Spring 3.x & eGovFrame 3.0

삼성SDS 권 윤 정

Part1. Spring 3.x JavaConfig Annotation

Part2. eGovFrame 3.0 Migration Guide

Part1. Spring 3.x JavaConfig Annotation

STEP1. @Configuration 자바클래스 기본 설정

STEP2. Bean Definition Profiles

STEP3. PropertySource & Environment Abstract

STEP4. BootStrap @Configuration class
in web.xml & WebApplicationInitializer

STEP5. Spring Test Support Annotation

@Configuration	자바기반 설정 클래스 선언
@Bean	<bean /> 대체
@Import	@Configuration in @Configuration
@ImportResource	XML 설정파일 in @Configuration
@Value	Using property in javacode
@ComponentScan	<context:component-scan /> 대체
@EnableTransactionManagement	<tx:annotation-driven /> 대체
@PropertySource	Properties 파일을 PropertySource에 등록
@Profile	동일한 ID를 가지는 Bean을 환경별로 중복정의
@ContextConfiguration	테스트 케이스에서 설정정보 로딩
@ActiveProfiles	테스트 케이스에서 Profile 선택
@Autowired	동일한 타입의 빈을 주입받아 참조

STEP1. @Configuration 자바클래스 기본 설정

STEP1. Configuration Scenario

AppConfig
설정 클래스

```
<context:component-scan base-package="egovframework">
    <context:exclude-filter type="annotation”
        expression="org.sfwk.stereotype.Controller"/>
</context:component-scan>
```

```
<jdbc:embedded-database id="dataSource” type=“HSQL”>
    <jdbc:script location= "classpath:/db/ddl.sql"/>
    <jdbc:script location= "classpath:/db/dml.sql"/>
</jdbc:embedded-database>
```

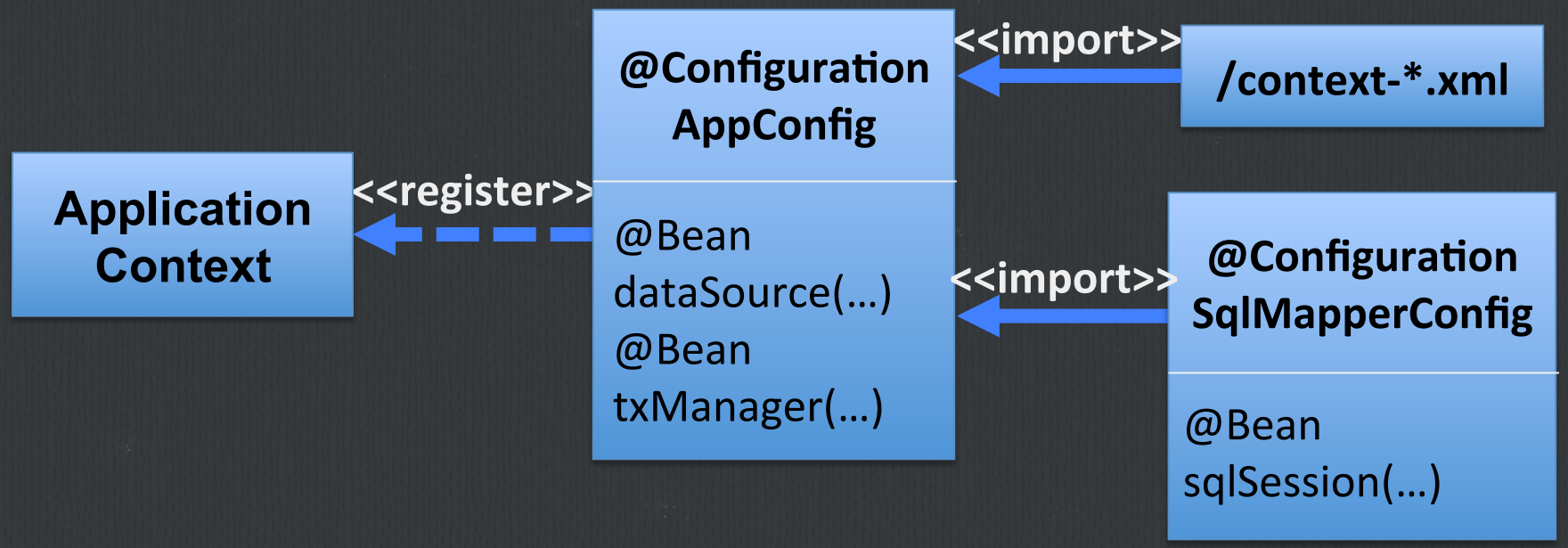
```
<bean id=”txManager”
    class="org.sfwk...DataSourceTransactionManager">
    <property name=“dataSource” ref=“dataSource” />
</bean>
<tx:annotation-driven />
```

STEP1. Configuration Scenario

SqlMapperConfig
설정 클래스

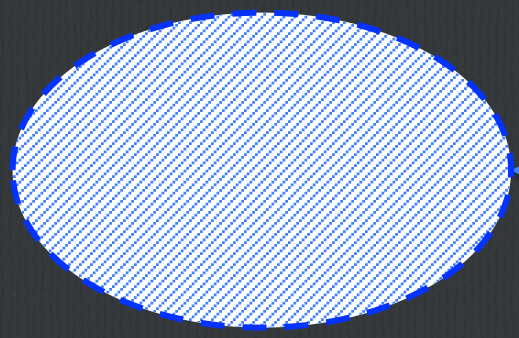
```
<bean id="sqlSession”
      class="org.mybatis.spring.SqlSessionFactoryBean">
      <property name="dataSource" ref="dataSource" />
      <property name="configLocation"
      value="classpath:/mybatis-config.xml" />
</bean>
```

STEP1. Configuration Scenario



STEP1-1. @Configuration AppConfig

```
@Configuration
@ImportResource("classpath:/spring/context-*.xml")
public class AppConfig {

    // other beans metadata ...
}
```



/spring
- /context-common.xml
- /context-datasource.xml
- /context-mybatis.xml
- ...

