Spring Boot Starter 만들기

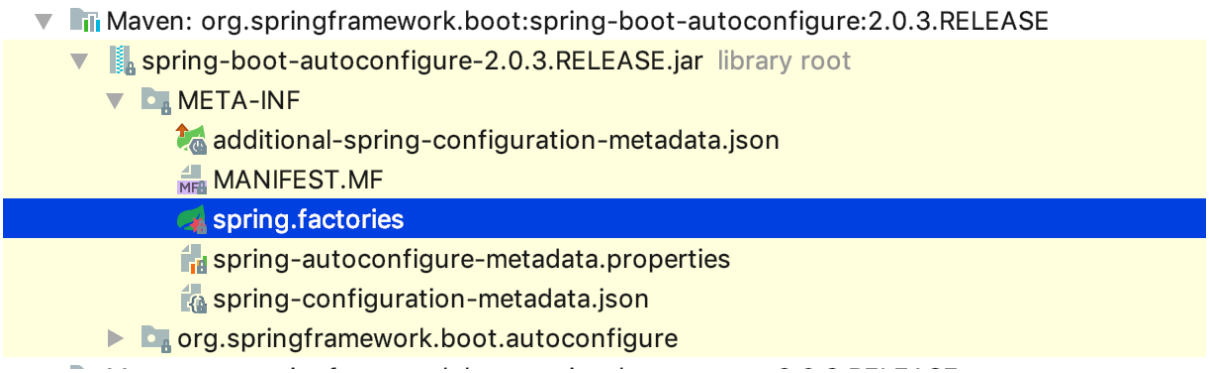
김성박(urstory@gmail.com)

Spring Boot 에서는 starter라고 불리는 라이브러리를 추가하면, 관련 라이브러리와 함께 자동 설정이 추가된다고 하였습니다.

이런 일은 어떻게 가능할까요?

Spring Boot 2.0에서 starter를 하나라도 추가한다면 spring-boot-autoconfigure 라는 라이브러리가 추가됩니다.

해당 라이브러리를 열면 다음과 같습니다.



위의 파일에서 spring.factories를 열면 다음과 같은 내용이 보여집니다.

```
# Auto Configure
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
org.springframework.boot.autoconfigure.admin.SpringApplicationAdminJmxAutoConfiguration,\
org.springframework.boot.autoconfigure.aop.AopAutoConfiguration,\
org.springframework.boot.autoconfigure.amqp.RabbitAutoConfiguration,\
org.springframework.boot.autoconfigure.batch.BatchAutoConfiguration,\
org.springframework.boot.autoconfigure.cache.CacheAutoConfiguration,\
org.springframework.boot.autoconfigure.cassandra.CassandraAutoConfiguration,\
org.springframework.boot.autoconfigure.cloud.CloudAutoConfiguration,\
org.springframework.boot.autoconfigure.context.ConfigurationPropertiesAutoConfiguration,\
org.springframework.boot.autoconfigure.context.MessageSourceAutoConfiguration,\
org.springframework.boot.autoconfigure.context.PropertyPlaceholderAutoConfiguration,\
org.springframework.boot.autoconfigure.couchbase.CouchbaseAutoConfiguration,\
org.springframework.boot.autoconfigure.dao.PersistenceExceptionTranslationAutoConfiguration,\
org.springframework.boot.autoconfigure.data.cassandra.CassandraDataAutoConfiguration,\
org.springframework.boot.autoconfigure.data.cassandra.CassandraReactiveDataAutoConfiguration,\
org.springframework.boot.autoconfigure.data.cassandra.CassandraReactiveRepositoriesAutoConfiguration,\
org.springframework.boot.autoconfigure.data.cassandra.CassandraRepositoriesAutoConfiguration,\
org.springframework.boot.autoconfigure.data.couchbase.CouchbaseDataAutoConfiguration,\
org.springframework.boot.autoconfigure.data.couchbase.CouchbaseReactiveDataAutoConfiguration,\
org.springframework.boot.autoconfigure.data.couchbase.CouchbaseReactiveRepositoriesAutoConfiguration,\
org.springframework.boot.autoconfigure.data.couchbase.CouchbaseRepositoriesAutoConfiguration,\
org.springframework.boot.autoconfigure.data.elasticsearch.ElasticsearchAutoConfiguration,\
org.springframework.boot.autoconfigure.data.elasticsearch.ElasticsearchDataAutoConfiguration,\
org.springframework.boot.autoconfigure.data.elasticsearch.ElasticsearchRepositoriesAutoConfiguration,\
org.springframework.boot.autoconfigure.data.jpa.JpaRepositoriesAutoConfiguration,\
```

스프링 부트 애플리케이션은 spring.factories파일의 org.springframework.boot.autoconfigure.EnableAutoConfiguration 키로 설정된 모든 설정파일을 읽어들여 자동으로 설정할 수 있는 것들을 설정하려고 합니다.

