name);
List<User> findByNameIsNull();
List<User> findByNameIsNotNull();
List<User> findByActiveTrue();
List<User> findByActiveFalse();
List<User> findByNameStartingWith(String prefix);
List<User> findByNameContaining(String infix);
List<User> findByNameLike(String likePattern);
List<User> findByAgeBetween(Integer startAge, Integer endAge);
List<User> findByAgeIn(Collection<Integer> ages);
List<User> findByBirthDateAfter(ZonedDateTime birthDate);
List<User> findByBirthDateBefore(ZonedDateTime birthDate);
List<User> findByNameOrBirthDate(String name, ZonedDateTime birthDate);
List<User> findByNameOrBirthDateAndActive(String name, ZonedDateTime birthDate, Boolean active);
List<User> findByNameOrderByNameAsc(String name);

@Query("SELECT f FROM Foo f WHERE LOWER(f.name) = LOWER(:name)")
Foo retrieveByName(@Param("name") String name);
```

```
public interface PersonRepository extends Repository<User, Long> {
List<Person> findByEmailAddressAndLastname(EmailAddress emailAddress, String lastname);
// Enables the distinct flag for the query
  List<Person> findDistinctPeopleByLastnameOrFirstname(String lastname, String firstname);
  List<Person> findPeopleDistinctByLastnameOrFirstname(String lastname, String firstname);
// Enabling ignoring case for an individual property
  List<Person> findByLastnameIgnoreCase(String lastname);
  // Enabling ignoring case for all suitable properties
  List<Person> findByLastnameAndFirstnameAllIgnoreCase(String lastname, String firstname);
// Enabling static ORDER BY for a query
  List<Person> findByLastnameOrderByFirstnameAsc(String lastname);
  List<Person> findByLastnameOrderByFirstnameDesc(String lastname);
}
```