STEP1-1. @Configuration AppConfig

```
<context:component-scan base-package="egovframework">
    <context:exclude-filter type="annotation"
        expression="org.sfwk.stereotype.Controller"/>
</context:component-scan>
```

```
@ComponentScan(
    basePackages="egovframework",
    excludeFilters=
        @Filter(
            type=FilterType.ANNOTATION,
            value = Controller.class)
)
public class AppConfig {
    // bean metadata ...
}
```

```
@Target(value=TYPE)
@Retention(value=RUNTIME)
@Documented
@Component
public @interface Controller
public @interface Service
public @interface Repository
```

STEP1-1. @Configuration AppConfig

```
<jdbc:embedded-database id="dataSource" type="HSQL">
    <jdbc:script location= "classpath:/db/ddl.sql"/>
    <jdbc:script location= "classpath:/db/dml.sql"/>
</jdbc:embedded-database>
```

```
@Bean
public DataSource dataSource() {
    return new EmbeddedDatabaseBuilder()
        .setType(EmbeddedDatabaseType.HSQL)
        .addScript("classpath:/db/ddl.sql")
        .addScript("classpath:/db/dml.sql")
        .build();
}
```

STEP1-1. @Configuration AppConfig

```
<bean id="txManager"
      class="org.sfwk...DataSourceTransactionManager">
      <property name="dataSource" ref="dataSource" />
</bean>
```

```
@Bean
public PlatformTransactionManager txManager() {
      return new
            DataSourceTransactionManager(dataSource());
   }
}

@Bean
public DataSource dataSource() { ... }
```

STEP1-1. @Configuration AppConfig

<tx:annotation-driven />

```
@Service(”myService")
@Transactional
public class MyServiceImpl {
    ...
}

@EnableTransactionManagement
public class AppConfig {
    ...
}
```

STEP1-2. @Configuration SqlMapperConfig

```
<bean id="sqlSession”
      class="org.mybatis.spring.SqlSessionFactoryBean">
      <property name="dataSource" ref="dataSource" />
      <property name="configLocation"
            value="classpath:/mybatis-config.xml" />
</bean>
```

```
@Configuration
public class SqlMapperConfig {

      @Autowired
      DataSource dataSource;

      // sqlSession bean …
}
```

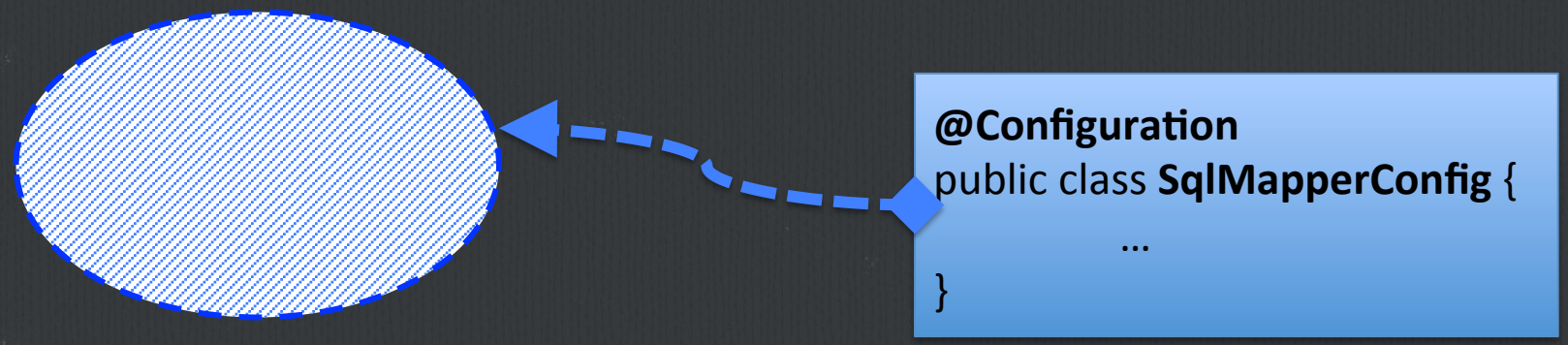
STEP1-2. @Configuration SqlMapperConfig

```
<bean id="sqlSession”
    class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="configLocation"
        value="classpath:/mybatis-config.xml" />
</bean>
```

```
@Bean
public SqlSessionFactoryBean sqlSession() {
    SqlSessionFactoryBean sfb =
        new SqlSessionFactoryBean();
    sfb.setDataSource(this.dataSource);
    sfb.setConfigLocation(
        new ClassPathResource(“/mybatis-config.xml"));
    return sfb;
}
```

