

스프링 기반의 공통 모듈 기본 사용법

1. 컨트롤러와 서비스의 역할과 책임
2. 스프링 기반의 에러 핸들링
3. 로그 출력과 예외 처리 AOP(관점 지향 프로그래밍) 사용
4. JPA CRUD 기능 사용
5. KeyValue(IMDG) CRUD 기능 사용
6. MyBatis CRUD 기능 사용
7. 스프링 시큐리티와 JWT 기반의 API 인증/인가 사용
8. 이메일 발송
9. (비)동기 HTTP Client 사용
10. 스웨거(REST API 테스트 화면) 기본 및 권한 테스트
11. 스케줄러 이중화(Active-Standby) 구성
12. 테스트 코드 작성

1. 컨트롤러와 서비스의 역할과 책임 1/2

```
@Operation(summary = "복구 API", description = "백업한 데이터를 반환하는 API")
@ResultLogging(result = true)
@PostMapping("restore")
public Map<String, Object> restore(
    @io.swagger.v3.oas.annotations.parameters.RequestBody(description = "복구 요청문(uniqueId)")
    @RequestBody String reqMsg) {
    Map<String, Object> mapResult = JsonUtil.readValue(reqMsg, new TypeReference<Map<String, Object>>() {
    });
    String uniqueId = (String) mapResult.get("uniqueId");
    if (StringUtils.isEmpty(uniqueId)) {
        throw new CommonException(CommonError.COM_EMPTY_INPUT_DATA, "uniqueId: " + uniqueId);
    }

    DidDocument didDoc = service.restore(uniqueId);

    return ImmutableMap.of("did", didDoc.getDid());
}
```

- 서비스의 역할

1. 비즈니스 로직 (에러 시 Exception 발생)

- 서비스 메소드 작성 가이드

1. 재사용 가능한 단위로 분할 (주석 필수)
2. 범용적이고 구체적인 입출력 객체를 정의 (무분별한 String/Collection 사용 자제)
3. 가능한 입력 인자는 5개가 넘지 않도록
4. 가능한 한개의 메소드가 100 라인이 넘지 않도록
5. 가능한 checked Exception은 RuntimeException으로 변환하여 에러 핸들링을 내부에서 처리

- 컨트롤러의 역할

1. 요청 데이터 검증 (에러 시 4xx 응답)
2. 서비스 호출 (에러 시 5xx 응답)
3. 응답 데이터 생성

- 컨트롤러 메소드 작성 가이드

1. RestController는 JSON 문자열로 구지 변환하지 않아도 됨
2. 클라이언트에서 성공 여부는 응답 데이터가 아닌 HTTP 상태 코드로 결정
3. HTTP 상태 코드가 매핑된 Error나 Exception을 사용
4. 요청 메소드당 테스트 코드 작성 권고

```
public DidDocument restore(String uniqueId) {
    // MagicPDS 서버 준비
    MagicPDS magicPDS = new MagicPDS(
        pdsProperties.getPdsUrl(), pdsProperties.isSSL(), pdsProperties.getAuthToken(), pdsProperties.getPw(),
        pdsProperties.getAuthType(), pdsProperties.getAuthId(), pdsProperties.getCompanyId());

    String result;
    try {
        // 전송 데이터는 base64로 인코딩하여 전송한다.
        result = magicPDS.sendGet(uniqueId, "didDoc");

        // 데이터 원본을 확인하기 위해서는, base64로 디코딩하여 출력해야한다. sendGet 메소드에는 디코딩하여 반환한다.
        log.debug("magicPDS에 저장한 데이터 확인: {}", result);
    } catch (Exception e) {
        throw new VcaException(VpsError.VPS_MAGICPDS_EXCEPTION, e);
    }

    return DidUtil.deserialize(result, DidDocument.class);
}
```

1. 컨트롤러와 서비스의 역할과 책임 2/2

<참고>

- 다양한 웹 강좌 (HTML, CSS, JAVASCRIPT, SQL, JAVA, BOOTSTRAP, REACTR, JQUERY 등) <https://www.w3schools.com>
- 개발자를 위한 웹 기술 (HTML, CSS, JAVASCRIPT 등) <https://developer.mozilla.org/ko/docs/Web>
- 개발자를 위한 웹 기술 > HTTP > HTTP 상태 코드 <https://developer.mozilla.org/ko/docs/Web/HTTP/Status>
- REST API 이해와 설계 - #1 개념 잡기 <http://bcho.tistory.com/953>
- REST API 이해와 설계 - #2 디자인 가이드 <http://bcho.tistory.com/954>
- REST API 이해와 설계 - #3 보안 가이드 <http://bcho.tistory.com/955>
- Spring Boot Reference (영문) <https://docs.spring.io/spring-boot/docs/2.7.12/reference/htmlsingle>
- Spring Doc – Swagger (영문) <https://springdoc.org/#springdoc-oUserpenapi-core-properties>
- Hazelcast Configuration (영문) <https://docs.hazelcast.com/imdg/4.1/configuration/configuring-declaratively>
- Hazelcast xml Config <https://github.com/hazelcast/hazelcast/blob/master/hazelcast/src/main/resources/hazelcast-config-4.1.xsd>
- 객체지향 프로그래밍의 5가지 설계 원칙(SOLID) : <https://mangkyu.tistory.com/194>

2. 스프링 기반의 에러 핸들링

```
/**
 * 기본 컨트롤러 예외 핸들러
 * <pre>
 * - @ControllerAdvice를 사용하지 않을 경우
 *   {@link org.springframework.web.servlet.mvc.support.DefaultHandlerExceptionResolver}가 기본으로 사용
 * </pre>
 */
@Slf4j
@Order(100)
@ControllerAdvice
public class ControllerExceptionHandler
    extends ResponseEntityExceptionHandler {
    @Override
    protected ResponseEntity<Object> handleExceptionInternal(Exception ex, Object body, HttpHeaders headers,
        HttpStatus status, WebRequest request) {

    }

    @ExceptionHandler
    private Object handleException2(Exception e, WebRequest request) {
        return ofResponse(e, request);
    }
}
```

- 에러 처리

1. "에러 코드, 에러 메시지, HTTP 상태 코드" 로 구성
2. 1개의 DidException 에서 n개의 DidError, BrokerError 등의 유형을 처리 가능
3. 실패 시 응답메시지 예시
 - HTTP Status Code: [400] BAD_REQUEST => [HTTP상태코드] HTTP상태명
 - HTTP Body: {
"error": "DID-10-06", => "에러유형-중분류번호-소분류번호"
"message": "[didm][DID_FAIL_VERIFICATION_CHALLENGE]챌린지 검증을 실패하였습니다. server:
null client: 3HVFPp7aSHJb6kj8S1US8Dx3z5n1" => "[발생서버][에러식별자]에러메세지"
}

