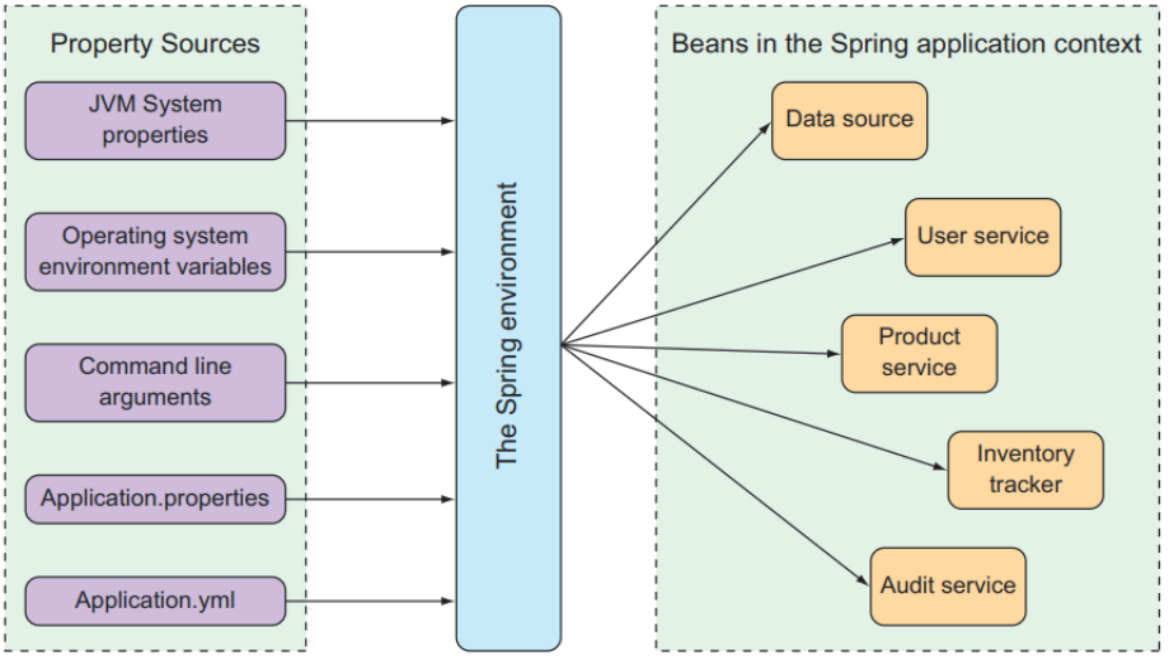
Chapter5_자율발표



1. Property Sources

Property

> DTO(Data Transfer Object) 또는 VO(Value Object)에서 볼 수 있음인스턴스 변수와 getter setter로 만들어진 형태객체와 관련하여 이름 붙여진 속성을 말하며 빈이 관리하는 데이터를 의미

Properties

- 응용 프로그램의 구성 가능한 파라미터들을 저장하기 위해 자바 관련 기술을 사용하는 파일들을 위한 파일 확장
- 키와 값을 쌍으로 갖는 자료구조 또는 파일
- 키와 값을 자료형이 모두 String(문자열)
- 변경 가능성이 있는 문자열을 별도의 properties 파일에 작성해두고 프로그램 내에서 호출하는 형태로 많이 사용
- 변경 가능성이 있는 문자열을 String 타입의 객체에 저장해두고 사용하면 수정을 할 때 컴파일을 다시해서 실행
    - 컴파일을 다시해서 실행하게 되면 시간이 오래 걸리게 되면 컴파일을 하다 예기치 못한 오류를 발생가능성 존재

    - 별도의 파일이나 테이터베이스를 이용하게 되면 컴파일을 다시 하지 않고 재실행 만으로 변경 내용을 적용할 수 있기 때문에 예기치 못한 오류를 발생시킬 가능성이 줄어듦
- Environment의 property 기능은 외부 설정, 속성을 다룰 수 있도록 함