STEP1-3. @Configuration 클래스간 관계 연결

```
@Configuration
@ImportResource("classpath:/spring/context-*.xml")
@ComponentScan(basePackages="egovframework",
    excludeFilters=@Filter(type=FilterType.ANNOTATION,
    value = Controller.class))
@Import(SqlMapperConfig.class)
public class AppConfig {

}
```



(결과) STEP1. AppConfig

```
@Configuration
@ImportResource(”…")
@ComponentScan(basePackages=”…”, excludeFilters=…)
@Import(SqlMapperConfig.class)
@EnableTransactionManagement
public class AppConfig {
    @Bean
    public DataSource dataSource() {
        return new EmbeddedDatabaseBuilder()….build();
    }
    @Bean
    public PlatformTransactionManager txManager() {
        return new DataSourceTransactionManager(dataSource());
    }
}
```

(결과) STEP1. SqlMapperConfig

```
@Configuration
public class SqlMapperConfig {

    @Autowired
    DataSource dataSource;

    @Bean
    public SqlSessionFactoryBean sqlSession() {
        SqlSessionFactoryBean sfb =
            new SqlSessionFactoryBean();
        sfb.setDataSource(this.dataSource);  ...
        return sfb;
    }
}
```

STEP2. Bean Definition Profiles

STEP2. Configuration Scenario

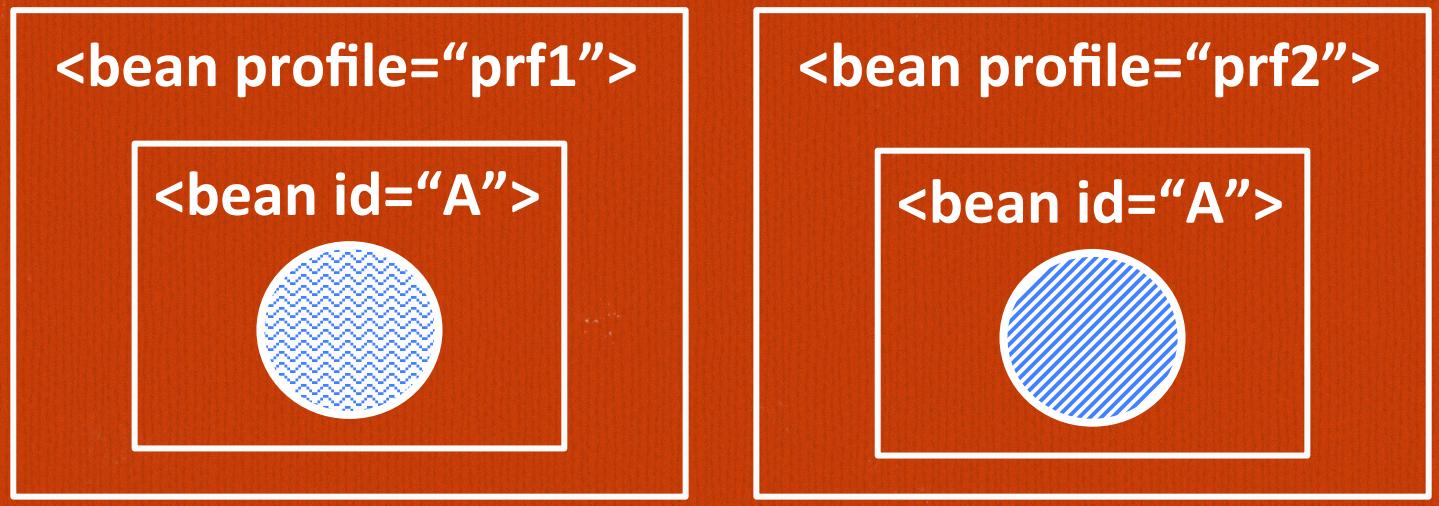
```
public class AppConfig {
    // @Bean
    // public DataSource dataSource() { // 테스트용
    //   return new EmbeddedDatabaseBuilder()
    //   ....build();
    // }
    @Bean
    public DataSource dataSource() { // 운영용
        BasicDataSource dataSource =
            new BasicDataSource(); ...
        return dataSource;
    }
}
```

둘 다 정의해놓고
자유롭게 골라쓸
수는 없을까?