- @ControllerAdvice

1. AOP를 사용하여 컨트롤러에서 발생한 예외가 한곳으로 모이도록 한다. (모든 컨트롤러를 catch 한 효과)
2. handleExceptionInternal() 메소드에서는 일반 예외를 처리하고, handleException2() 메소드에서는 사용자 예외를 처리한다.
3. 스프링은 이 어노테이션이 없으면 DefaultHandlerExceptionResolver 에 정의된 규칙을 따른다.
4. 만약 예외 규칙보다 우선순위를 더 높여서 실행하려면 @Order의 번호를 더 낮게 클래스를 만들어 준다. (비권장)

```
/**
 * DID 예외
 */
public class CommonException
    extends DefaultException {
}

/**
 * DID 에러
 */
@Getter
@AllArgsConstructor(access = AccessLevel.PRIVATE)
public enum CommonError
    implements Error {
    DID_EMPTY_DID_DOMAIN(DidError.CODE_PREFIX + "10-01", "DID 도메인이 비어있습니다.",
        HttpStatus.BAD_REQUEST),
}
```

3. 로그 출력과 예외 처리 AOP(관점 지향 프로그래밍) 사용 1/2

- 로그 출력

```
/**
 * 대상 메소드의 입출력 값과 소요시간을 로그에 출력
 */
@Documented
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface ResultLogging {
    /**
     * 클래스명 앞에 그룹핑할 타이틀명을 지정 (기본값: 없음)
     */
    String value() default "";

    /**
     * 입력 인자값을 로그에 출력할지 여부를 지정 (기본값: true)
     * 주의) 데이터가 너무 크면 성능에 영향을 준다
     */
    boolean args() default true;

    /**
     * 입력 인자값 중에 출력할 대상을 선택 (기본값: 없음)
     * 예) 1, 2번째 값만 출력할 경우: indexesOfArgs = {0, 1}
     */
    int[] indexesOfArgs() default {};

    /**
     * 출력 결과값을 로그에 출력할지 여부를 지정 (기본값: false)
     * 주의) 데이터가 너무 크면 성능에 영향을 준다
     */
    boolean result() default false;

    /**
     * 입출력 값을 JSON 형태로 출력할지 여부를 지정 (기본값: false)
     */
    boolean json() default false;

    /**
     * {@link com.fasterxml.jackson.annotation.JsonView} 로 정의한 필드가 있는 모델의 경우 사용할 뷰 모델을 지정
     */
    Class<? extends CommonModel.None> jsonView() default CommonModel.None.class;

    /**
     * 출력할 로그 레벨을 지정
     * (ERROR/WARN/INFO/DEBUG/TRACE, 기본값: DEBUG)
     */
    AopConfig.LogLevel logLevel() default AopConfig.LogLevel.DEBUG;
}
```

- 예외 처리

```
/**
 * 대상 메소드 안에서 발생하는 예외를 처리할 방법 정의
 */
@Documented
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface ExceptionHandling {
    /**
     * 입력 인자값 중에 출력할 대상을 선택 (기본값: 없음)
     * 예) 1, 2번째 값만 출력할 경우: indexesOfArgs = {0, 1}
     */
    int[] indexesOfArgs() default {};

    /**
     * 입출력 값을 JSON 형태로 출력할지 여부를 지정 (기본값: false)
     */
    boolean json() default false;

    /**
     * 예외 발생시 메시지에 입력 인자값을 포함할지 여부를 지정 (기본값: false)
     */
    boolean argsInException() default false;

    /**
     * 메소드에서 발생하는 모든 예외 중에 잡아서 처리할 예외 유형을 지정 (기본값: Exception.class)
     */
    Class<? extends Exception>[] catchTypes() default {Exception.class};

    /**
     * 지정한 예외 유형을 찾을 경우 로그에 출력할 로그 레벨을 지정
     * (THROW/SHORT/ERROR/WARN/INFO/DEBUG/TRACE, 기본값: ERROR)
     */
    LogLevel logLevel() default LogLevel.ERROR;

    /**
     * 지정한 예외 유형을 찾을 경우 로그에 출력하고 다시 예외를 상위로 던짐 (기본값: false)
     */
    boolean throwException() default false;

    /**
     * 지정한 예외 유형을 찾을 경우 결과값을 SPEL 표현식으로 지정
     * (기본값: {@link Defaults#defaultValue(Class)})
     */
    String returnExpression() default "";
}
```

3. 로그 출력과 예외 처리 AOP(관점 지향 프로그래밍) 사용 2/2

: @Transactional, @Cacheable, @Retryable 등 많은 스프링 어노테이션이 AOP 로 구현 - 테스트 코드

```
2023-07-14 15:31:18,008 WARN [main] around(AopConfig.java:279) ArithmeticException: / by zero args: {test} => ExceptionHandling 결과
java.lang.ArithmeticException: / by zero args: {test}
    at com.dreamsecurity.common.config.AopConfigTest$AopTest.test01(AopConfigTest.java:45)
    at com.dreamsecurity.common.config.AopConfigTest$AopTest$$FastClassBySpringCGLIB$$28b7e985.invoke(<generated>)
    at org.springframework.cglib.proxy.MethodProxy.invoke(MethodProxy.java:218)
    ...
```

```
2023-07-14 15:31:18,016 DEBUG [main] around(AopConfig.java:183) test.AopTest.test01(double,String) is execution time: 21 ms(00:00:00.021) args: {test} result: -1 => ResultLogging 결과
```

```
2023-07-14 15:31:18,023 DEBUG [main] around(AopConfig.java:282) RuntimeException: 1 Caused by: ArithmeticException: 4 => ExceptionHandling - SHORT 결과
```

```
2023-07-14 15:31:18,025 ERROR [main] apply(AopConfigTest.java:26) RuntimeException: 1 Caused by: ArithmeticException: 4 => ExceptionHandling - throwException 결과
```

```
java.lang.RuntimeException: 1
    at com.dreamsecurity.common.config.AopConfigTest$AopTest.test02(AopConfigTest.java:52)
    at com.dreamsecurity.common.config.AopConfigTest$AopTest$$FastClassBySpringCGLIB$$28b7e985.invoke(<generated>)
    at org.springframework.cglib.proxy.MethodProxy.invoke(MethodProxy.java:218)
    ...
```

```
Caused by: com.dreamsecurity.common.exception.CommonException: [app] 2
```

```
... 105 common frames omitted
```

```
Caused by: com.dreamsecurity.common.exception.CommonException: [app] 3
```

```
... 105 common frames omitted
```

```
Caused by: java.lang.ArithmeticException: 4
```

```
... 105 common frames omitted
```

```
2023-07-14 15:31:18,023 DEBUG [main] around(AopConfig.java:171) test.AopTest.test02(double) is execution message: RuntimeException: 1 Caused by: ArithmeticException: 4 args: {1.0} => ResultLogging 결과
```

```
@ResultLogging(value = "test", indexesOfArgs = 1, result = true)
@ExceptionHandler(indexesOfArgs = 1, argsInException = true,
    logLevel = AopConfig.LogLevel.WARN, returnExpression = "-1")
public int test01(double d, String s) {
    return 1 / 0;
}

@ResultLogging(value = "test", result = true)
@ExceptionHandler(catchTypes = CommonException.class,
    logLevel = AopConfig.LogLevel.SHORT, throwException = true)
public int test02(double d) {
    throw new RuntimeException("1", new CommonException("2",
        new CommonException("3", new ArithmeticException("4"))));
}
```

