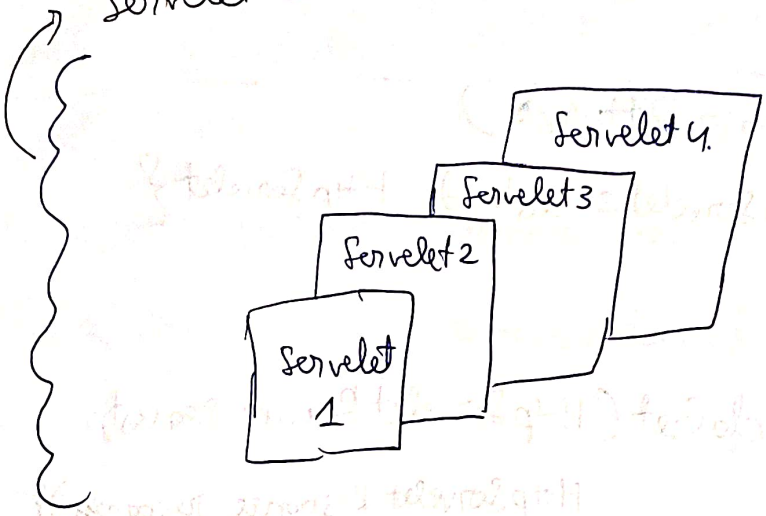
1. [Introduction of Spring Boot | Its advantage over Spring MVC and Servlet based Web Applications.]

To understand Spring Boot, lets first understand why Spring Boot came into picture and what problem it solved.

During 2015-16 Era when I was in class 11 or class 12, the then working software engineers did not have Spring or Spring Boot. They operated on something called **Servelet**

What is a Servelet?

* It is foundation for building web Applications.
* Servlet is a Java class which handles client request and processes it.
* Servelet container manages Servelets.



Lets see how servelet used to look like.

Servelet 1

```
@WebServelet("/demoservletone/*")
public class Demo Servellt1 extends HttpServelet{

    @Override
    protected void doGet(HttpServeletRequest request,
                         HttpServeletResponse response){

        String requestPathInfo = request.getPathInfo();

        if(requestPathInfo.equals("/")){
            // do something
        }
        else if(requestPathInfo.equals("/firstendpoint")){
            // do something
        }
        else if(requestPathInfo.equals("/secondendpoint")){
            // do something
        }
    }
}
```

Any Get request which come will call this method.

Servelet 2

```
@WebServelet("/demoservelettwo/*")
public class DemoServelet2 extends HttpServelet{

    @Override
    protected void doGet(HttpServeletRequest request,
                         HttpServeletResponse response){

        //do something
    }

    @Override
    protected void doPut(HttpServeletRequest request,
                         HttpServeletResponse response){

        //do something
    }
}
```

Now comes Web.xml which is making everything Complex.