> Bean Definition Profiles <

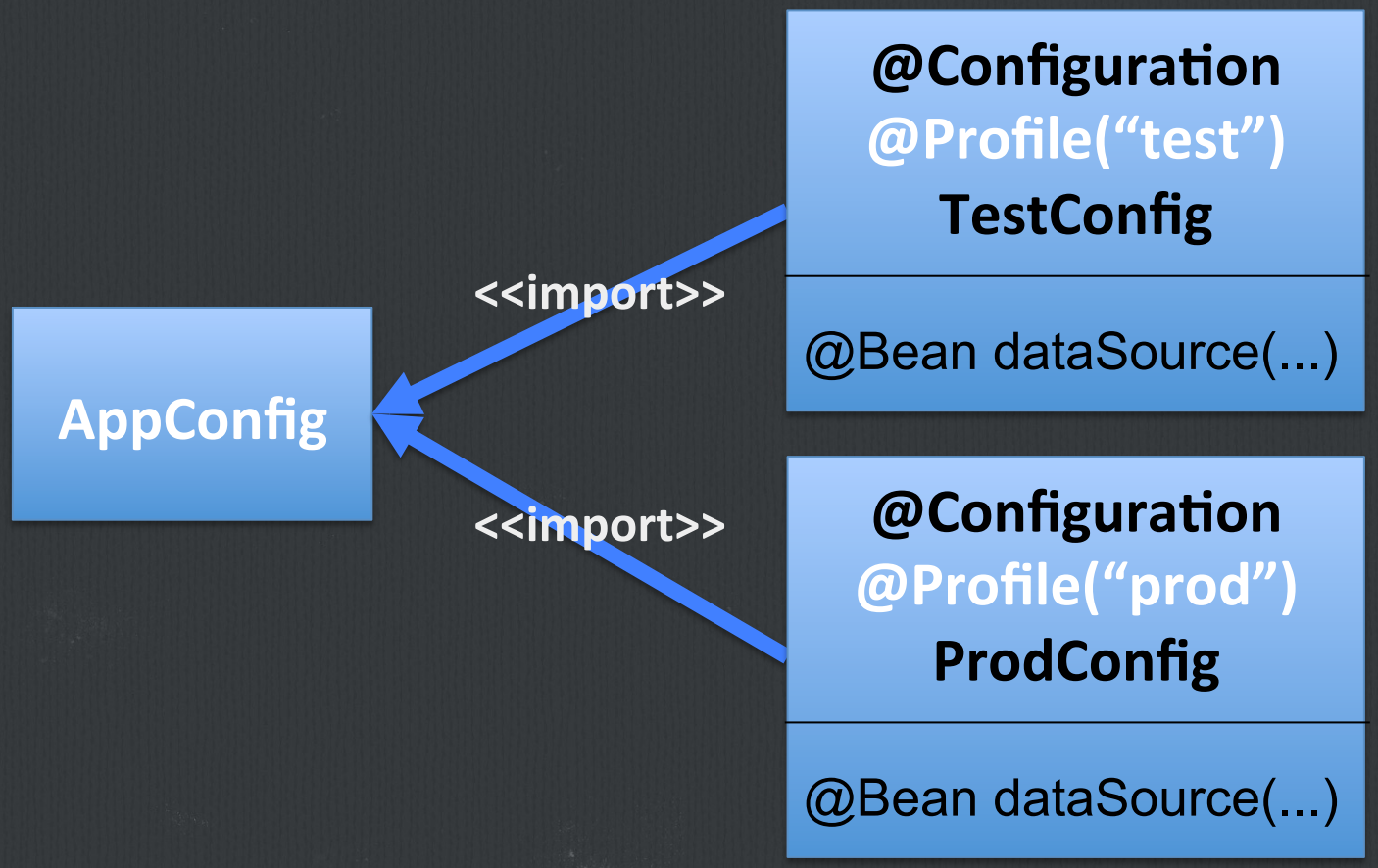
- **Profile** (dic.)

  - A **Profile** is A **subset internal to A specification.**
  - A **Specification have more than one *definition***, and there are probably many *optional* features.



STEP2. Configuration Scenario

- **Any Profile will be skipped or processed** based on which Spring profiles **are currently active.**



STEP2-1. @Profile TestConfig

```
@Configuration
@Profile("test")
public class TestConfig {
    @Bean
    public DataSource dataSource() {
        return new EmbeddedDatabaseBuilder()
            .setType(EmbeddedDatabaseType.HSQL)
            .addScript("classpath:/db/ddl.sql")
            .addScript("classpath:/db/dml.sql")
            .build();
    }
}
```

STEP2-2. @Profile ProdConfig

```
@Configuration
@Profile("prod")
public class ProdConfig {
    @Bean
    public DataSource dataSource() {
        BasicDataSource dataSource = new BasicDataSource();
        dataSource.setDriverClassName("com…Driver");
        dataSource.setUrl("jdbc:mysql://…");
        dataSource.setUsername("user");
        dataSource.setPassword("user01");
        return dataSource;
    }}
```

STEP2-3. @Configuration 클래스간 관계 연결

```
@Configuration
@Import({SqlMapperConfig.class,
    TestConfig, ProductionConfig})
@EnableTransactionManagement
public class AppConfig {
    @Autowired
    DataSource dataSource;
    @Bean
    public PlatformTransactionManager txManager() {
        return new DataSourceT..Manager(this.dataSource);
    }
}
```

STEP2-4. Profile Activation

- ‘spring.profiles.active’ property
  - servlet context parameters in web.xml
  - JVM system properties
  - Programmatically by coding
  - .properties file
  - @ActiveProfiles

STEP2-4. Profile Activation

- ‘spring.profiles.active’ property
  - **servlet context parameters in web.xml**

```
<init-param>
  <param-name>spring.profiles.active</param-name>
  <param-value>profilename</param-value>
</init-param>
```

```
DEBUG [org.springframework.core.env.PropertySourcesPropertyResolver] Found key 'spring.profiles.active' in [servletContextInitParams]
```

  - JVM system properties
  - Programmatically by coding
  - .properties file

STEP2-4. Profile Activation

- ‘spring.profiles.active’ property
  - servlet context parameters in web.xml
  - **JVM system properties**

Run Configuration > VM arguments:

-Dspring.profiles.active=profilename

```
DEBUG [org.springframework.core.env.PropertySourcesPropertyResolver] Found key 'spring.profiles.active' in [systemProperties]
```