4. JPA CRUD 기능 사용 – Entity 1/4

```
/**
 * 샘플
 */
@Schema(description = Sample.TABLE_DESC)
@Getter
@Setter
@EqualsAndHashCode(callSuper = true)
@SuperBuilder
@NoArgsConstructor
@Entity
@DynamicInsert
@DynamicUpdate
@Table(indexes = {
    @Index(name = "idx_sample_name", columnList = "name"),
    @Index(name = "idx_sample_state", columnList = "state"),
    @Index(name = "idx_sample_reg_date", columnList = "regDate"),
    @Index(name = "idx_sample_mod_date", columnList = "modDate")})
@org.hibernate.annotations.Table(appliesTo = Sample.NAME_SPACE, comment = Sample.TABLE_DESC)
@KeySpace(DataGridConfig.MAP_NAME_DEFAULT)
public class Sample
    extends AbstractDefault<Long> {
    public static final String NAME_SPACE = "sample";
    public static final String TABLE_DESC = "샘플";

    @JsonProperty(index = 10)
    @Schema(description = "이름")
    @NotBlank(groups = ValidationGroup.Data.class)
    @Column(length = 100, nullable = false)
    @Comment("이름")
    protected String name;

    @JsonProperty(index = 20)
    @Schema(description = "설명")
    @Column(length = 1000)
    @Comment("설명")
    protected String desc;
}
```

1. Sample

- 모델의 기본 필드(아이디: AUTO INCREMENT, 상태, 등록일자, 수정일자)를 포함한 AbstractDefault 를 상속
- 또는 기본 필드(아이디: UUID, 상태, 등록일자, 수정일자)를 포함한 AbstractDefaultUuid 를 상속
 - 예) 성능이 중요한 테이블
- 또는 기본 필드(아이디: String, 상태, 등록일자, 수정일자)를 포함한 AbstractCommon 를 상속
 - 예) 사용자 테이블
- 또는 기본 필드(등록자, 수정자, 등록일자, 수정일자)를 포함한 AbstractUserCommon 를 상속
 - 예) 로그인 후 저장하는 테이블
- 스웨거 관련 어노테이션
 - @Schema : 데이터 스키마 정의
 - 데이터 관련 어노테이션 (클래스 단위)
 - @KeySpace : 데이터 저장 시 사용할 저장소 지정
 - @DynamicInsert, @DynamicUpdate : DB 등록/수정 시 쿼리를 동적으로 생성하기 위한 어노테이션

2. application.yml

- ddl-auto: update
- JPA 사용 시 DDL문 자동 실행
- 주의) DDL 업데이트 시 오동작의 우려가 있어 운영서버에는 생략(기본값: none)하고 DDL문을 직접 작성해서 실행
- dialect: 필요 시 DB 벤더를 지정 (기본값: 자동 인식)

```
spring:
  jpa:
    hibernate.ddl-auto: update
    properties.hibernate:
      dialect: org.hibernate.dialect.MySQL8Dialect
```

4. JPA CRUD 기능 사용 – Repository 2/4

```
/**
 * 샘플 DB 레파지토리
 */
public interface SampleDbRepository
    extends CrudDbRepository<Sample, Long>, SampleDynamicRepository {
}
```

```
/**
 * 샘플 동적 쿼리 인터페이스
 */
public interface SampleDynamicRepository {
    /**
     * 해당 조건으로 샘플 목록을 조회
     */
    * @param entity 검색 조건
    * @param params 일자 조건
    * @param sort 정렬 조건
    * @return 샘플 목록
    */
    List<Sample> search(Sample entity, StatsParams params, Sorting sort);

    /**
     * 해당 조건으로 샘플 페이징 목록을 조회
     */
    * @param entity 검색 조건
    * @param params 일자 조건
    * @param pageable 페이징 조건
    * @return 샘플 페이징 목록
    */
    Page<Sample> search(Sample entity, StatsParams params, Pageable pageable);

    /**
     * 해당 조건으로 샘플 현황을 조회
     */
    * @param entity 검색 조건
    * @param params 일자 조건
    * @param sort 정렬 조건
    * @return 샘플 현황
    */
    List<StatsResult> stats(Sample entity, StatsParams params, Sorting sort);
}
```

1. SampleDbRepository

- 정적(자동) 쿼리 인터페이스
 - JpaRepository(Spring 제공)를 상속
 - DB 레파지토리의 기본 CRUD 기능을 변경(재정의)하거나 새로운 기능을 추가
 - 저장 시 id 컬럼은 PK가 됨
 - 수정 시 id 컬럼은 조회 조건이 됨
 - 조회 시 Entity 객체의 특정 필드에 값이 있으면 "=" 조건을 생성(단, ">, <" 등의 조건은 동적 쿼리로)
 - 조회 시 condition 필드에 값이 있으면 "LIKE 와 대소문자 구분" 조건을 생성
- 사용법)
- condition=필드명1,[STARTS_WITH/ENDS_WITH/CONTAINS/IGNORE_CASE],...:필드명2,...
- 예)
- condition=name,STARTS_WITH:desc,CONTAINS,IGNORE_CASE
- => name LIKE ' ?% ' AND LOWER(desc) LIKE ' %?% '