위의 클래스 목록 중에서 다음의 클래스를 보도록 하겠습니다.

org.springframework.boot.autoconfigure.web.servlet.WebMvcAutoConfiguration,₩

이름만 봐도 딱 느낌이 옵니다. 웹과 관련된 자동 설정입니다.

```
@Configuration
@ConditionalOnWebApplication(
    type = Type.SERVLET
)
@ConditionalOnClass({Servlet.class, DispatcherServlet.class,
WebMvcConfigurer.class})
@ConditionalOnMissingBean({WebMvcConfigurationSupport.class})
@AutoConfigureOrder(-2147483638)
@AutoConfigureAfter({DispatcherServletAutoConfiguration.class,
ValidationAutoConfiguration.class})
public class WebMvcAutoConfiguration {
    public static final String DEFAULT_PREFIX = "";
    public static final String DEFAULT_SUFFIX = "";
    private static final String[] SERVLET_LOCATIONS = new
String[]{"/"};

    public WebMvcAutoConfiguration() {
    }

    @Bean
    @ConditionalOnMissingBean({HiddenHttpMethodFilter.class})
```

```
    public OrderedHiddenHttpMethodFilter hiddenHttpMethodFilter() {
        return new OrderedHiddenHttpMethodFilter();
    }
```

해당 클래스를 열어보면 위와 같은 코드가 나옵니다.

@Conditional 로 시작 하는 애노테이션은 조건에 따라서 해당 설정을 적용하라는 의미입니다. @ConditionalOnWebApplication은 web환경에서 해당 설정이 사용된다는 것이고, @ConditionalOnClass({Servlet.class, DispatcherServlet.class, WebMvcConfigurer.class})은 해당 클래스가 클래스 패스에 있을 때 설정한다는 것입니다. 즉 @Conditional로 시작하는 애노테이션을 이용해서 특정 상황에서만 Bean이 생성되거나 설정이 되도록 합니다.

이런 형식으로 만들어진 설정 파일이 모두 실행되고, Spring Boot제작자가 만들어놓은 최적의 설정이 적용되게 됩니다.

이러한 Starter를 직접 만들려면 어떻게 해야할까요?

일단 maven 프로젝트를 생성합니다.

pom.xml 파일을 다음과 같이 수정합니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>examples.boot</groupId>
    <artifactId>my-spring-boot-starter</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-autoconfigure</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-autoconfigure-
processor</artifactId>
            <optional>true</optional>
        </dependency>

        <dependency>
```

```
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-configuration-
processor</artifactId>
            <optional>true</optional>
        </dependency>

    </dependencies>

    <!-- 의존성을 관리해주는 dependecyManagement 영역을 추가한다. -->
    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-dependencies</artifactId>
                <version>2.0.2.RELEASE</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>
</project>
```

groupId와 artifactId는 사용자가 원하는대로 수정해주세요. 보통 사용자가 만드는 starter는 groupId는 도메인 주소를 거꾸로 적고, artifactId는 spring-boot-starter로 끝나게 적습니다.

Dependencies와 dependencyManagement는 위와 같이 작성합니다.

Spring Boot를 사용하다보면 DataSource등의 객체가 자동으로 설정되는 것을 보았습니다. 자동으로 설정되기 위한 클래스를 하나 작성합니다.

ServerInfo.java

```
package examples.boot;

// 1. 자동으로 설정될 클래스
public class ServerInfo {
    private String address;
    private int port;

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public int getPort() {
        return port;
    }
```

```
    public void setPort(int port) {
        this.port = port;
    }

    @Override
    public String toString() {
        return "ServerInfo{" +
                "address='" + address + '\'' +
                ", port=" + port +
                '}';
    }
}
```

ServerInfo.java는 address와 port속성을 가집니다. 해당 필드에 대한 setter, getter메소드를 만듭니다. 사용자가 Lombok 사용을 원하지 않을 수 있으니, Lombok은 사용하지 않습니다.

ServerInfoConfiguration.java

```
package examples.boot;

import
org.springframework.boot.autoconfigure.condition.ConditionalOnMissing
Bean;
import
org.springframework.boot.context.properties.EnableConfigurationProper
ties;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

// 2. ServerInfo Bean 을 생성하는 Java Config 를 작성한다.
@Configuration
@EnableConfigurationProperties(ServerInfoProperties.class)
public class ServerInfoConfiguration {