  - Programmatically by coding
  - .properties file

STEP2-4. Profile Activation

- ‘spring.profiles.active’ property
  - servlet context parameters in web.xml
  - JVM system properties
  - **Programmatically by coding**

```
context.getEnvironment()
    .setActiveProfiles(“profilename”);
```

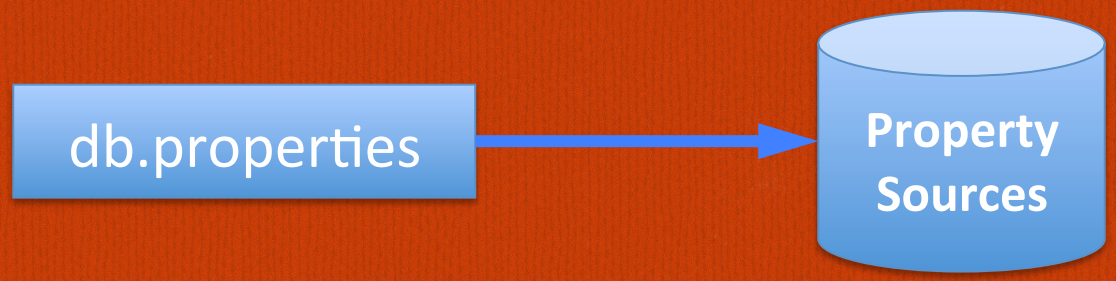
  - **.properties file**

```
spring.profiles.active=profilename
```

STEP3. PropertySource
& Environment Abstract

> PropertySource <

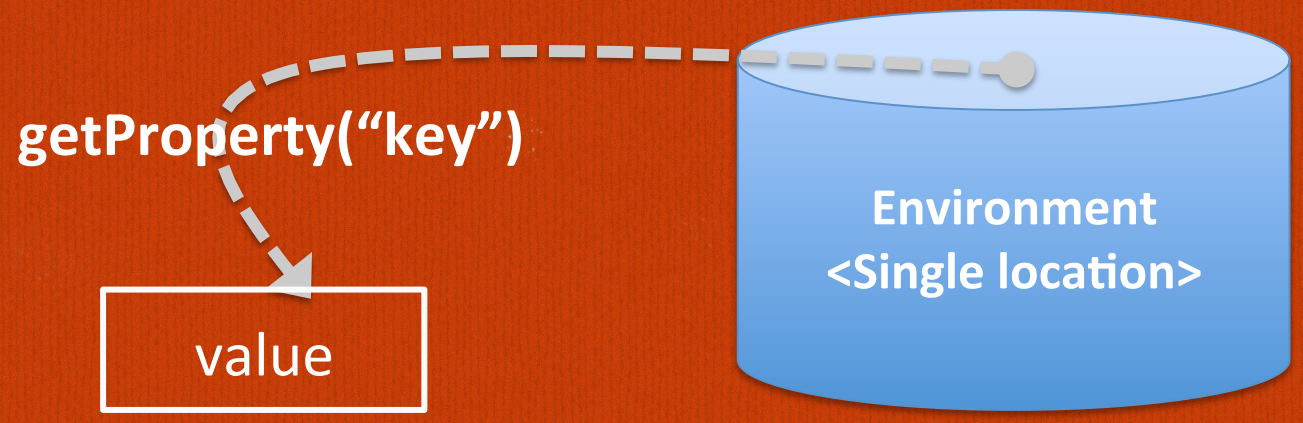
- **PropertySource**
  - A PropertySource is a simple abstraction over any source of key-value pairs.




```
<context:property-placeholder
    location="classpath:/db.properties" />

@PropertySource("classpath:/db.properties")
```

> Environment Abstraction <

- Spring's Environment abstraction <u>provides a single location</u> to configure both *profiles* and *properties*.
- Under the abstraction, all system properties, environment variables, and application properties are served by the Environment interface,



STEP3-1. @PropertySource 추가/추출 (1)

```
@Configuration
@PropertySource(value = "classpath:/db.properties”)
public class ProductionConfig {
    @Autowired Environment env;

    @Bean
    public DataSource dataSource() {
        BasicDataSource ds= new BasicDataSource();
        …
        ds.setUrl(env.getProperty("db.url"));
        ds.setUsername(env.getProperty("db.username"));
        …
        return dataSource;
    }
}
```

STEP3-1. @PropertySource 추가/추출 (2)

```
@Configuration
@PropertySource(value = "classpath:/db.properties”)
public class ProductionConfig {
    @Value("${db.url}") private String url;
    @Value("${db.username}") private String username;
    ...
    @Bean
    public DataSource dataSource() {
        BasicDataSource ds= new BasicDataSource();
        ...
        ds.setUrl(url);  ds.setUsername(username);
        ...
        return dataSource;
    }
}
```

STEP3-1. @PropertySource 추가/추출 (2)

- **@Value**로 프로퍼티값을 가져오려면.. 추가설정 필요

```
// 선택1. 아래 빈 설정 추가
@Bean
public static PropertySourcesPlaceholderConfigurer
propertyPlaceholder() {
return new PropertySourcesPlaceholderConfigurer();
}
```

```
// 선택2. XML설정 파일에 아래 태그 선언
<context:property-placeholder />
```