---

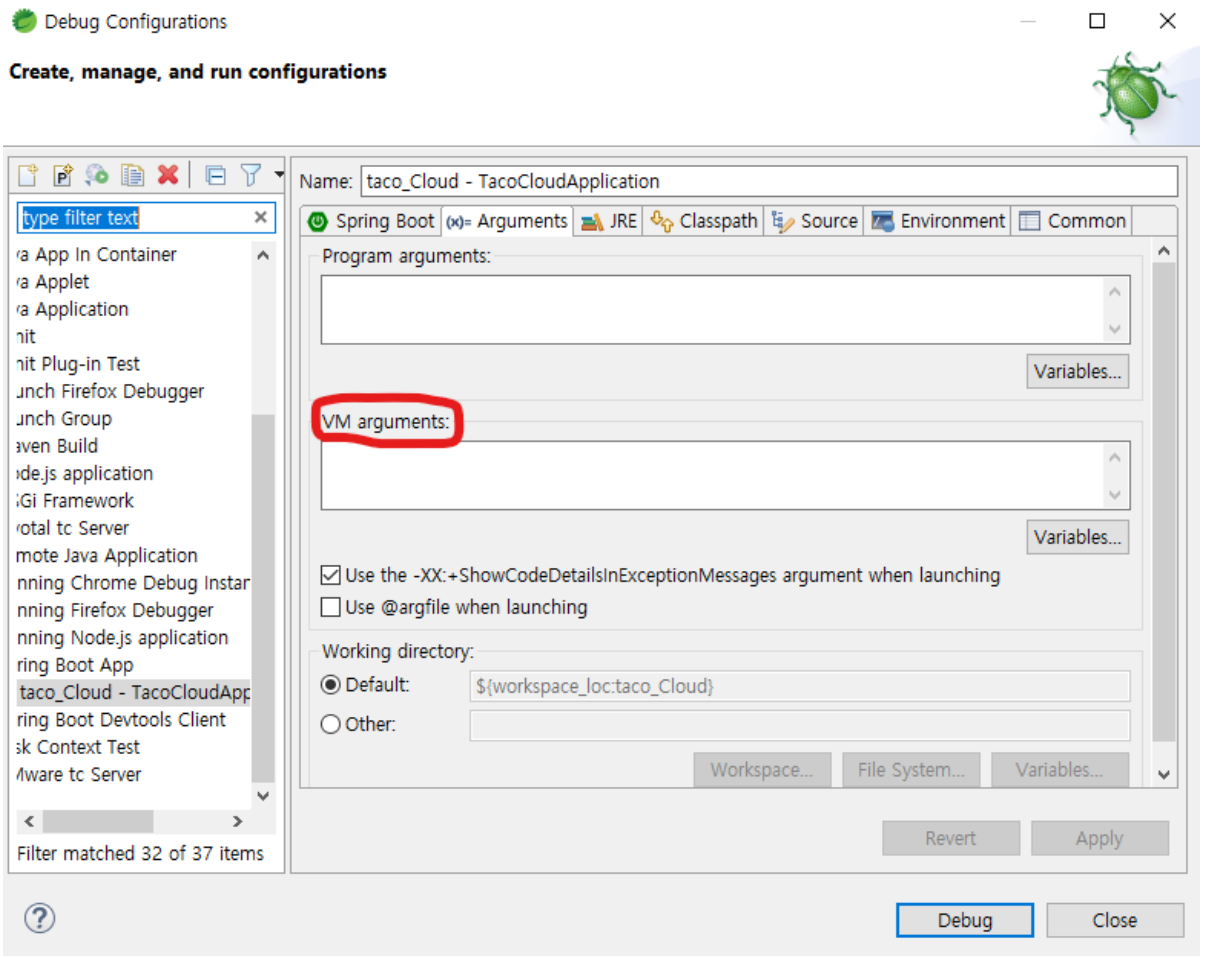
1. JVM System property

VM options에 key=value 형식으로 속성을 설정하고 이를 추출하는 방식edit configurations -> VM options 에 아래 "key", "value"를 대신해서 원하는 키와 값을 넣어준다.

VM arguments(VM options)

JVM에 전달되어 VM의 동작방식 및 시스템 속성을 정의합니다.

- X 옵션 : JVM Heap Memory, Permanent Generation , Direct Buffer 크기 지정 등
- D 옵션 : 전역 시스템 속성 정의
    - -DKey = Value
    - 단건조회 : System.getPropertie(String key)
    - 전체조회 : System.getProperties()



VM Options

<VM Options>

```
-Dspring.profiles.active="A_Group"
-Ditem="user"
```

<ApplicationRunner>

```
@Component
public class AppRunner implements ApplicationRunner {
    @Autowired
    Environment environment;

    @Override
    public void run(ApplicationArguments args) throws Exception{
        String item = environment.getProperty("item");
        System.out.println(item);  // user 출력
    }
}
```