    @Bean
    public ServerInfo serverInfo(
            ServerInfoProperties serverInfoProperties){
        ServerInfo serverInfo = new ServerInfo();
        serverInfo.setAddress(serverInfoProperties.getAddress());
        serverInfo.setPort(serverInfoProperties.getPort());
        return serverInfo;
    }
}
```

ServerInfo 클래스에 대한 Bean을 생성하는 Java Config파일을 생성합니다. properties파일로 부터 설정을 읽어들이도록 하기 위해서 @EnableConfigurationProperties(ServerInfoProperties.class) 를 설정하였습니다.

ServerInfoProperties.java

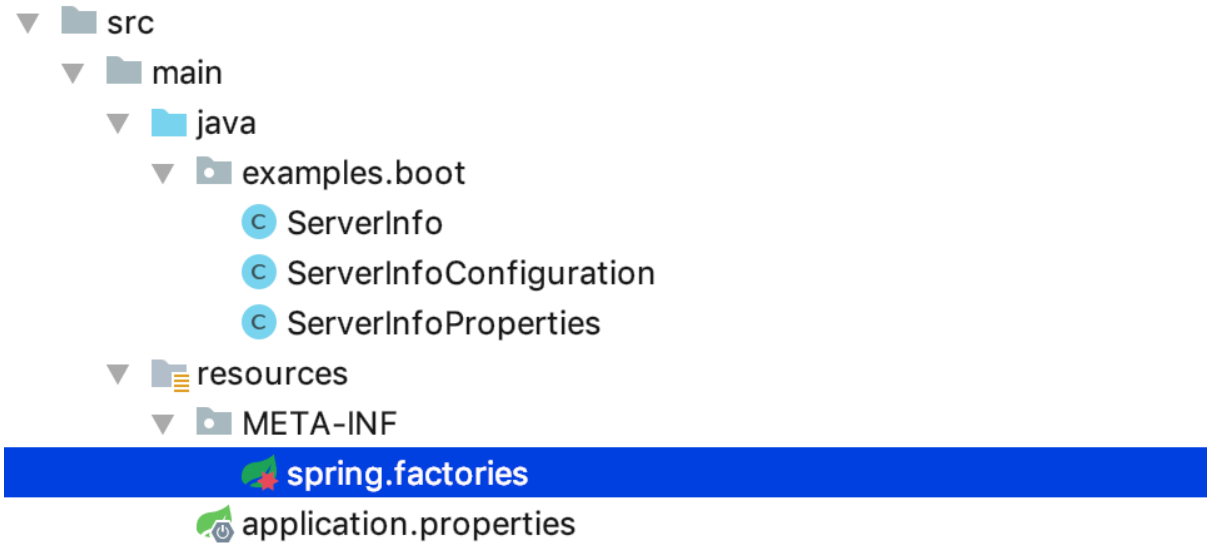
```java
package examples.boot;

import
org.springframework.boot.context.properties.ConfigurationProperties;

// serverinfo.address
// serverinfo.port
@ConfigurationProperties("serverinfo")
public class ServerInfoProperties {
    private String address;
    private int port;

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public int getPort() {
        return port;
    }

    public void setPort(int port) {
        this.port = port;
    }
}
```

properties파일 안에서 serverinfo로 시작하는 이름을 사용하기 위해서 @ConfigurationProperties("serverinfo")를 설정하였습니다. 해당 클래스는 serverinfo.address와 serverinfo.port 속성이름에 대한 정보를 읽어들입니다.

application.properties 파일

```
serverinfo.address=192.168.0.1
serverinfo.port=8888
```

application.properties 파일은 위와 같이 내용을 작성합니다.



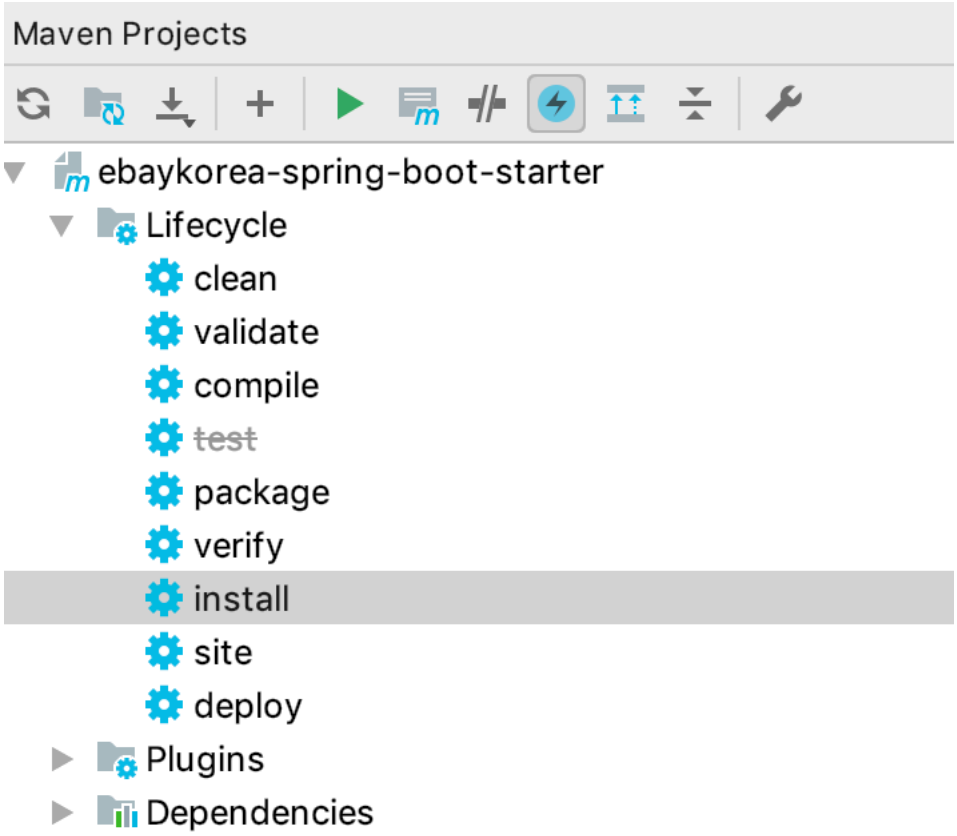
starter 에서 가장 중요하다고 말할 수 있는 spring.factories 파일을 작성합니다.

spring.factories