STEP4. BootStrap @Configuration class in web.xml & WebApplicationInitializer

STEP4. BootStrap @Configuration in web.xml

```
<context-param>
    <param-name>contextClass</param-name>
    <param-value>
        o.s.w.c.s.AnnotationConfigWebApplicationContext
    </param-value>
</context-param>
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
        egovframework.sample.config.AppConfig
    </param-value>
</context-param>
```

STEP4. BootStrap @Configuration in WebApplicationInitializer

- WebApplicationInitializer
  - 서블릿 3.0 기반 ServletContainerInitializer
  - SpringServletContainerInitializer

```
@Override
public void onStartup(ServletContext container) {
    AnnotationConfigWebApplicationContext ctx =
        new AnnotationConfigWebApplicationContext();
    ctx.register(AppConfig.class);
    container.addListener(
        new ContextLoaderListener(ctx));
}
```

STEP5. Spring Test Support Annotation

STEP5. Spring Test Support Annotation

- @RunWith(SpringJUnit4ClassRunner.class)
- @ContextConfiguration
  - classes, locations
- @ActiveProfiles
- @Before, @After
- @TransactionConfiguration, @Rollback 등
- @Autowired, @PersistenceContext 등

STEP5. 테스트 코드

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes={AppConfig.class})
@ActiveProfiles("test")
public class ActiveProfilesTest {
    @Autowired  DefaultListableBeanFactory bf;
    @Test
    public void testMethod() {
        for (String name : bf.getBeanDefinitionNames()) {
            if(name.contains("Config”))
                System.out.println(name + ”/”);
        }
    } // 결과: appConfig / sqlMapperConfig / testConfig /
}
```

Part2. eGovFrame 3.0 Migration Guide

- @CommandMap
- DB기반 PropertySource
- SLF4J Logging

실행환경 Migration 가이드 (2.X -> 3.0)

1. 실행환경 library 변경 (maven dependency 수정)
2. Generics 적용에 따른 casting 적용
3. Spring Security 업그레이드 반영
4. 명명규칙 변경 (AbstractServiceImpl -> EgovAbstractServiceImpl)
5. MultiPart 관련 API 변경
6. Excel Service interface 변경
7. **로그서비스 방식 변경**
8. **HandlerMapping 정의클래스 변경**
9. **controller에서 request값을 Map으로 받는 방법) 사용 변경**
10. quartz 관련 TriggerBean 변경
11. 실행환경 JDK 1.6 이상 (eGov개발환경 케플러 버전은 JDK 1.8 X)

eGovFrame @Deprecated

- CommandMapArgumentResolver (2.7)
  → AnnotationCommandMapArgumentResolver (3.0)
- AbstractServiceImpl (2.7)
  → EgovAbstractServiceImpl (3.0)
- EgovJDBCAppender (log4j 1.x)
  → EgovConnectionFactory (log4j 2)
- SimpleUrlHandlerMapping (2.7)
  → <mvc:interceptors> <mvc:exclude-mapping /> (spring 3.2)

Spring @Deprecated

- DefaultAnnotationHandlerMapping (3.0)
  → RequestMappingHandlerMapping (3.1)
- AnnotationMethodHandlerAdapter (3.0)
  → RequestMappingHandlerAdapter (3.1)
- WebArgumentResolver (3.0)
  → HandlerMethodArgumentResolver (3.1)

@CommandMap

eGovFrame 2.7 ArgumentResolver 서비스

- CommandMapArgumentResolver
  - Spring 3.0, WebArgumentResolver 구현체
  - 요청 파라미터값들을 Map 객체에 담아 컨트롤러 메소드 파라미터(commpandMap)에 바인딩

```
public String hello(Map commandMap,...) {
}
```

  - AnnotationMethodHandlerAdapter에 등록

```
<bean
class="o.s.w.s.m.a.AnnotationMethodHandlerAdapter">
    <property name="customArgumentResolver">
        <bean class="e..CommandMapArgumentResolver" />
...
```

eGovFrame 3.0 ArgumentResolver 서비스

- AnnotationCommandMapArgumentResolver
  - Spring 3.1, HandlerMethodArgumentResolver 구현체
  - 요청 파라미터값들을 Map 객체에 담아 컨트롤러 메소드 파라미터(@CommandMap)에 바인딩

```
public String hello(@CommandMap Map map,...) {
}
```

  - RequestMappingHandlerAdapter에 등록?? NO!!

    → EgovRequestMappingHandlerAdapter에 등록

    → Why?

eGovFrame 3.0 ArgumentResolver 서비스

- EgovRequestMappingHandlerAdapter에 등록

```
<bean
    class="e.r.p.m.b.a.EgovRequestMappingHandlerAdapter">
    <property name="customArgumentResolver">
        <bean class=
        “e..AnnotationCommandMapArgumentResolver” />
    </property>
</bean>
```

- 주의사항
  - <mvc:annotation-driven /> 선언X

DB기반 PropertySource

기존 Property 서비스들

- In Spring
  - Using placeholder-${}

```
<context:property-placeholder
        location=”xxx.properties"/>
<jdbc:embedded-database id="dataSource”>
        <jdbc:script location= "${Globals.hsqlDB}"/>
</jdbc:embedded-database>
```

  - <util:properties>
    - ${}
    - @Value(“key”)

기존 Property 서비스들

- In eGovFrame 2.7
  - EgovPropertyServiceImpl
    - 외부에 properties파일을 두어 운영 중에도 refresh가능

    (refreshPropertyFiles() 호출)