2. SampleDynamicRepository

- 동적(수동) 쿼리 인터페이스
- 사용자 정의

3. SampleDbRepositoryImpl

- 동적 쿼리 인터페이스 구현체
- 동적(수동) 쿼리(JPQL) 사용시 필요한 기본 기능을 구현한 DbRepository
- < AbstractDbRepository 를 상속
- 단, 현황 조회 같은 집계 쿼리는 mybatis 사용은 권장

```
/**
 * 샘플 레파지토리와 동적 쿼리 구현체
 */
public class SampleDbRepositoryImpl
    extends AbstractDbRepository
    implements SampleDynamicRepository {
    @Override
    public List<Sample> search(Sample entity, StatsParams params, Sorting sort) {
        Query query = createQuery(entity, params);
        query.setSelect("DISTINCT a");
        query.setOrderBy(sort.toString());
        return findAll(query, Sample.class);
    }

    @Override
    public Page<Sample> search(Sample entity, StatsParams params, Pageable pageable) {
        Query query = createQuery(entity, params);
        query.setSelect("DISTINCT a");
        return findPage(query, pageable, Sample.class);
    }

    @Override
    public List<StatsResult> stats(Sample entity, StatsParams params, Sorting sort) {
        String groupBy = "DATE_FORMAT(a.regDate, '%Y-%m-%d')";
        StringBuilder select = new StringBuilder(groupBy + " AS date, COUNT(1) AS total");
        List<String> states = ImmutableList.of("ACTIVE", "DELETED");
        if (!states.isEmpty()) {
            select.append(", ");
            select.append(states.stream()
                .map(a -> "CONCAT('' + a + ''', '' + VALUE_DELIMITER + ''', SUM(CASE WHEN a.state = '' + a + '' THEN 1 ELSE 0 END))")
                .collect(Collectors.joining(", ")));
        }

        Query query = createQuery(entity, params);
        query.setSelect(select.toString());
        query.setGroupBy(groupBy);
        query.setOrderBy(sort.toString());

        List<Object[]> result = findAll(query, Object[].class);
        return result.stream()
            .map(SampleDbRepositoryImpl::createStatsResult)
            .collect(Collectors.toList());
    }
}
```


4. JPA CRUD 기능 사용 – Service 3/4

```
/**
 * 샘플 DB 서비스
 */
@Service
@CacheConfig(cacheNames = DataGridConfig.MAP_NAME_CACHE + '.' + Sample.NAME_SPACE)
public class SampleDbService
    extends AbstractCrudDbService<Sample, Long> {
    protected final SampleDbRepository repository;

    protected SampleDbService(@Lazy SampleDbService self,
        SampleDbRepository repository) {
        super(self, repository);
        this.repository = repository;
    }

    public List<Sample> search(Sample entity,          params, Sorting sort) {
        return repository.search(entity, params, sort);
    }

    public Page<Sample> search(Sample entity, StatsParams params, Pageable pageable) {
        return repository.search(entity, params, pageable);
    }

    public List<StatsResult> stats(Sample entity, StatsParams params, Sorting sort) {
        return repository.stats(entity, params, sort);
    }

    @Override
    @CacheGet
    public List<Sample> findAllOrEmpty(Sample entity, Sorting sort) {
        return super.findAllOrEmpty(entity, sort);
    }

    @CacheEvictKey
    public void cacheEvictKey(Sample entity, Sort sort) {
    }

    @CacheEvictAll
    public void cacheEvictAll() {
    }
}
```

1. SampleDbService

- DB 서비스의 기본 CRUD 기능을 구현한 CrudDbService < AbstractCrudDbService 를 상속
- DB 서비스의 기본 CRUD 기능을 변경(재정의)하거나 새로운 기능을 추가
- 캐시 관련 어노테이션 (메소드 단위)
 - @CacheConfig : 메모리 저장소(캐시명)를 지정
 - @CacheGet : 캐시 저장소에 파라미터(키)로 데이터(값)를 보관
 - @CacheEvictKey : 캐시 저장소에 보관된 데이터 중 파라미터(키)와 같은 것만 비움
 - @CacheEvictAll : 캐시 저장소에 보관된 데이터를 모두 비움
- @Lazy SampleDbService self 의 의미
 - 내부 참조로 AOP 기능을 수행하려면 자기 자신(프록시 객체)을 호출해야 가능

```
/**
 * 현황 파라미터
 */
@Data
@SuperBuilder
@NoArgsConstructor
@AllArgsConstructor
public class StatsParams {
    /**
     * 시작 등록일자(yyyy-mm-dd)
     */
    @JsonProperty(index = 10)
    @Schema(description = "시작 등록일자(yyyy-mm-dd)")
    protected String regDateStart;

    /**
     * 종료 등록일자(yyyy-mm-dd)
     */
    @JsonProperty(index = 20)
    @Schema(description = "종료 등록일자(yyyy-mm-dd)")
    protected String regDateEnd;
}
```

1. StatsParams

- ">, <" 등의 조건은 파라미터를 추가로 정의
예)
WHERE reg_date >= :regDateStart
AND reg_date <= :regDateEnd

4. JPA CRUD 기능 사용 – Controller 4/4

```
/**
 * 샘플 DB 컨트롤러
 */
@Tag(name = "샘플 DB", description = "샘플을 DB에서 관리하는 API")
@RestController
@RequestMapping(SampleDbRestController.PATH)
@ConditionalOnProperty(value = "enabled", prefix = SampleRestController.PROPERTY_PREFIX,
havingValue = "true")
public class SampleDbRestController
    extends AbstractCrudDbController<Sample, Long> {
    public static final String PATH = "/v1/" + Sample.NAME_SPACE;

    protected final SampleDbService service;

    protected SampleDbRestController(SampleDbService service) {
        super(service);
        this.service = service;
    }

    @Operation(summary = "검색")
    @ResultLogging
    @GetMapping("search")
    public List<Sample> search(Sample entity, StatsParams params, Sorting sort) {
        return service.search(entity, params, sort);
    }

    @Operation(summary = "페이지별 검색")
    @ResultLogging
    @GetMapping("search/page")
    public Page<Sample> search(Sample entity, StatsParams params, Paging page) {
        return service.search(entity, params, page.pageable());
    }

    @Operation(summary = "통계")
    @ResultLogging
    @GetMapping("stats")
    public List<StatsResult> stats(Sample entity, StatsParams params, Sorting sort) {
        return service.stats(entity, params, sort);
    }
}
```