2. OS 환경변수(Operating System environment variables)

개발환경, 운영환경(배포환경)에 필요한 프로퍼티는 각 OS 환경 변수를 통해 Spring Boot 기동시점에 설정 정보 주입

- Spring Boot에서 요구하는 문법에 맞춰서 OS 환경변수를 설정하면 됨
- application.properties 혹은 **application.yml** 파일에서 사용했던 설정값을 아래와 같은 3가지 규칙에 따라 변환해서 사용
  1. Replace dots (.) with underscores (_).
  2. Remove any dashes (-).
  3. Convert to uppercase.
  4. property를 List로 받아올 경우 [0] -> **0**

```
THIRDPARTIES_KAKAO_URL=https://kakao.com
THIRDPARTIES_KAKAO_NAME=kakao
THIRDPARTIES_NAVER_URL=https://naver.com
THIRDPARTIES_NAVER_NAME=naver
THIRDPARTIES_0_URL=https://kakao.com
THIRDPARTIES_0_NAME=kakao
THIRDPARTIES_1_URL=https://naver.com
THIRDPARTIES_1_NAME=naver
```

- OS 환경변수 가져오는 법

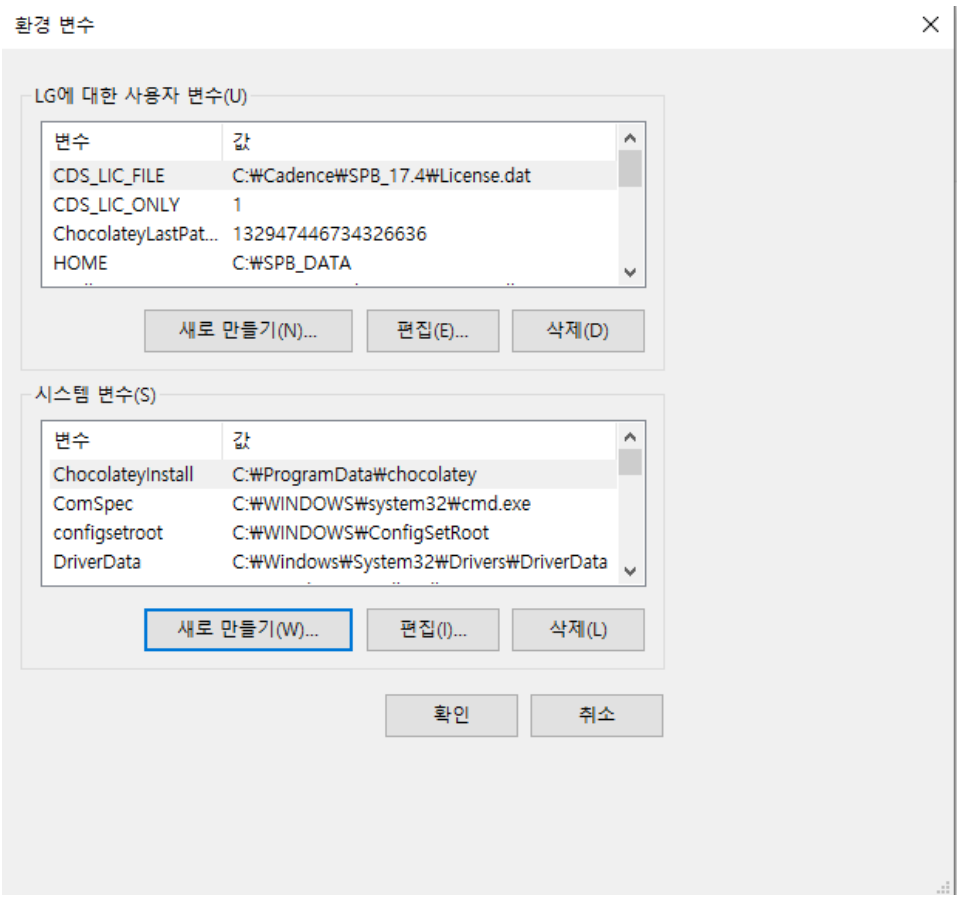
System.getenv()