```
<bean name="propertyService"
      class="e.r.f.p.i.EgovPropertyServiceImpl>
      <property name="extFileName">
            …<entry key="fileName" value="file:…" />
            …
      </property>
</bean>
```

```
String value = propertyService.getString("key");
```

DB기반 PropertySoruce 서비스

- In eGovFrame 3.0
  - DBPropertySourceInitializer
    - ApplicationContextInitializer initialize() 메소드
    - ApplicationContext 로딩시점에 특정 DB 테이블에서 읽어들인 프로퍼티 정보(key, value)를 PropertySource에 등록하도록 구현
    - web.xml 과 bean 설정 필요

DB기반 PropertySoruce 서비스

- web.xml 설정

```
<context-param>
  <param-name>contextInitializerClasses</param-name>
  <param-value>e.r.f....DBPropertySourceInitializer</param-value>
</context-param>

<context-param>
  <param-name>propertySourceConfigLocation</param-name>
  <param-value>classpath:/propertysource.xml</param-value>
 </context-param>
```

DB기반 PropertySoruce 서비스

- propertysource.xml 설정 (DbPropertySource 빈 정의)

```
<bean id="dbPropertySource"
    class=”e.r.f.p.d.DbPropertySource”>
    <constructor-arg ref="dataSource"/>
    <constructor-arg
        value="SELECT PKEY, PVALUE FROM PROPERTY"/>
    <constructor-arg value="dbPropertySource"/>
</bean>
```

- DB 테이블 생성 (PROPERTY(PKEY, PVALUE))

DB기반 PropertySoruce 서비스

- 사용법
  - <context:property-placeholder />
    - ${key} in XML
    - @Value(“key”) in Java
  - @Autowired Environment env
    - getProperty(“key”);

SLF4J Logging

기존 Logging 방식

- Commons Logging (JCL)

```
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

Log logger = LogFactory.getLog();
```

- Log4j 1.x

```
import org.apache.log4j.Logger;

Logger logger = Logger.getLogger();
```

SLF4J Logging 방식

- SLF4J 메서드
  - Logging 방식 통일
  - Log4j, Logback, JCL 등 다양한 구현체 사용 가능

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

Logger egovLogger = LoggerFactory.getLogger(...);
```

SLF4J Logging 방식

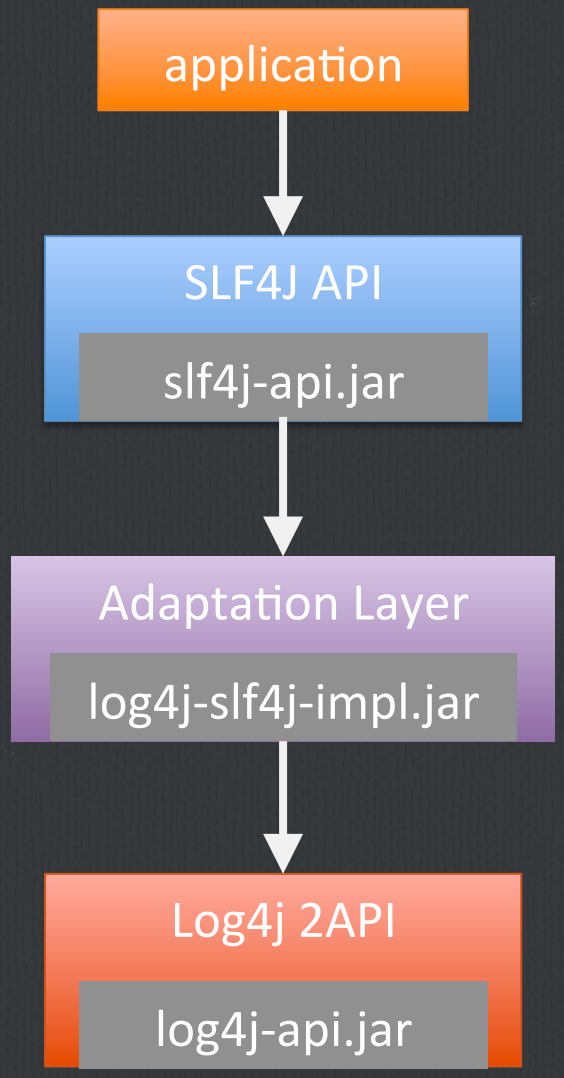
- Parameterized Logging 지원

```
String message = "Hello";
String message2 = "eGovFrame 3.0";

logger.debug("{}!!! {}", message, message2);
// 출력결과 - Hello!!! eGovFrame 3.0

Object[] args = new Object[];
args[0] = "1";
args[] = new Date().toString();

logger.debug("{}, {}", args);
// 출력결과 - 1, Fri Mar 23 11:08:28 KST 2014
```

SLF4J Logging 방식

- SLF4J Logging을 위한 설정
  - slf4j-api-x.x.x.jar
  - Logging 구현체 Jar + XML설정파일
    ex) log4j2-api.jar, log4j2.xml
  - Logging 처리 위임을 위한 Binding Jar

Logging 구현체별	Binding Jar
Log4j 2	log4j-slf4j-impl.jar
Log4j 1.2	slf4j-log4j12.jar
JDK 1.x Logging	slf4j-jdk14.jar
Logback	logback-class.jar, logback-core.jar
JCL	slfj-jcl.jar
NOP	slf4j-nop.jar

