Spring Framework 시작하기 3️⃣

Environment

ApplicationContext에서 제공하는 또 다른 중요한 기능중의 하나인 Environment에 대해서 살펴보겠습니다.

```
public interface ApplicationContext extends EnvironmentCapable,
  @Nullable
  String getId();

  String getApplicationName();

  String getDisplayName();

  long getStartupDate();

  @Nullable
  ApplicationContext getParent();

  AutowireCapableBeanFactory getAutowireCapableBeanFactory() thr
}
```

Properties

Property Search Hierarchy

> The search performed is hierarchical. By default, system properties have precedence over environment variables. So, if the `my-property` property happens to be set in both places during a call to `env.getProperty("my-property")`, the system property value "wins" and is returned. Note that property values are not merged but rather completely overridden by a preceding entry.
>
> For a common `StandardServletEnvironment`, the full hierarchy is as follows, with the highest-precedence entries at the top:
>
> 1. ServletConfig parameters (if applicable — for example, in case of a `DispatcherServlet` context)
> 2. ServletContext parameters (web.xml context-param entries)
> 3. JNDI environment variables (`java:comp/env/` entries)
> 4. JVM system properties (`-D` command-line arguments)
> 5. JVM system environment (operating system environment variables)

https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-property-source-abstraction

YAML로 프로퍼티 작성

> YAML이라는 이름은 "YAML은 마크업 언어가 아니다 (YAML Ain't Markup Language)” 라는 재귀적인 이름에서 유래되었다. 원래 YAML의 뜻은 “또 다른 마크업 언어 (Yet Another Markup Language)”였으나, YAML의 핵심은 문서 마크업이 아닌 데이터 중심에 있다는 것을 보여주기 위해 이름을 바꾸었다. 오늘날 XML과 JSON이 데이터 직렬화에 주로 쓰이기 시작하면서, 많은 사람들이 YAML을 '가벼운 마크업 언어'로 사용하려 하고 있다.

(https://ko.wikipedia.org/wiki/YAML)

YAML 문법

⚡ 목표 YAML 문법에 대해 전반적으로 알아봅니다. 이미 익숙하더라도 한 번 더 체크해보세요! 쿠버네티스 실습에서 중요한 건 아키텍처, 그다음이 YAML입니다. 앞으로 YAML파일을 수십 개 만들 겁니다 😨 "블로그를

https://subicura.com/k8s/prepare/yaml.html#%E1%84%80%E1%85%B5%E1%84%87%E1%85%A9%E1%86%AB%E1%84%86%E1%85%AE%E1%86%AB%E1%84%87%E1%85%A5%E1%86%B8

https://www.baeldung.com/spring-yaml-propertysource

@ConfigurationProperties

스프링부트에서 외부 속성을 통해서 별도의 빈을 만들수 있게 @ConfigurationProperties 애노테이션을 지원합니다. 이 애노테이션을 이용하면 특정 그룹의 속성을 모델링 할 수 있고 빈으로 등록해서 사용할 수 있습니다.

스프링 Profile

스프링 프로파일은 애플리케이션 설정 일부를 분리하여 특정 환경에서만 사용 가능케합니다. 프로파일을 이용하면 여러 빈 정의들이 특정 프로파일에서만 동작하게 할수도 있고 특정 프로퍼티들을 특정 프로파일로 정의해서 해당 프로파일이 엑티브일때 적용되게 할 수도 있습니다. 그럼 빈 정의와 프로퍼티들에 프로파일을 적용해 보겠습니다.

리소스

스프링 애플리케이션을 만들다 보면 외부 리스소를 읽을 필요가 있습니다. 여기서 외부 리소스라면 이미지파일, 텍스트파일, 암복호화를 위한 키파일 등이 있을 수 있습니다.

스프링은 `Resource` 와 `ResourceLoader` 인터페이스를 제공함으로써 하나의 API 로 제공합니다.

```java
public interface Resource extends InputStreamSource {
  boolean exists();

  default boolean isReadable() { return this.exists(); }

  default boolean isOpen() { return false; }

  default boolean isFile() { return false; }

  URL getURL() throws IOException;

  URI getURI() throws IOException;

  File getFile() throws IOException;

  default ReadableByteChannel readableChannel() throws IOException {
    return Channels.newChannel(this.getInputStream());
  }

  long contentLength() throws IOException;

  long lastModified() throws IOException;

  Resource createRelative(String var1) throws IOException;

  @Nullable
  String getFilename();

  String getDescription();
}
```

`ResourceLoader`를 통해 `Resource` 를 가져올수 있습니다.

```java
import org.springframework.lang.Nullable;

public interface ResourceLoader {
  String CLASSPATH_URL_PREFIX = "classpath:";

  Resource getResource(String var1);

  @Nullable
  ClassLoader getClassLoader();
}
```

스프링 공식 문서를 보면 다양한 구현체를 볼 수 있습니다.

2.3. Built-in `Resource` Implementations

Spring includes several built-in `Resource` implementations:

- `UrlResource`
- `ClassPathResource`
- `FileSystemResource`
- `PathResource`
- `ServletContextResource`
- `InputStreamResource`
- `ByteArrayResource`

https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#resources-implementations

Let's write some code!