초격차 패키지 Online.

IT 서비스 회사에서 사용하는 진짜 프로젝트 가성비있게 맛보기

Clip 1 | Git Flow 전략 알아보기

Clip 2 | Spring Multi Module 프로젝트 생성

Clip 3 | Spring Multi Module 의존성 설정

Clip 4 | Spring Multi Module 정리

Clip 5 | Profile 설정

Chapter 1에서 다룰 내용들

- Git Flow
- Multi Module 프로젝트
- Profile 설정

반드시 알아야만 하는 개념들

1 Git Flow 전략 알아보기

Git을 사용하는 기본적인 명령어

- Commit
- Push
- Pull

Git Flow 전략

- Git을 사용하여 개발하는 환경에서
- Branch 간의 문제 없이 배포까지 안정적으로 할 수 있도록
- Branch를 관리하는 전략이다.

주요 Branch

1. Main(= Master)
2. Dev
3. Feature
4. Release
5. Hotfix

실습 시나리오 리스트

- Release Branch 생성 후 추가 작업이 필요해질 경우
- Release Branch 생성 후 추가 작업이 없는 경우
- Hotfix가 나가야 할 경우

정기배포를 위한 Git Flow 전략

정기배포를 위한 GitFlow 전략

1.
Git Flow 전략 알아보기

- 요구 사항

1. Login 기능 개발
2. Logout 기능 개발

정기배포를 위한 GitFlow 전략

- 다음과 같은 구조로 Branch를 생성 후 Feature 작업을 한다.

```
Level 2 :         Dev -- Feature/login
        :             -- Feature/logout
Level 1 : Master
```

정기배포를 위한 GitFlow 전략

- Feature 개발 완료 후 배포를 하기 위해 Release Branch를 생성한다.

```
Level 2 :         Dev -- Feature/login     Release
        :             -- Feature/logout
Level 1 : Master
```

Release Branch 생성 후 추가 작업이 필요해질 경우

Release Branch 생성 후 추가 작업이 필요해질 경우

- Release Branch를 기준으로 추가 작업을 위한 Branch를 생성한다.

```
Level 3 :                                Release -- Feature/FindPW
Level 2 :         Dev -- Feature/login
        :             -- Feature/logout
Level 1 : Master
```

Release Branch 생성 후 추가 작업이 필요해질 경우

- 추가 작업이 끝나면
- Release Branch에 Merge 후
- Release Branch를 Master Branch에 Merge 한다.

```
Level 4 : Master <- - - - - - - - - - - - - -┐
Level 3 :                                    Release  -- Feature/FindPW
Level 2 :         Dev -- Feature/login
        :             -- Feature/logout
Level 1 :
```

Release Branch 생성 후 추가 작업이 필요해질 경우

- Master에 추가된 작업 내용을
- Dev Branch에 Master Branch를 Merge 하여 Master와 Dev의 Sync를 맞춘다.

```
Level 4 : Master  Dev
Level 3 :
Level 2 :
Level 1 :
```

Q. 왜 Master와 Dev 간 Sync를 맞춰야 할까?

Q. 왜 Master와 Dev 간 Sync를 맞춰야 할까?

- Dev는 Master를 Base로 생성된 Branch이다.
- 그러므로 **Dev = Master +** @의 코드를 갖고 있어야 한다.
- 만약 Sync가 안 맞을 경우엔 **Dev != Master +** @가 된다.
- 이런 상태로 누군가 Dev에서 신규 Branch를 생성하게 되면 코드가 꼬이게 된다.

Release Branch 생성 후 추가 작업이 없는 경우

Release Branch 생성 후 추가 작업이 없는 경우

- Release Branch를 Master Branch에 Merge 한다.

```
Level 3 : Master <- - - - - - - - - - - - - -┐
Level 2 :         Dev -- Feature/login    Release
        :             -- Feature/logout
Level 1 :
```

Release Branch 생성 후 추가 작업이 없는 경우

- Dev Branch에 Master Branch를 Merge 하여 Master와 Dev의 Sync를 맞춘다.

```
Level 3 : Master  Dev
Level 2 :
Level 1 :
```

Hotfix가 나가야 할 상황이라면

Hotfix가 나가야 할 상황이라면

- 다음과 같은 상황에서 운영에서 장애가 발생하여
- Hotfix로 이슈를 수정하여 배포가 나가야한다.

```
Level 2 :         Dev -- Feature/login
        :             -- Feature/logout
Level 1 : Master
```

Hotfix가 나가야 할 상황이라면

- Master Branch를 기준으로 Hotfix Branch를 생성한다.

```
Level 2 :                    Dev -- Feature/login
        :                        -- Feature/logout
Level 1 : Hotfix       Master
```

Hotfix가 나가야 할 상황이라면

- Hotfix Branch에 수정 작업을 진행한다.

```
Level 2   : Hotfix -- Feature/FindPW          Dev -- Feature/login
          :                                       -- Feature/logout
Level 1   :                           Master
```