자바 API에서는 환경 변수를 읽어오는 아래 두 메서드를 제공

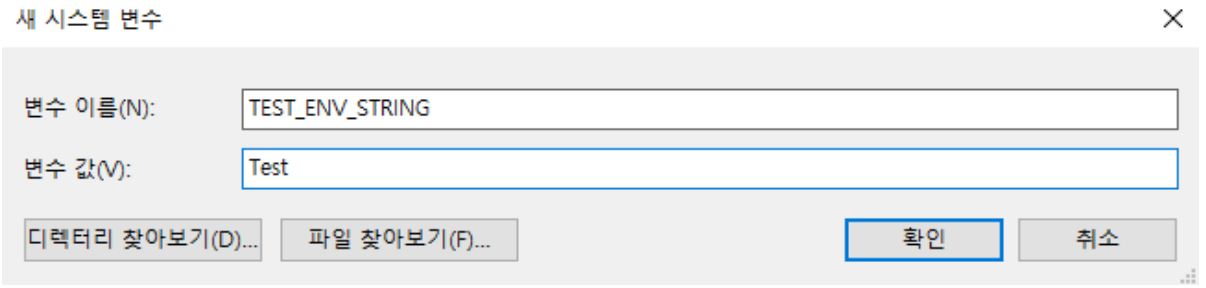
- System.getenv()
  - 모든 시스템 환경변수에 대한 값을 key, value (Map<String,String>) 형태로 반환합니다.
- System.getenv(String name)
  - name에 해당되는 시스템 환경 변수의 값을 반환합니다.

윈도우 환경 변수

- 제어판 -> 시스템 -> 고급 시스템 설정 -> 환경 변수



환경 변수 설정 1



환경 변수 설정 2

```
public void testGetEnvString() {
    //시스템 환경변수 값 전체 가져오기 (key, value 형태)
    Map <String, String> map = System.getenv();
    for (Map.Entry <String, String> entry: map.entrySet()) {
      System.out.println("Variable Name:- " + entry.getKey() + " Value:- " +
entry.getValue());
    }

    //'TEST_ENV_STRING' 이라는 환경변수의 값 가져오기
    String testString = System.getenv("TEST_ENV_STRING");
  }
```

3. Command Line Arguments

Spring Boot는 기본적으로 애플리케이션 실행시(java -jar {application}.jar) 전달된 --(마이너스 문자 2개) 문자로 시작하는 모든 명령어 인자(커맨드라인 아규먼트)를 인식

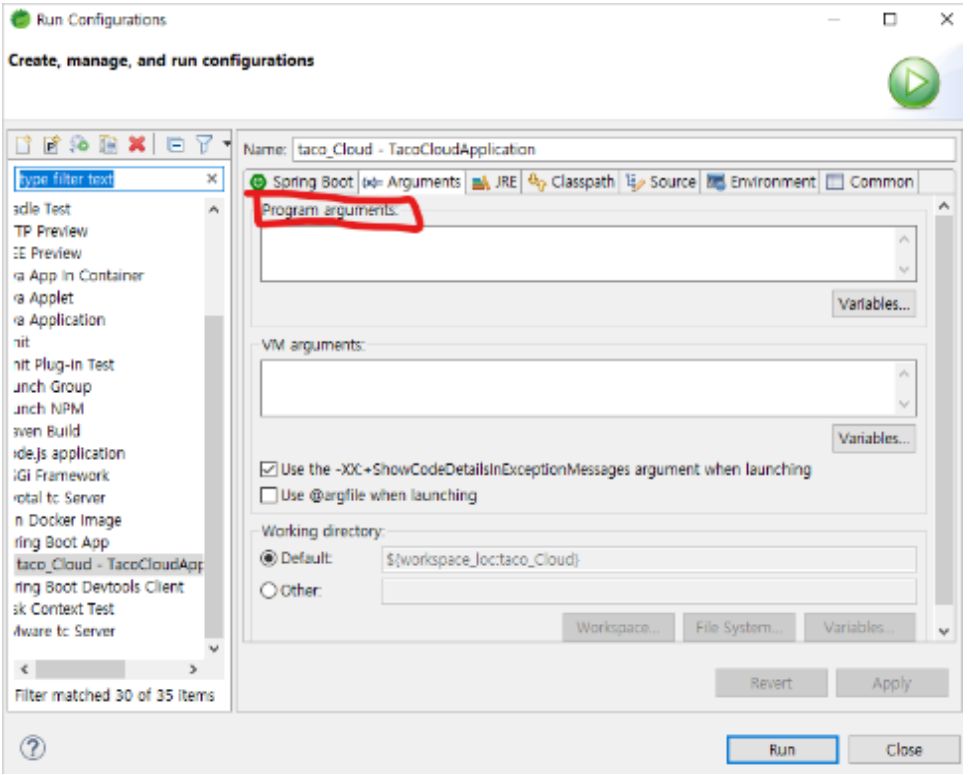
-

```
@Component
public class SomeSpringComponent {

    @Value("${some.option}")
    private String someOption;

    @Value("${some.option2:defaultValue}")
    private String someOption2;
    ...
}
```