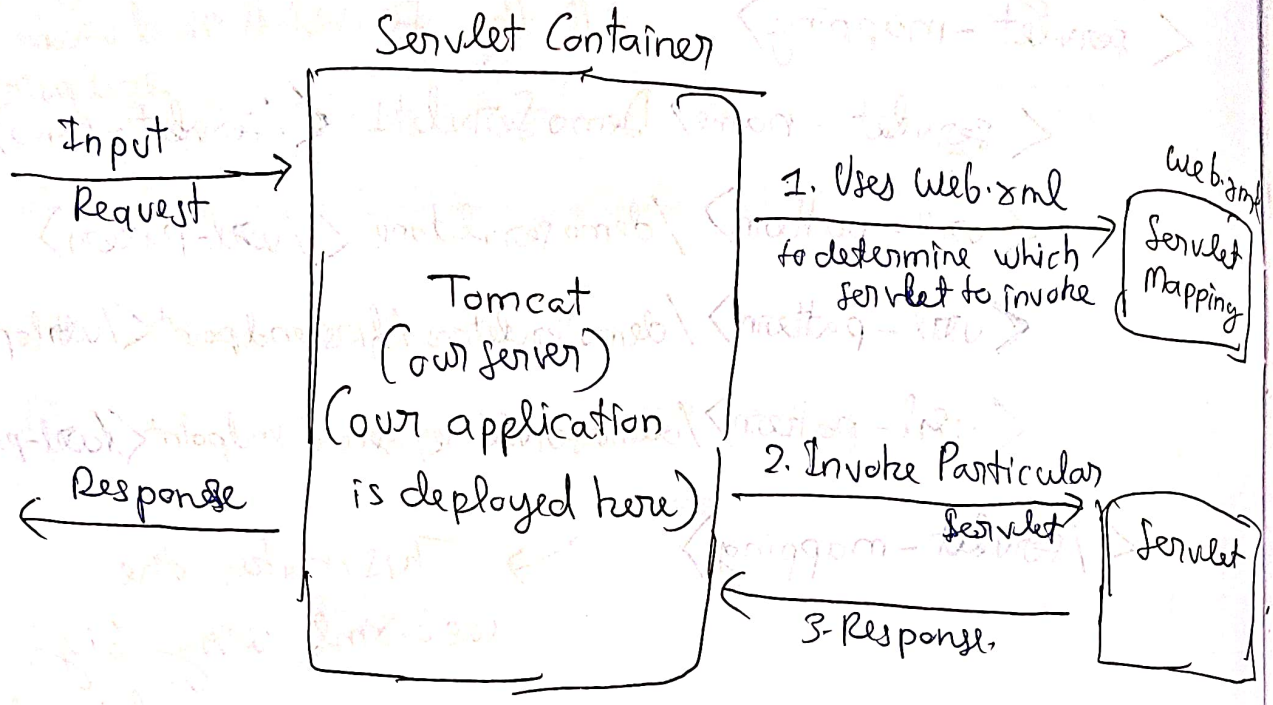
```
<!-- my first servelet configuration below-->
<servelet>
    <servelet-name> DemoServelet1 </servelet-name>
    <servelet-class> DemoServelet1 </servelet-class>
</servelet>
```

In this web.xml you have to specify all the servlets and their endpoints.

```
<servlet-mapping>
    <servlet-name>DemoServlet1</servlet-name>
    <url-pattern>/demoservletone</url-pattern>
    <url-pattern>/demoservletone/firstendpoint</url-pattern>
    <url-pattern>/demoservletone/secondendpoint</url-pattern>
</servlet-mapping>
```

→ This makes the web.xml very big and hard to maintain.

```
<!-- my second servlet configuration below -->
<servlet>
    <servlet-name>DemoServlet2</servlet-name>
    <servlet-class>DemoServlet2</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>DemoServlet2</servlet-name>
    <url-pattern>/demoservlettwo</url-pattern>
</servlet-mapping>
```



How Spring Framework solves Servlet Challenges.

1) Removal of web.xml

* Spring framework introduced Annotations based configurations. @GetMapping and @PostMapping

2) Inversion of Control (IOC)

* IOC is a more flexible way to manage object dependencies and its lifecycle.
  ↳ we will learn more about this, how @Singleton, @Request can create on demand beans and inject them. @Component.

3) Unit Testing is much harder in servlet.