Hotfix가 나가야 할 상황이라면

- Hotfix 작업이 끝나면 Master Branch에 Merge 한다.

```
Level 3 :    ┌- - - - - - - - - - - - -> Master
Level 2 : Hotfix -- Feature/FindPW              Dev -- Feature/login
        :                                           -- Feature/logout
Level 1 :
```

Hotfix가 나가야 할 상황이라면

- Dev Branch에 Master Branch를 Merge 하여 Master와 Dev의 Sync를 맞춘다.

```
Level 3 : Master  Dev
Level 2 :
Level 1 :
```

Hotfix가 나가야 할 상황이라면

- 이 후 신규 작업은 Dev Branch를 Base로
- 새로운 Feature Branch를 생성한다.

```
Level 4 :          Dev -- Feature/payment
        :              -- Feature/refund
Level 3 : Master
Level 2 :
Level 1 :
```

Q. Hotfix 이 후 왜 Master와 Dev 간 Sync를 맞춰야 할까?

- Master를 Dev에 Merge 하지 않고
- Dev에서 Feature Branch 생성하게 되면
- **최신 코드가 아닌 상태**로 개발이 진행된다.

반드시 알아야만 하는 개념들

2 Spring Multi Module 프로젝트

강의를 시작하기 앞서

사전 학습이 필요한 개념

- Spring Bean
- Spring Component Scan
- @RestControllerAdvice

DB 연동을 위한 환경 설정

- 프로젝트에서 사용할 DB 환경 설정 필요
- ex) MySQL, Oracle, H2 등등

강의에서 다룰 상세한 내용들

- API / Common 모듈 생성
- API 모듈은 Common 모듈의 존재하는 클래스를 참조 및 사용
- IT 회사에서 사용하는 Exception 핸들링 컨벤션
- Multi Module 구조에서 DB 연동
- Multi Module 구조에서 Gradle을 사용한 빌드 및 배포

Multi Module이란?

- 필요한 기능별로 Module을 생성한다.
- 레고를 조립하듯 필요한 Module을 조립한다.
- **N개의 Module이 조립**되어있는 프로젝트를 Multi Module 프로젝트라 부른다.
- ex) 로그인 Module, 인증 Module, DB 엔티티 Module 등등

Multi Module 왜 사용할까?

- 예를 들어
- API 서버에서도 DB Entity가 필요하고
- Batch 서버에서도 동일한 DB Entity가 필요하다면
- 중복된 Entity를 Module화 시켜 사용하기 위해 Multi Module 프로젝트를 사용한다.
- 만약 독립적으로 관리한다면
- 중복해서 관리해야하므로 Risk가 늘어난다.

Exception 핸들링

- 언어 혹은 프레임워크에서 발생한 Exception은 반드시 **Custom하게 Wrapping**하여 처리한다.
- @RestControllerAdvice 어노테이션을 사용하여
- 모든 예외를 해당 클래스에서 클라이언트와 사전에 정의한 값으로 재정의한다.
- ex) NPE(=Null Point Exception)일 경우 Error Code를 4001로 내린다.

Multi Module 구조에서 Gradle을 사용한 배포

- Multi Module 구조에서는 원하는 Module을 골라서 빌드&배포가 가능하다.
- 빌드 툴로는 Gradle 혹은 Maven을 사용하는데
- 요근래 생성되는 프로젝트는 대부분 Gradle을 사용한다.
- Gradle을 사용하여 빌드&배포를 하려면 Gradle 문법을 학습해야 한다.
- ex) ./gradlew clean :module-api:buildNeeded --stacktrace --info --refresh-dependencies -x test