- 전달된 명령어 인자는 @Value 어노테이션을 사용하여 스프링 빈의 변수로 주입하여 사용
- 만약 애플리케이션 실행시 --**some.option=someValue**라는 명령어 인자를 전달했다면 **${some.option}**을 명시하여 **someValue** 값을 주입
- 만약 전달되지 않은 명령어 인자를 주입한다면 주입 시점에 IllegalArgumentException 예외가 발생
  - 이 문제를 예방하기 위해 기본값을 주입 가능(문자 뒤에 기본값을 명시 가능. 위 예제의 경우 **defaultValue**라는 값이 주입)



Command Line Arguement

---

4. Application.properties

로컬 환경은 config data(예: application.properteis, application.yml)를 이용해서 설정하고 개발

```
third-parties.kakao.url=https://kakao.com
third-parties.kakao.name=kakao
third-parties.naver.url=https://naver.com
third-parties.naver.name=naver
third-parties[0].url=https://kakao.com
third-parties[0].name=kakao
third-parties[1].url=https://naver.com
third-parties[1].name=naver
```

- XXX.properties라는 환경변수를 담는 파일을 만든다.
  - .properties파일은 resources디렉토리 밑에 있어야 target파일 밑에 제대로 들어감.
- **XXX.properties파일** 안에 key, value 로 값을 넣어준다
- 설정한 환경변수를 가져올 때는 @PropertySource("XXX.properties 위치")를 사용
  - classpath : target의 classes 파일을 가리킴
- #--- -> 문서를 분할할 위치를 나타낼 수 있음
-

<읽어올 .properties 파일 알려주기>

```
@SpringBootApplication
@PropertySource("classpath:/app.properties")
public class DemoApplication {

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

    }

}
```

<@Value("")로 가져오기>

```
@Component
public class AppRunner implements ApplicationRunner {

    @Autowired
    ApplicationContext context;

    @Value("{app.name:""}")
    String appName;
}
```

<Environment 객체로 가져오기>

- Environment의 getProperty("") 매서드 활용

```
@Component
public class AppRunner implements ApplicationRunner {

    @Autowired
    ApplicationContext context;


    @Override
    public void run(ApplicationArguments args) throws Exception {
        Environment env = context.getEnvironment();

        env.getProperty("app.name");
    }
}
```

5.Application.yml

로컬 환경은 config data(예: application.properteis, application.yml)를 이용해서 설정하고 개발계층 적 구성 데이터를 지정하기위한 편리한 형식