1. SampleDbRestController

- DB 컨트롤러의 기본 CRUD 기능을 구현한 [AbstractCrudDbController](#) 를 상속
- DB 컨트롤러의 기본 CRUD 기능을 변경(재정의)하거나 새로운 기능을 추가
- 컨트롤러 재정의 시
 - @ResultLogging 은 상속되지 않으므로 다시 기술
- 프로젝트(현장)의 제약 사항으로 인해 외부에 오픈하는 메소드는 GET 과 POST 만 사용
 - 스웨거 작성 가이드
 - @Tag : REST API 설명
 - @Operation : API 별 상세 설명
 - 최소한 컨트롤러 단위로는 테스트 코드 작성을 권장

```
@Disabled
@Slf4j
@ControllerTest
public class SampleDbRestControllerTest {
    private static final Sample entity = JsonUtil.copy(SampleDbServiceTest.ENTITY, Sample.class);
    private static final Sample entityResult = Sample.builder()
        .id(SampleDbServiceTest.TEST_ID)
        .build();

    @Autowired
    public MockMvc mvc;

    @Test
    void t01save() throws Exception {
        mvc.perform(post(PATH)
            .contentType(MediaType.APPLICATION_JSON)
            .content(JsonUtil.toString(entity)))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$.").isEmpty())
            .andExpect(r -> {
                Sample result = JsonUtil.readValue(r.getResponse().getContentAsString(), Sample.class);
                assertTrue(result.getId() > 0);
                entityResult.setId(result.getId());
            })
        );
    }
}
```

5. KeyValue(IMDG) CRUD 기능 사용 – Repository, Controller 1/2

```
/**
 * 샘플 레파지토리
 */
public interface SampleRepository
    extends CrudRepository<Sample, Long> {
}
```

```
/**
 * 샘플 컨트롤러
 */
@Tag(name = "샘플", description = "샘플을 메모리에서 관리하는 API")
@RestController
@RequestMapping(SampleRestController.PATH)
public class SampleRestController
    extends AbstractCrudKvController<Sample, Long> {
    public static final String PATH = "/v1/" + Sample.NAME_SPACE + "/dg";

    protected SampleRestController(SampleService service) {
        super(service);
    }
}
```

1. SampleRepository

- 정적 쿼리 인터페이스 (SQL 과 동적 쿼리 불가)
- [KeyValueRepository](#)(Spring 제공)를 상속
- IMDG 레파지토리의 기본 CRUD 기능을 변경(재정의)하거나 새로운 기능을 추가
- 저장 시 id 컬럼은 PK가 됨
- 수정 시 id 컬럼은 조회 조건이 됨
- 조회 시 Entity 객체를 조회 조건으로 사용할 수 없고 오직 전체와 id 로만 조회 가능

2. SampleRestController

- IMDG 컨트롤러의 기본 CRUD 기능을 구현한 [AbstractCrudController](#) 를 상속
- IMDG 컨트롤러의 기본 CRUD 기능을 변경(재정의)하거나 새로운 기능을 추가

5. KeyValue(IMDG) CRUD 기능 사용 – Service 2/2

```
/**
 * 샘플 서비스
 */
@Service
public class SampleService
    extends AbstractCrudService<Sample, Long> {
    protected SampleService(@Lazy SampleService self,
        SampleRepository repository) {
        super(self, repository);
    }
}
```

1. SampleService

- IMDG 서비스의 기본 CRUD 기능을 구현한 CrudService < AbstractCrudService 를 상속
- IMDG 서비스의 기본 CRUD 기능을 변경(재정의)하거나 새로운 기능을 추가

2. hazelcast.yaml

- 데이터 그리드(분산 메모리 저장소)를 사용하기 위한 설정
- 네트워크 설정을 안 하면 로컬 메모리만 사용
- map.default
 - 메모리를 사용하는 기본 저장소 (IMDG용)
- map.cache*
 - 이름이 cache로 시작하는 저장소 (캐시용)
- map.schedule
 - 스케줄러 이중화를 위한 저장소 (스케줄용)
- eviction-policy
 - LFU : 가장 적은 참조횟수를 갖는 페이지를 교체하는 알고리즘
- size : 최대 보관할 데이터 수
- time-to-live-seconds : 데이터를 유지하는 최대 시간(초)

```
##### 설정 메뉴얼
#
https://docs.hazelcast.com/imdg/4.1/configuration/configuring-declaratively
#
https://github.com/hazelcast/hazelcast/blob/master/hazelcast/src/main/resources/hazelcast-config-4.1.xsd

##### 세션/캐시/메모리/파일 등 서버 이중화 관리
hazelcast:
  cluster-name: didm
  network:
    port:
      port: 5711
    join:
      ### 로컬 개발용
      auto-detection:
        enabled: false
      ### 운영 서버용
      tcp-ip:
        # enabled: true
      member-list:
        - 127.0.0.1:5711
        - 127.0.0.2:5711
  map:
    default:
      eviction:
        eviction-policy: LFU
        size: 10000
      time-to-live-seconds: 300
    cache*:
      eviction:
        eviction-policy: LFU
        size: 10000
      time-to-live-seconds: 60
    schedule:
      eviction:
        eviction-policy: LFU
        size: 100
```

6. MyBatis CRUD 기능 사용 – Mapper 1/3

```
<mapper namespace="com.dreamsecurity.sample.mapper.SampleMapper">
  <insert id="insert" useGeneratedKeys="true" keyProperty="id">
    INSERT INTO sample (
  </insert>

  <select id="find" resultType="com.dreamsecurity.sample.model.Sample">
    <include
refid="com.dreamsecurity.common.mapper.CommonMapper.page_top"/>
    <include refid="com.dreamsecurity.sample.mapper.SampleMapper.find"/>
  </select>

  <select id="count" resultType="long">
    <include
refid="com.dreamsecurity.common.mapper.CommonMapper.count_top"/>
    <include refid="com.dreamsecurity.sample.mapper.SampleMapper.find"/>
  </select>

  <update id="update">
    UPDATE sample
    <where>
      <if test="id != null">
        AND id = #{id}
      </if>
    </where>
  </update>

  <delete id="delete">
    DELETE FROM sample
    <include refid="where"/>
  </delete>

  <sql id="find">
    SELECT *
    FROM sample
    <include refid="where"/>
  </sql>

  <sql id="where">
    <where>
      <if test="id != null">
        AND id = #{id}
      </if>
    </where>
  </sql>
```