실습

- 다음 순서로 실습 진행

1. API 모듈 생성
2. Common 모듈 생성
3. API 와 Common 모듈 의존성 설정
4. DB 연동

Multi Module 프로젝트 정리

1. settings.gradle 역할
2. 하위 Module에 대한 Bean Scan
3. Gradle을 사용한 빌드 및 동작 원리

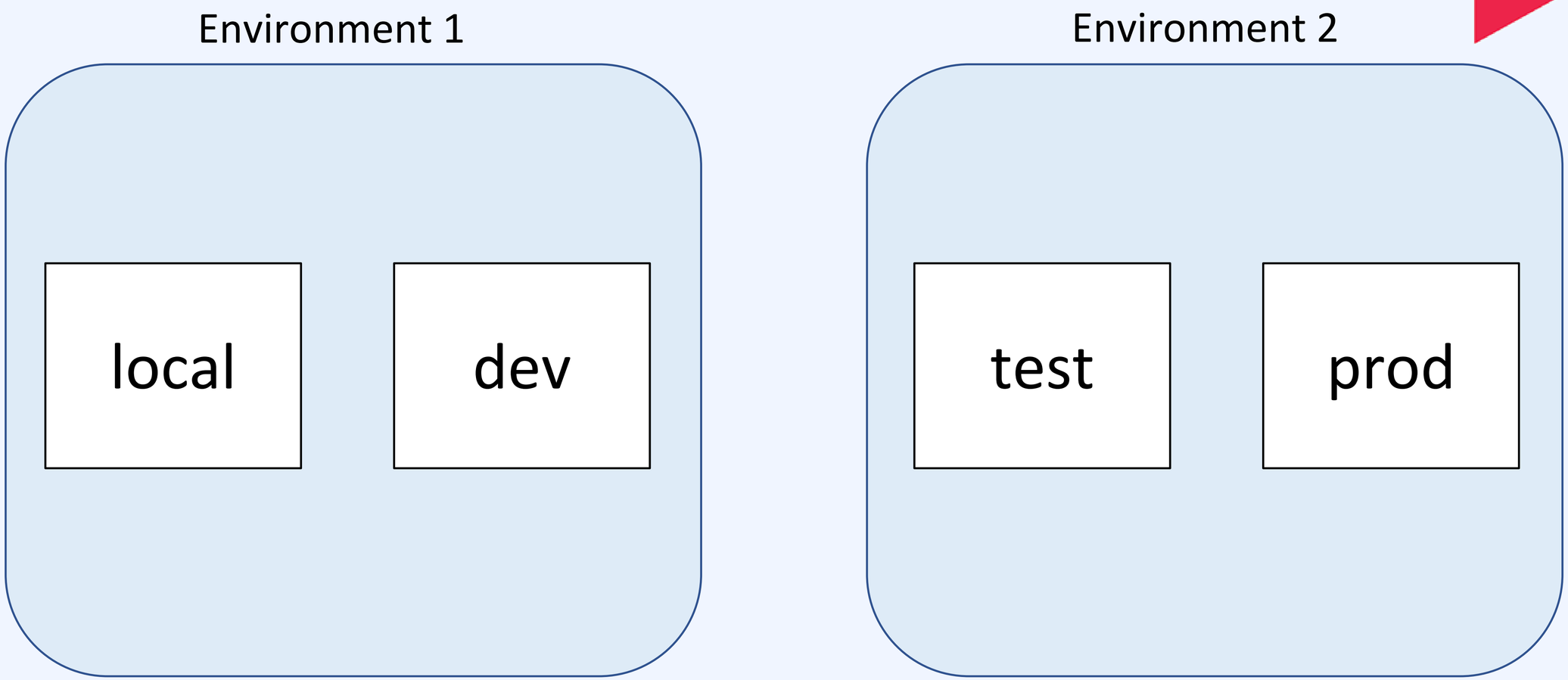
반드시 알아야만 하는 개념들

3 Profile

Profile이 필요한 이유

- 실제 회사에서 개발할 땐 N개의 Profile을 설정한다.
- ex) local, dev, test, prod
- 이렇게 Profile을 나누는 이유는 환경별로 설정해야 하는 Property 값들이 다르기 때문이다.

다양한 환경



Profile사용 방법

- 실제 프로젝트에서는 다음과 같은 파일로 환경별 Property를 구분한다.
- ex) application-{env}.yaml
- ex) application-local.yaml, application-pdev.yaml, application-prod

24. Externalized Configuration

Spring Boot lets you externalize your configuration so that you can work with the same application code in different environments. You can use properties files, YAML files, environment variables, and command-line arguments to externalize configuration. Property values can be injected directly into your beans by using the `@Value` annotation, accessed through Spring's `Environment` abstraction, or be bound to structured objects through `@ConfigurationProperties`.

Spring Boot uses a very particular `PropertySource` order that is designed to allow sensible overriding of values. Properties are considered in the following order:

1. Devtools global settings properties on your home directory (`~/.spring-boot-devtools.properties` when devtools is active).
2. `@TestPropertySource` annotations on your tests.
3. `properties` attribute on your tests. Available on `@SpringBootTest` and the test annotations for testing a particular slice of your application.
4. Command line arguments.
5. Properties from `SPRING_APPLICATION_JSON` (inline JSON embedded in an environment variable or system property).
6. `ServletConfig` init parameters.
7. `ServletContext` init parameters.
8. JNDI attributes from `java:comp/env`.
9. Java System properties (`System.getProperties()`).
10. OS environment variables.
11. A `RandomValuePropertySource` that has properties only in `random.*`.
12. Profile-specific application properties outside of your packaged jar (`application-{profile}.properties` and YAML variants).
13. Profile-specific application properties packaged inside your jar (`application-{profile}.properties` and YAML variants).
14. Application properties outside of your packaged jar (`application.properties` and YAML variants).
15. Application properties packaged inside your jar (`application.properties` and YAML variants).
16. `@PropertySource` annotations on your `@Configuration` classes.
17. Default properties (specified by setting `SpringApplication.setDefaultProperties`).

ref : https://docs.spring.io/spring-boot/docs/2.1.9.RELEASE/reference/html/boot-features-external-config.html

Profile 실습

1. 환경별 Property 값 사용
2. Intellij 실행 시 Profile 적용
3. JAR 실행 시 JVM 옵션으로 Profile 적용

Profile 정리

1. 환경별 Property 값 사용
2. JAR 실행 시 Environment 값 적용하여 배포