```
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
  examples.boot.ServerInfoConfiguration
```

spring.factories 파일에는 자동으로 읽어들일 Java Config 를 적습니다.

여기까지 작성했다면 maven 의 goal 중 install 을 실행합니다.



mvn install 이라고 하면 됩니다. (위의 캡쳐를 보면 test 는 동작하지 않도록 꺼놨습니다.)

mvn install 을 하면 maven 로컬 저장소에 만들어진 jar 파일이 저장됩니다. 회사의 경우는 별도의 maven 저장소 서버(nexus 서버)에 저장을 한 후 사용하도록 합니다.

이제 새로운 spring boot 프로젝트를 생성합니다. 해당 프로젝트의 pom.xml 파일에 다음의 내용을 추가합니다.

pom.xml 파일에 추가할 내용
<dependency> <groupId>examples.boot</groupId> <artifactId>my-spring-boot-starter</artifactId> <version>1.0-SNAPSHOT</version> </dependency>

사용자가 앞에서 작성한 jar 파일에 대한 의존성을 추가합니다. 로컬 maven 저장소에 저장되어 있기 때문에 사용할 수 있습니다.

StarterclientApplication.java

```java
package examples.boot.starterclient;

import examples.boot.ServerInfo;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
public class StarterclientApplication implements CommandLineRunner {
    @Autowired
    ServerInfo serverInfo;

    public static void main(String[] args) {
        SpringApplication.run(StarterclientApplication.class, args);
    }

    @Override
    public void run(String... args) throws Exception {
        System.out.println(serverInfo);
    }
}
```

Spring Boot 시작점인 클래스가 CommandLineRunner 인터페이스를 구현하도록 작성합니다. ServerInfo 객체를 @Autowired 로 주입받습니다. (해당 객체에 대한 설정은 사용자가 직접 작성한 starter 에 있습니다.)

run()메소드에서 serverInfo 를 출력합니다.

출력된 결과는 다음과 같습니다. (이 때 주의해야할 점은 application.properties 파일이 없어야 됩니다.)

```
2018-10-10 11:02:56.047  INFO 31314 --- [  restartedMa
ServerInfo{address='192.168.0.1', port=8888}
```

사용자가 application.properties 파일을 작성한 후 servierinfo.address 와 serverinfo.port 값을 다음과 같이 작성한 후 실행하면 결과는 다음과 같이 나옵니다.

application.properties

```
serverinfo.address=localhost
serverinfo.port=100
```

```
2018-10-10 11:05:16.525  INFO 31348 --- [  res
ServerInfo{address='localhost', port=100}
```

실행한 결과는 다음과 같습니다.

정리.

스프링 부트 프로젝트를 생성할 때 start.spring.io 를 많이 이용합니다. IntelliJ 도 해당 사이트를 이용합니다. 해당 사이트에 대한 오픈소스는 https://github.com/spring-io/initializr 에서 받을 수 있습니다. 해당 소스를 이용해서 회사에다가도 spring boot 프로젝트를 생성하는 사이트를 제공할 수 있습니다. 여기에 회사에서 사용하는 starter 를 등록해서 프로젝트를 생성할 때 사용할 수도 있을 것입니다.

해당 내용을 이해함으로써 spring boot 의 자동 설정에대해서도 좀 더 잘 이해하리라 생각합니다.

감사합니다.