1. SampleMapper

- SQL 매퍼 인터페이스
- CrudMapper를 상속
- 매퍼의 기본 CRUD 기능을 변경(재정의)하거나 새로운 기능을 추가

2. SampleMapper.xml

- 모델 객체를 기반으로 자동으로 기본(참고: 왼쪽) 쿼리(xml 파일)를 생성
 1. 명령어: mapper.bat/sh 모듈명 모델객체명 매퍼객체명 매퍼파일명 (참고: 아래 명령문)
 2. 스웨거: 매퍼 컨트롤러의 기본 "GET /down" API를 활용
- 공통 쿼리(CommonMapper.xml)를 사용하여 페이징 처리
- 검색(참고: 아래) 및 통계(참고: 오른쪽 아래) 쿼리와 같은 복잡한 쿼리는 기본 쿼리에 추가 작성

```
...
  <sql id="where">
    <where>
      <if test="regDateStart != null">
        AND reg_date >= #{regDateStart}
      </if>
      <if test="regDateEnd != null">
        <![CDATA[
          AND reg_date <= #{regDateEnd}
        ]]>
      </if>
    </where>
  </sql>
...
```

```
/**
 * 샘플 CRUD 매퍼 인터페이스
 */
public interface SampleMapper
  extends CrudMapper<Sample, Long> {
  /**
   * 해당 조건으로 현황 목록을 조회
   *
   * @param params 파라미터
   * @return 현황 목록
   */
  @ResultLogging
  List<Map<String, Object>> stats(Map<String, Object> params);
}
```

```
...
  <select id="stats" resultType="java.util.LinkedHashMap">
    <![CDATA[
      SELECT DATE_FORMAT(reg_date, '%Y-%m-%d') AS date,
        COUNT(1) AS total,
        CONCAT('ACTIVE', '|', CAST(SUM(CASE WHEN state =
          'ACTIVE' THEN 1 ELSE 0 END) AS CHAR)) AS option1
      FROM sample
    ]]>
    <include refid="where"/>
    <![CDATA[
      GROUP BY DATE_FORMAT(reg_date, '%Y-%m-%d')
    ]]>
    <if test="order_by != null">
      ORDER BY ${order_by}
    </if>
  </select>
</mapper>
```

2621.2428]
s reserved.

C:\workspace\wbct\did>mapper ds-common com.dreamsecurity.sample.model.Sample
com.dreamsecurity.sample.mapper.SampleMapper src/main/resources/mapper/mysql/SampleMapper.xml

6. MyBatis CRUD 기능 사용 – Service 2/3

```
/**
 * 샘플 매퍼 서비스
 */
@Service
@ConditionalOnProperty(value = "enabled", prefix =
SampleService.PROPERTY_PREFIX, havingValue = "true")
public class SampleMapperService
    extends AbstractCrudMapperService<Sample, Long> {
    protected final SampleMapperService self;
    protected final SampleMapper mapper;

    protected SampleMapperService(@Lazy SampleMapperService self,
        SampleMapper mapper) {
        super(self, mapper);
        this.self = self;
        this.mapper = mapper;
    }

    public List<Sample> search(Sample entity, StatsParams params, Sorting sort) {
        Map<String, Object> map = createParams(entity, params, sort);
        return mapper.find(map);
    }

    public Page<Sample> search(Sample entity, StatsParams params, Paging page) {
        Map<String, Object> map = createParams(null, params, null);
        return self.findPageOrEmpty(entity, map, page);
    }

    public List<StatsResult> stats(Sample entity, StatsParams params, Sorting sort) {
        Map<String, Object> map = createParams(entity, params, sort);
        List<Map<String, Object>> result = mapper.stats(map);
        return result.stream()
            .map(a -> a.values().toArray())
            .map(SampleDbRepositoryImpl::createStatsResult)
            .collect(Collectors.toList());
    }
}
```

1. SampleMapperService

- 매퍼 서비스의 기본 CRUD 기능을 구현한 CrudDbService
- < AbstractCrudMapperService 를 상속
- 매퍼 서비스의 기본 CRUD 기능을 변경(재정의)하거나 새로운 기능을 추가

6. MyBatis CRUD 기능 사용 – Controller 3/3

```
/**
 * 샘플 매퍼 컨트롤러
 */
@Tag(name = "샘플 매퍼", description = "샘플을 관리하는 매퍼 API")
@RestController
@RequestMapping(SampleMapperRestController.PATH)
@ConditionalOnProperty(value = "enabled", prefix =
SampleService.PROPERTY_PREFIX, havingValue = "true")
public class SampleMapperRestController
    extends AbstractCrudMapperController<Sample, Long> {
    public static final String PATH = SampleDbRestController.PATH + "/" +
AbstractCrudMapperService.MAPPER;

    protected final SampleMapperService service;

    protected SampleMapperRestController(SampleMapperService service) {
        super(service);
        this.service = service;
    }

    @Operation(summary = "검색")
    @ResultLogging
    @GetMapping("search")
    public List<Sample> search(Sample entity, StatsParams params, Sorting sort) {
        return service.search(entity, params, sort);
    }

    @Operation(summary = "페이지별 검색")
    @ResultLogging
    @GetMapping("search/page")
    public Page<Sample> search(Sample entity, StatsParams params, Paging page) {
        return service.search(entity, params, page);
    }

    @Operation(summary = "통계")
    @ResultLogging
    @GetMapping("stats")
    public List<StatsResult> stats(Sample entity, StatsParams params, Sorting sort) {
        return service.stats(entity, params, sort);
    }
}
```

1. SampleMapperRestController

- 매퍼 컨트롤러의 기본 CRUD 기능을 구현한 [AbstractCrudMapperController](#)를 상속
- 매퍼 컨트롤러의 기본 CRUD 기능을 변경(재정의)하거나 새로운 기능을 추가

7. 스프링 시큐리티와 JWT 기반의 API 인증/인가 사용

```
spring:
  autoconfigure.exclude:
    # - org.springframework.boot.autoconfigure.security.servlet.SecurityAutoConfiguration

common:
  security:
    enabled: true
    api.enabled: true
    token:
      secret: dream1004!dream1004!dream1004!dream1004!dream1004!dream1004!drea
    #   expire-time-min: 240
    #   role-hierarchy: ROLE_ADMIN > ROLE_MANAGER > ROLE_USER

  include:
    cors-origins:
      # - https://*.domain1.com:[8080,8081]
    api-paths:
      - /**
    exclude:
      web-paths:
        # - /html/**
      csrf-paths:
        # - /**
      api-paths:
        - /**/api-docs/**
        - /**/common/login
        # - /**/admin/**, ALL, hasIpAddress('127.0.0.1') or hasIpAddress('0:0:0:0:0:0:1')
      ip-paths:
        # - path: /**/admin/**
        #   ips:
        #     - 127.0.0.1
        #     - 0:0:0:0:0:0:1

  default-users:
    - id: admin
      password: $2a$10$hmq/jplrU8JKkDfGXU4x1uDBgmq46D90vJ.MAOmiU3uidCxfPWmfa
      state: ACTIVE
      roles: ROLE_ADMIN
  append-users:
```