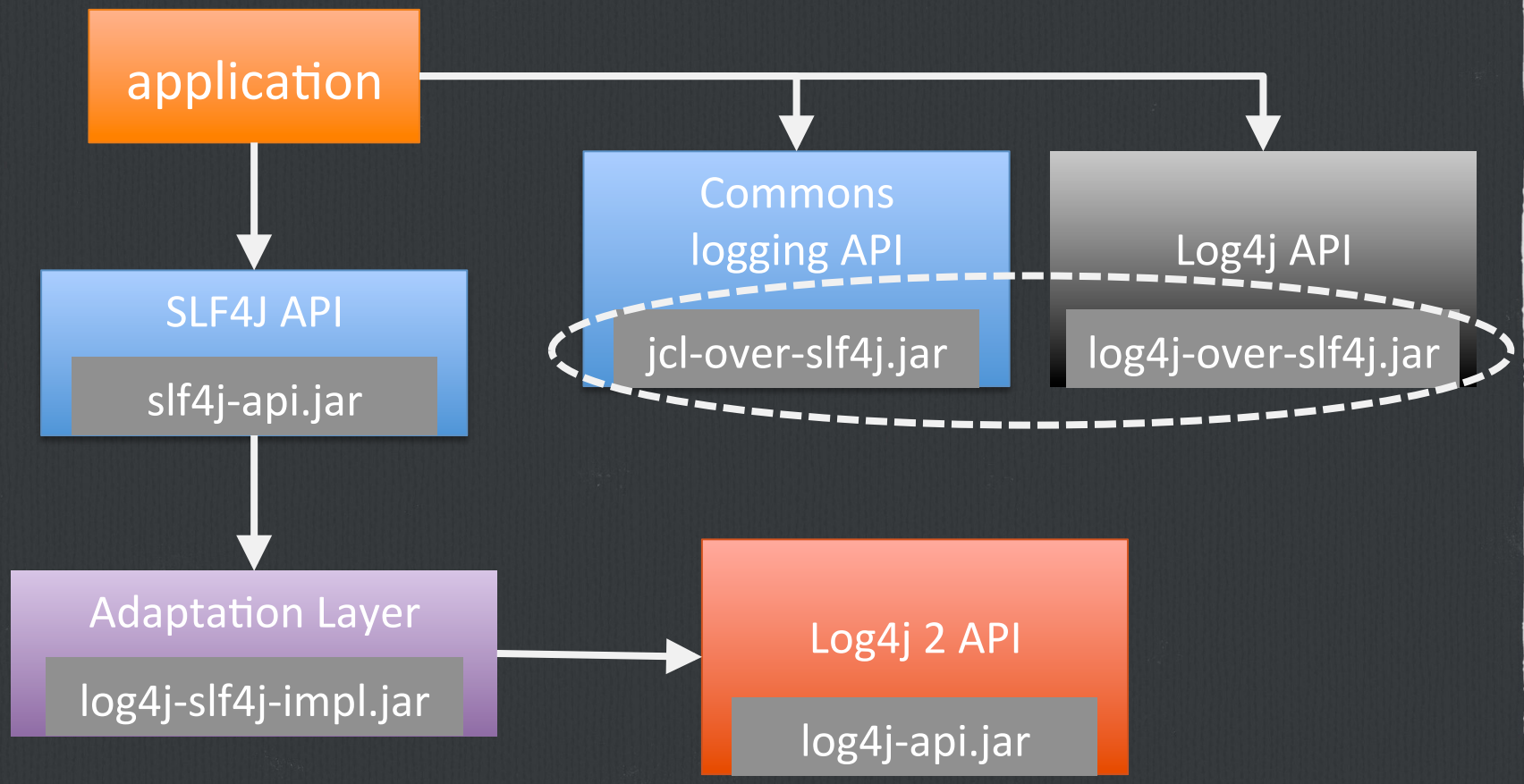


SLF4J Logging 방식

```
<!-- Slf4j API -->
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
        <version>x.x.x</version>
    </dependency>
<!-- Log4j2-SLF4J Bridge -->
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
        <artifactId>log4j-slf4j-impl</artifactId>
        <version>2.0</version>
    </dependency>
<!-- Log4j2 Core -->
    <dependency>
        <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
        <version>2.0</version>
    </dependency>
```

SLF4J Logging 방식

- SLF4J Bridge Jar
  - Legacy Code를 위해 SLF4J가 제공하는 jar



SLF4J Logging 방식

```
<!-- Commons-SLF4J Bridge -->
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>jcl-over-slf4j</artifactId>
        <version>x.x.x</version>
    </dependency>

<!– log4j-SLF4J Bridge -->
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>log4j-over-slf4j</artifactId>
        <version>x.x.x</version>
    </dependency>
```

SLF4J Logging 방식

- spring 모듈

```
<!-- Exclude Commons Logging in favor of SLF4j -->
    <exclusion>
        <groupId>commons-logging</groupId>
        <artifactId>commons-logging</artifactId>
    </exclusion>
```

참고문헌

- 표준프레임워크 3.0 실행환경 가이드
- 토비스프링 3.1
- Pro Spring MVC: with Web Flow

Thank you.