Definition and Usage of @Media

The @media rule is used to define different style rules for different media types/devices.

In CSS2 this was called media types, while in CSS3 it is called media queries.

Media queries look at the capability of the device, and can be used to check many things, such as:

* width and height of the viewport
* width and height of the device
* orientation (is the tablet/phone in landscape or portrait mode?)
* resolution
* and much more

**Thymleaf Advantages over JSPs:**

**Thymeleaf is same as like normal html, and it replaces the JSPs with better and easy ways.**

On the bright side:

* ThymeLeaf is a healthy open source project: new features coming up each month, good documentation, responsive user forums…
* It is the ideal template engine if you want your web designer to be able to read your view files
* The Expression Language used (actually called Standard Dialect) is much more powerful than JSP Expression Language.
* Unlike JSPs, Thymeleaf works well for Rich HTML emails (see<http://www.thymeleaf.org/springmail.html).>

For more Detailed information :-

<https://spring.io/blog/2012/10/30/spring-mvc-from-jsp-and-tiles-to-thymeleaf>

**Overview of Spring Framework**

If you are a Java developer then there is a high chance that you might have heard about Spring framework and probably have used it in your projects. Spring framework was created primarily as a Dependency Injection container but it is much more than that.

**Spring is very popular because of several reasons:**

* Spring’s dependency injection approach encourages writing testable code
* Easy to use but powerful database transaction management capabilities
* Spring simplifies integration with other Java frameworks like JPA/Hibernate ORM, Struts/JSF/etc. web frameworks
* State of the art Web MVC framework for building web applications

**SPRING BOOT ADVANTAGES:**

Coming to SpringBoot, it replaces the SPRING framework in an easy way. Below are some advantages of using spring boot.

**1. Easy dependency Management**

* First thing to observe is we are using some dependencies named like **spring-boot-starter-\***. Remember I said “95% of the times I use the same configuration. So when you add **springboot-starter-web**dependency by default it will pull all the commonly used libraries while developing Spring MVC applications such as **spring-webmvc, jackson-json, validation-api**and**tomcat**.
* We have added **spring-boot-starter-data-jpa**dependency. This pulls all the **spring-data-jpa**dependencies and also adds **Hibernate**libraries because the majority of the applications use Hibernate as JPA implementation.

**2. Auto Configuration**

* Not only the **spring-boot-starter-web** adds all these libraries but also configures the commonly registered beans like **DispatcherServlet, ResourceHandlers, MessageSource**etc beans with sensible defaults.
* We also added **spring-boot-starter-Thymeleaf**which not only adds the Thymeleaf library dependencies but also configures **ThymeleafViewResolver** beans as well automatically.
* We haven’t defined any of the **DataSource, EntityManagerFactory, TransactionManager**etc beans but they are automatically gets created. How? If we have any in-memory database drivers like **H2**or **HSQL**in our classpath then SpringBoot will automatically create an in-memory **DataSource** and then registers **EntityManagerFactory, TransactionManager**beans automatically with sensible defaults. But we are using MySQL, so we need to explicitly provide MySQL connection details. We have configured those MySQL connection details in **application.properties** file and SpringBoot creates a **DataSource** using these properties.

**3. Embedded Servlet Container Support**

The most important and surprising thing is we have created a simple Java class annotated with some magical annotation **@SpringApplication**having a main method and by running that main we are able to run the application and access it at **http://localhost:8080/**.

**Where is the servlet container comes from?**  
We have added **spring-boot-starter-web** which pulls the **spring-boot-starter-tomcat**automatically and when we run the main() method it started tomcat as an **embedded container**so that we don’t have to deploy our application on any externally installed tomcat server.

By the way have you observe that our packaging type in **pom.xml** is **‘jar’ not ‘war’**. Wonderful!

**Ok, but what if I want to use Jetty server instead of tomcat?**  
Simple, exclude **spring-bootstarter-tomcat** from **spring-boot-starter-web** and include**spring-boot-starter-jetty**.

That’s it.

But, this looks all magical!!!

I can imagine what you are thinking. You are thinking like SpringBoot looks cool and it is doing lot of things automatically for me. But still I am not fully understanding how it is all really working behind the scenes. Right?

I can understand. Watching a magic show is fun normally, but not in Software Development. Don’t worry, we will be looking at each of those things and explain in detail how things are happening behind the scenes in future articles. But I don’t want to overwhelm you by dumping everything onto you right now in this article.

### Summary

In this article we had a quick overview of various Spring configuration styles and understand the complexity of configuring Spring applications. Also, we had a quick look at SpringBoot by creating a simple web application.