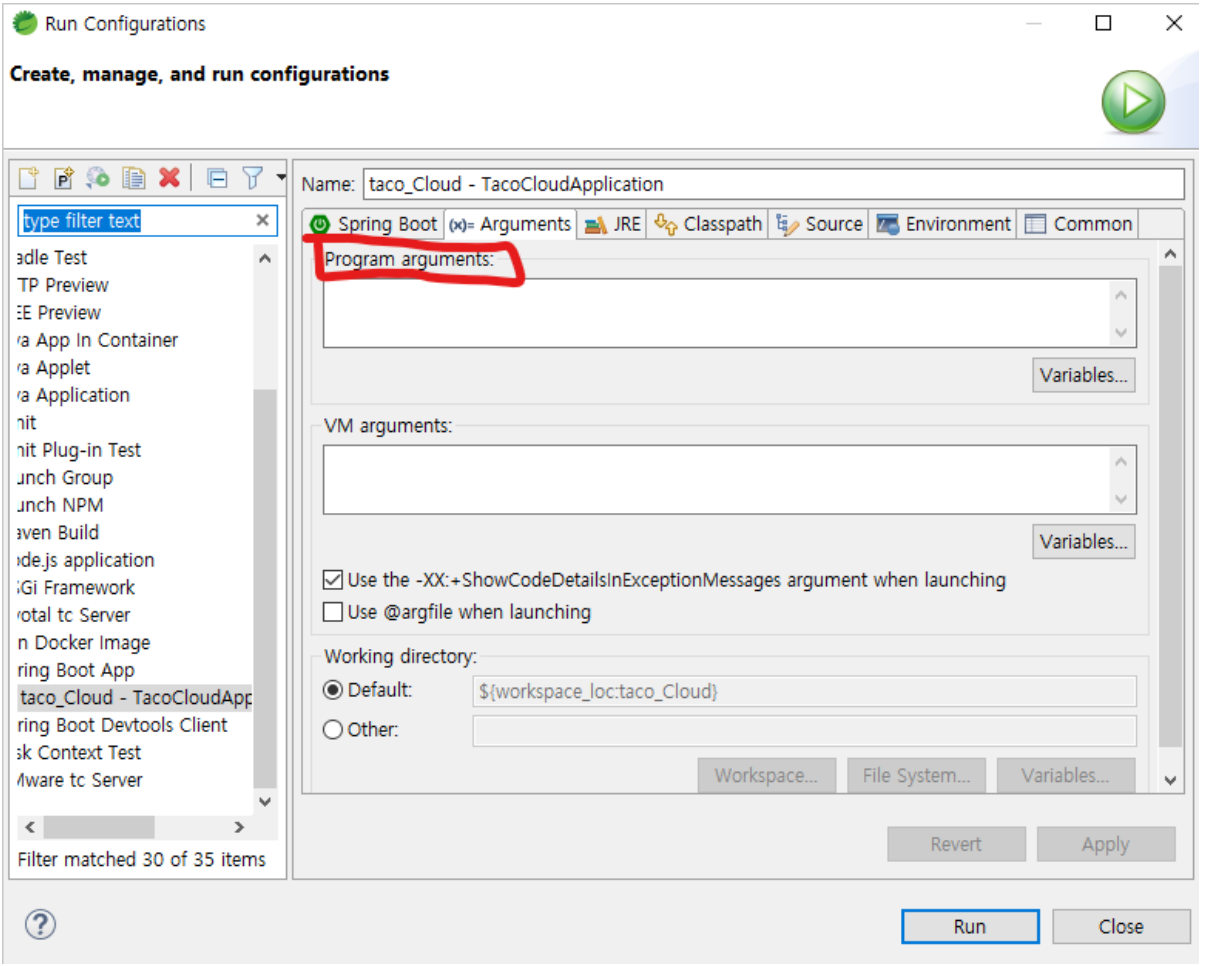
- 반복되는 접두사가 포함되지 않으므로 대체 속성 파일보다 더 읽기 쉬움
- List 표현은 - 하나로 표현
- --- -> **새 문서의 시작을 나타 내기** 위해 **세 개의 대시**
- 가져오는 방법은 @Value, Environment Abstraction(env.getProperty()), @ConfigurationProperties 사용 가능
    - @ConfigurationProperties(prefix="third-parties.kakao")

```
third-parties:
  kakao:
    url: https://kakao.com
    name: kakao
  naver:
    url: https://naver.com
    name: naver
third-parties:
  - url: https://kakao.com
    name: kakao
  - url: https://naver.com
    name: naver
```

- 위에서 언급한 Command Line Arguements 적용 가능

```
foo:
  bar: ${some.option:defaultValue}
```

- 애플리케이션 실행시 **—some.option=someValue** 옵션을 주었다면 위 **foo.bar=someValue**가 주입
- 옵션을 주지 않았다면 **foo.bar=defaultValue**가 주입



---

2. Beans in the Spring Application Context

실제 모두 다 쓰이는 건 아니겠지만 위에서 얻은 properties를 아래에 적용시킨다는 의미

1. Data Sources

DB와 관계된 커넥션 정보를 담고있으며 빈으로 등록하여 인자로 넘겨줌.이 과정을 통해 Spring은 DataSource로 DB와의 연결을 획득

2. UserService

주요기능인 로그인, 회원가입, 회원정보수정, 회원탈퇴 및 삭제 등을 비롯한 컨트롤러에서 검증할때 사용할 한명의 유저정보를 가져올 메소드와 전체유저의

정보를 가져올 메소드를 정의

```
package com.example.demo.service;
import java.util.List;
import com.example.demo.dto.UserDto;
public interface UserService {
    public boolean login (String userId, String userPw);
    public boolean join (UserDto user);
    public void modify (UserDto user);
    public void withdraw (String userId);
    public UserDto getUser (String userId);
    public List<UserDto> getUserList();
}
```

3. ProductService

```
<bean id = "inventoryService"
                        class = "com.example.InventoryService" />

<bean id = "productService"
                        class = "com.example.ProductService" />
        <contructor-arg ref="inventoryService" />
</bean>
```

```
@Configuration
public class ServiceConfiguration {
        @Bean
        public InventoryService inventoryService() {
                return new InventoryService();
        }
        @Bean
        public ProductService productService() {
                return new ProductService(inventoryService());
        }
}
```