- **common.security.enabled**
 - 보안 사용 여부 (기본값: false)
 - true : 웹 표준 토큰 사용 (JWT)
 - false : 보안 해제 (SecurityAutoConfiguration 를 자동 설정 제외에 추가)
- **common.security.api.enabled**
 - 공통(로그인/토큰 발급) API 사용 여부 (기본값: false)
- **common.security.token:secret**
 - 토큰 생성 암호 (64 byte = 512 bit)
- **common.security.token:expire-time-min**
 - 기본 토큰 만료 시간 (분, 기본값: 4시간)
- **common.security.role-hierarchy**
 - 권한 상속 체계 (기본값: ROLE_ADMIN > ROLE_MANAGER > ROLE_USER)
- **common.security.include.cors-origins**
 - 5. CORS 출처 (적용 url: /**)
- **common.security.include.api-paths**
 - 6. API 경로 (기본값: path, ALL, hasRole('USER'))
- **common.security.exclude.web-paths**
 - 1. 웹 허용 경로
- **common.security.exclude.csrf-paths**
 - 2. CSRF 허용 경로 (기본값: /**)
- **common.security.exclude.api-paths**
 - 3. API 허용 경로 (기본값: /**, ALL, permitAll)
- **common.security.exclude.ip-paths**
 - 3. IP 허용 경로
- **common.security.default-users**
 - 기본 사용자 관리
 - DB 사용자 테이블 연계 : [DBUserDetailsService](#)
- **common.security.append-users**
 - 추가 사용자 관리

8. 이메일 발송

(그룹웨어 개인 계정으로 설정하고, 공용 계정 필요시 인사팀에 문의)

```
spring:
  ### 메일 서버 관리
  mail:
    host: gw.dreamsecurity.com
    username: skoh38
    password:
    properties.mail.smtp:
      auth: true
      port: 465
      ssl.enable: true
      starttls.enable: false
```

```
common:
  ### 메일 관리
  mail:
    ### 보낸 사람 이메일
    from: dsbct1@gmail.com
```

```
@ServiceTest
public class MailServiceTest {
    private static final Mail.Content content = Mail.Content.builder()
        .title("title01")
        .link("https://www.naver.com")
        .body("body01\nbody02")
        .build();
    private static final Mail mail = Mail.builder()
        .subject("subject01")
        .content(content)
        .build();

    @Autowired
    private MailService service;

    @Test
    void t01send() {
        mail.getToList().add("skoh38@gmail.com");
        mail.getCcList().add("skoh83@yahoo.co.kr");
        service.send(mail);
    }
}
```

9. (비)동기 HTTP Client 사용

(비동기는 이벤트가 아닌 스레드 기반의 확장을 의미)

common:

(비)동기 HTTP Client

http-client:

default: &http-client-default

HTTP 클라이언트 공급 업체 (httpComponents(defalut)/okHttp/simple)

vendor: httpComponents

HTTP 최대 커넥션 시간 (초)

connect-timeout-sec: 2

HTTP 최대 응답 시간 (초)

read-timeout-sec: 10

HTTP 최대 커넥션 갯수

max-conn-total: \${server.tomcat.threads.max}

HTTP URL 당 최대 커넥션 갯수

max-conn-per-route: \${common.http-client.default.max-conn-total}

sample:

<<: *http-client-default

max-conn-total: \${server.tomcat.threads.max}

max-conn-per-route: \${common.http-client.sample.max-conn-total}

- common.http-client.default.vendor

. 사용할 HTTP 클라이언트의 구현체를 선택 (기본값: httpComponents)

- common.http-client.default.max-conn-total

. 사용할 HTTP 클라이언트의 최대 커넥션 개수를 내장 톰캣의 스레드 개수와 동일하게 설정하여 서버에 동시 접속자 개수 만큼은 최소한 대기하지 않고 사용할 수 있도록 설정 (simple 은 사용 안함)

- common.http-client.max-conn-per-route

. 사용할 HTTP 클라이언트의 최대 커넥션 개수와 동일하게 설정하여 동일한 URL(IP:Port)로 최대한 커넥션을 사용 (httpComponents 만 사용)

- common.http-client.sample

. 샘플 HTTP 클라이언트를 생성하여 공통 설정 또는 헤더 값 또는 예외 처리 핸들링 등이 서로 다른 HTTP 클라이언트를 추가할 수 있다.

```
@Slf4j
@ServiceTest
public class HttpClientTest {
    private static final int count = 100;
    private static final HttpClient.Request request = HttpClient.Request.builder()
        .url("http://10.10.40.150:8010/v1/did/challenge")
        .method(HttpMethod.GET)
        .build();
```

@Autowired

private HttpClient httpClient;

@Autowired

@Qualifier("httpClientSample")

private HttpClient httpClientSample;

@Test

```
void t02httpClient() {
    List<ResponseEntity<String>> results = new ArrayList<>();
    for (int i = 0; i < count; i++) {
        ResponseEntity<String> result = httpClient.request(request, String.class);
        results.add(result);
        log.debug(LoggingConfig.ONE_LINE_100);
    }
    log.debug("results: {} {}", results.size(), results);
}
```

@Test

```
void t03httpClientAsync() {
    List<Future<ResponseEntity<String>>> results = new ArrayList<>();
    for (int i = 0; i < count; i++) {
        CompletableFuture<ResponseEntity<String>> result = httpClient.requestAsync(request, String.class,
            this::success, this::fail);
        results.add(result);
        log.debug(LoggingConfig.ONE_LINE_100);
    }
    log.debug("results: {} {}", results.size(), results);
}
```

```
private void success(ResponseEntity<String> r) {
    log.debug("response: {}", JsonUtil.toPrettyString(r));
}
```

```
private void fail(Throwable e, String request) {
    String message = "request: " + request;
    log.error(ExceptionUtil.getMessageAndType(e) + " " + message, e);
    throw new CommonException(message, e);
}
```

10. 스웨거(Rest API 테스트 화면) 기본 및 권한 테스트

(사용자별(user, manager, admin)로 발급받은 토큰(JWT)으로 API 사용 권한을 테스트한다.)

Common로그인 등의 공통 API

POST/v1/common/login 1. 로그인(토큰 발급)

Parameters

No parameters

Request body required

application/json

```
{
  "id": "user",
  "password": "dream1004"
}
```

Execute

Clear

Responses

Curl


```
curl -X 'POST' \
'http://localhost:8080/v1/common/login' \
-H 'accept: */*' \
-H 'Content-Type: application/json' \
-d '{
  "id": "user",
  "password": "dream1004"
}'
```

Request URL

```
http://localhost:8080/v1/common/login
```

Server response

Code	Details
200	<div>Response body<pre>eyJhbGciOiJIUzI1NiIsInR5cGEiOiJ1diIwMjB...ayJzdml0Ij1ic2VhYyIiwiaWF0eSIsZm9keSIsImF1dG8iOiJSSkRfSVIvdPA...0.bOUKOiGcKntJcLHpsq6WfZqYzmDso0eVpl7iZreYpB-ZlLVanvxN2ZmLuft5b-pq...</pre></div> <div><div>Download</div></div>