Mocking any object is not easy

4) Difficult to manage REST APIs

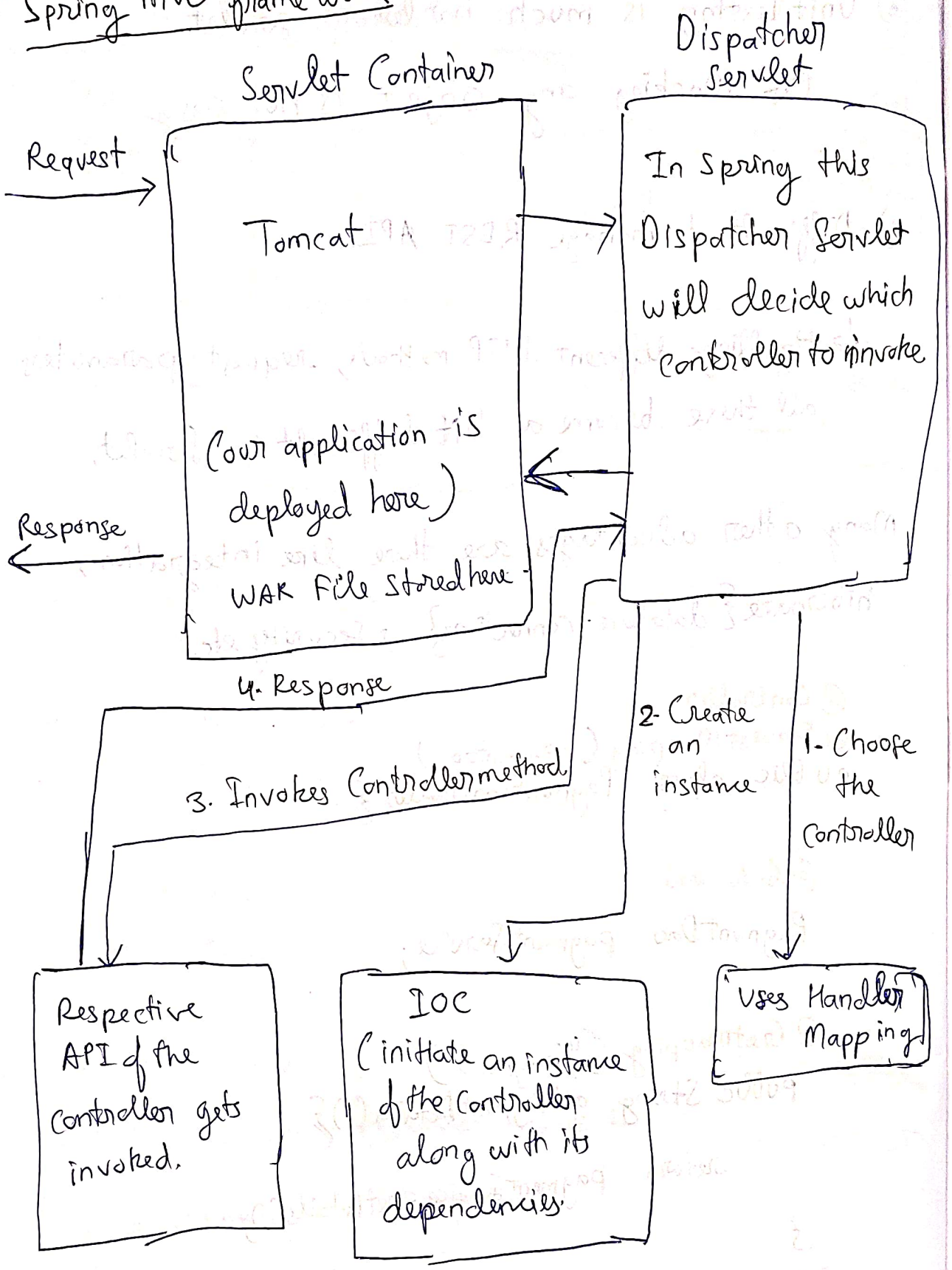
↳ Handling different HTTP methods, request parameters all these become a bit difficult in Servlet,

Many other advantages are there like integration, hibernate {database connections}, security etc

```
@Controller
@RequestMapping("/paymentapi")
public class PaymentController {

    @Autowired
    PaymentDao paymentService;

    @GetMapping("/payment")
    public String getPaymentDetails(){

        return paymentService.getDetails();
    }
}
```

» You can add multiple @GetMappings
@PostMappings.

Spring MVC frame work.



Lets see how a simple Spring MVC application looks like

pom.xml

```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>6.1.4</version>
  </dependency>
  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>servlet-api</artifactId>
    <version>2.5</version>
  </dependency>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13.2</version>
  </dependency>
</dependencies>
```

Controller.

```
@Controller
@RequestMapping("/paymentapi")
public class PaymentController{
  @Autowired
  IPaymentDao paymentService;

  @GetMapping("/payment")
  public String getPaymentDetails(){
    return paymentService.getDetails();
  }
```

Config Class

```
@Configuration
@EnableWebMvc
@ComponentScan(basePackages=
    "com.conceptandcoding")
public class AppConfig{
  //add configuration
}
```

If tells which dependencies to initiate.

```
public class MyApplicationInitializer extends AbstractAnnotationConfigDispatcherServletInitializer.{

  @Override
  protected Class<?>[] getRootConfigClasses(){
  }
  @Override
  protected String[] getServletMappings(){
    return new String[]{"/"}
  }
}
```

this tells which controllers to load.

here it loads all.

Now lets see what problems does SpringBoot solve which were there in Spring MVC.

