CONFIGURACIÓN DE THYMELEAF

CONFIGURACIÓN THYMELEAF

- Spring Boot nos ofrece una configuración *por defecto*.
- Varias vías de modificar la configuración
  - *application.properties*
  - JavaConfig

1.

CONFIGURACIÓN A TRAVÉS DE *PROPERTIES*

FICHERO application.properties

- Nos permite configurar cientos de aspectos a través de propiedades.
- Thymeleaf tiene un conjunto de ellas.
- En esta lección trabajaremos las más comunes.

FICHERO application.properties

Valor por defecto

```
spring.thymeleaf.cache=true
```

- Indica si se va a realizar un cacheo de plantillas.
- Si lo establecemos a *false*, podemos hacer un *hotswapping*.

```
spring.thymeleaf.check-template=true
```

- Comprueba si la plantilla existe antes de intentar renderizarla

FICHERO application.properties

```
spring.thymeleaf.check-template-location=true
```

- Comprueba si la ruta de la plantilla existe

```
spring.thymeleaf.enabled=true
```

- Podemos indicar si thymeleaf está habilitado o no

FICHERO application.properties

```
spring.thymeleaf.encoding=UTF-8
```

- Codificación de las plantillas

```
spring.thymeleaf.mode=HTML
```

- Modo en el que trabajará Thymeleaf al procesar las plantillas.

FICHERO application.properties

```
spring.thymeleaf.prefix=classplath:/templates/
```

- Prefijo en la ruta de las plantillas

```
spring.thymeleaf.suffix=.html
```

- Sufijo en la ruta de las plantillas

2.

CONFIGURACIÓN A TRAVÉS DE *JAVACONFIG*

CONFIGURACIÓN JavaConfig

- Configuración *programática*.
- Una clase anotada con @Configuration
- Implementamos la interfaz ApplicationContextAware para acceder al ApplicationContext (necesario en Thymeleaf 3.X)

```
@Configuration
public class ThymeleafConfig implements
                    ApplicationContextAware {
}
```

CONFIGURACIÓN JavaConfig

- Configuramos el acceso al contexto
- Necesario desde Thymeleaf 3.X

```
private ApplicationContext applicationContext;

public void setApplicationContext(
            ApplicationContext applicationContext) {

        this.applicationContext = applicationContext;
}
```

https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/context/ApplicationContext.html

CONFIGURACIÓN JavaConfig

- Configuramos el componente que *resolverá* las plantillas

```
@Bean
public SpringResourceTemplateResolver templateResolver() {
                                                                    Sp
ringResourceTemplateResolver templateResolver = new
                                SpringResourceTemplateResolver();

          templateResolver.setApplicationContext(this.applicationCo
ntext);
templateResolver.setPrefix("classpath:/templates/");
templateResolver.setSuffix(".html");
templateResolver.setTemplateMode(TemplateMode.HTML);
templateResolver.setCacheable(true);
return templateResolver;
}
```

CONFIGURACIÓN JavaConfig

- Configuramos el motor que procesará las plantillas

```
@Bean
public SpringTemplateEngine templateEngine() {
 SpringTemplateEngine engine = new SpringTemplateEngine();
 //engine.setEnableSpringELCompiler(true);
 engine.setTemplateResolver(templateResolver());

 return engine;
}
```

CONFIGURACIÓN JavaConfig

- Configuramos el *view resolver*, que resuelve las vistas en Spring MVC

```
@Bean
public ViewResolver viewResolver() {
 ThymeleafViewResolver resolver =
            new ThymeleafViewResolver();
 resolver.setTemplateEngine(templateEngine());
 resolver.setCharacterEncoding("UTF-8");

 return resolver;
}
```