 **Swagger**
Supported by SMARTBEAR

/v3/api-docs

Explore

Common(공통) API

XXX OAS3

/v3/api-docs

Apis provided by blockchain development team

[Dreamsecurity - Website](#)
[Send email to Dreamsecurity](#)

DreamSecurity Corporation PROPRIETARY/CONFIDENTIAL

Servers

Available authorizations

Token (http, Bearer)

Value:

[Authorize](#) [Close](#)

공통 테스트

로그인 권한 등의 공통 테스트 API

GET	/v1/common/admin	4. ADMIN 권한 테스트	↕ 🔒
GET	/v1/common/manager	3. MANAGER 권한 테스트	↕ 🔒
GET	/v1/common/user	2. USER 권한 테스트	↕ 🔒

common:

API 문서(스웨거) 관리

api-doc:

title: Common(공통) API

description: Apis provided by blockchain development team

version: x.x.x

email: blockchain@dreamsecurity.com

실행 URL (프록시 등으로 자동 설정이 안될 경우 사용)

```
# url: http://localhost:8080
```

11. 스케줄러 이중화(Active-Standby) 구성

(스케줄 서버가 중지될 경우 다른 서버가 스케줄을 실행하도록 하여 동시에 1대만 스케줄을 실행하기 위한 구성)

```
@Slf4j
@Setter
@Service
@ConfigurationProperties(AbstractScheduleService.SCHEDULE_PROPERTY_PREFIX)
@ConditionalOnProperty(value = "enabled", prefix =
AbstractScheduleService.SCHEDULE_PROPERTY_PREFIX,
havingValue = "true")
public class SampleScheduleService<T extends Schedule>
    extends AbstractScheduleService<T> {
    @NestedConfigurationProperty
    private Properties sample = new Properties();

    public SampleScheduleService(SpringUtil springUtil,
        IScheduleService<T> service) {
        super(springUtil, service);
    }

    @Scheduled(cron = "${app.schedule.sample.cron}")
    protected void schedule() {
        super.schedule();
    }

    @Override
    protected Properties getProperties() {
        return sample;
    }

    @Override
    public void process(T active) {
        try {
            log.debug("sample");
            Thread.sleep(5_000);
        } catch (InterruptedException e) {
            log.error(e.getMessage(), e);
        }
    }
}
```

```
common:
  ### 스케줄 관리
  schedule:
    ### 스케줄 이중화 저장소 (DG: 데이터 그리드, DB:
    데이터 베이스, NONE: 사용안함, 기본값: NONE)
    save-to: DG

app:
  ### 스케줄 관리
  schedule:
    ### 전체 스케줄 활성화 여부 (기본값: false)
    enabled: true
  sample:
    ### 스케줄 활성화 여부 (기본값: false)
    enabled: true
    ### 스케줄 반복 주기
    cron: "*/10 * * * *"
    ### 헬스 체크 시간 (기본값:
    cron반복주기+(cron반복주기/2)초, 음수이면 사용안함)
    health-check-time-sec: 15
```

- **common.schedule.save-to**
 - . DG : 분산메모리를 사용한 서버 간 스케줄 정보 공유
 - . DB : 데이터베이스를 사용한 서버 간 스케줄 정보 공유
- **app.schedule.sample**
 - . 아래 스케줄 코드의 프로퍼티 이름과 동일하게
`private Properties sample = new Properties();`
- **app.schedule.health-check-time-sec**
 - . 스케줄 정보의 수정일시가 설정한 시간 동안 업데이트되지 않으면 실행 중인 스케줄러가 정지된 것으로 판단하고 해당 스케줄을 다른 서버가 점유

12. 테스트 코드 작성

(가능하면 단위테스트도 간단한 시나리오(CRUD)를 구상해서 순서대로 실행하고, 예외 상황도 발생시킬 것을 권장)

```
@Documented
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@TestMethodOrder(MethodOrderer.MethodName.class)
@Rollback(false)
@SpringBootTest(classes = CommonApplication.class)
@ActiveProfiles("test")
public @interface ServiceTest {
}
```

```
@Documented
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@ServiceTest
@AutoConfigureMockMvc
@Import(ControllerTest.ControllerConfig.class)
public @interface ControllerTest {
    class ControllerConfig {
        @Bean
        MockMvcBuilderCustomizer utf8Config() {
            return builder ->
                builder.addFilters(new CharacterEncodingFilter(StandardCharsets.UTF_8.toString(), true));
        }
    }
}
```

```
@ServiceTest
public class SampleServiceTest {
    private static final Sample entity = JsonUtil.copy(SampleDbServiceTest.ENTITY,
        Sample.class);
    private static final Sample entityResult = Sample.builder()
        .id(SampleDbServiceTest.TEST_ID)
        .build();

    @Autowired
    private SampleService service;

    @Test
    void t01save() {
        log.debug("entity: {}", JsonUtil.toPrettyString(entity.toString()));
        Sample result = service.insert(entity);
        log.debug("result: {}", JsonUtil.toPrettyString(result.toString()));
        Assertions.assertNotNull(result.getId());
        entityResult.setId(result.getId());
    }
}
```

```
@ControllerTest
public class SampleDbRestControllerTest {
    @Autowired
    public MockMvc mvc;

    @Test
    void t91validate() throws Exception {
        mvc.perform(post(PATH)
            .contentType(MediaType.APPLICATION_JSON)
            .content(JsonUtil.toString(new Sample())))
            .andDo(print())
            .andExpect(status().isBadRequest());
    }
}
```

- 단위테스트 코드는 내가 또는 남이 작성한 코드를 수정했을 때 발생하는 **사이드 이펙트를 최소한으로 하기 위한 사전 예방 조치**이다.

- **ServiceTest** 어노테이션은 일반 클래스를 테스트할 때 사용

- **ControllerTest** 어노테이션은 컨트롤러 클래스를 테스트할 때 사용

- 컨트롤러가 있는 로직은 **컨트롤러 단위로 테스트**하고, 컨트롤러가 없는 로직은 서비스 단위로 테스트 코드를 작성

- 가능하면 모든 메소드 단위로 단위테스트 코드를 작성할 것을 권장하지만, 내부에서 호출이 일어나는 메소드는 **최상위 메소드만 테스트**하면 된다.

- 테스트 커버리지 **기본 목표는 80%** 이지만 현재는 50% 로 하향
(또한, 단위 테스트 코드가 없는 모듈은 별도 관리)

- 최근 프로젝트는 정적분석과 함께 **테스트 커버리지를 품질 지표로 채택**

- 현재 팀내 **패키지(제품) 중 Java 언어만 CI/CD 를 도입 중** (향후 확장)