In the previous Page you saw that even for writing a basic application, we need:

1. Controller
2. pom
3. Dispatcher Servlet
4. Config Classes

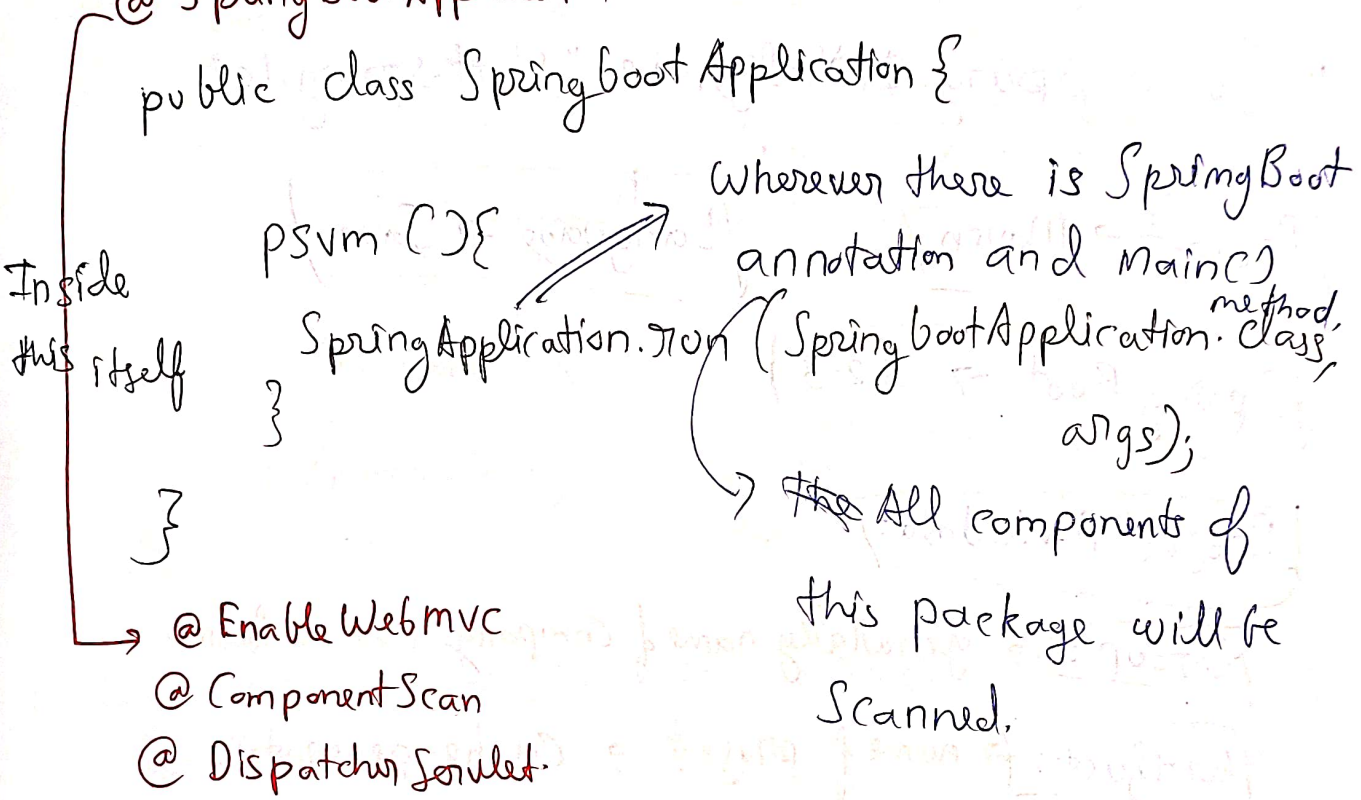
(a) In pom.xml we saw that will adding a dependency we have to add version as well. Now in future if we want to change the version of any of the dependencies then we have to make sure the new version is compatible with the other ~~versions of Spring~~ dependencies of Spring

Spring Boot solves this problem by removing <version>.

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

→ { Automatically latest and Compatible version chosen }

(b) Auto Configuration: No need for separately configuring "DispatcherServlet", "AppConfig", "ComponentScan" Spring boot adds internally by default.



(c) Embedded Server:

In traditional Spring MVC, we need to build a WAR file. Then we deploy the WAR file to a servlet container called Tomcat

In Spring Boot, servlet container is already embedded. Just run the application thats all.