당신도 할 수 있는 레거시 프로젝트 개선 이야기

작성자 : 이경일 (kellin.me@navercorp.com, leekyoungil@gmail.com)
소속팀 / 상위부서 : 쇼핑데이터개발 / 쇼핑플랫폼, 비즈OCR개발 / Glace개발

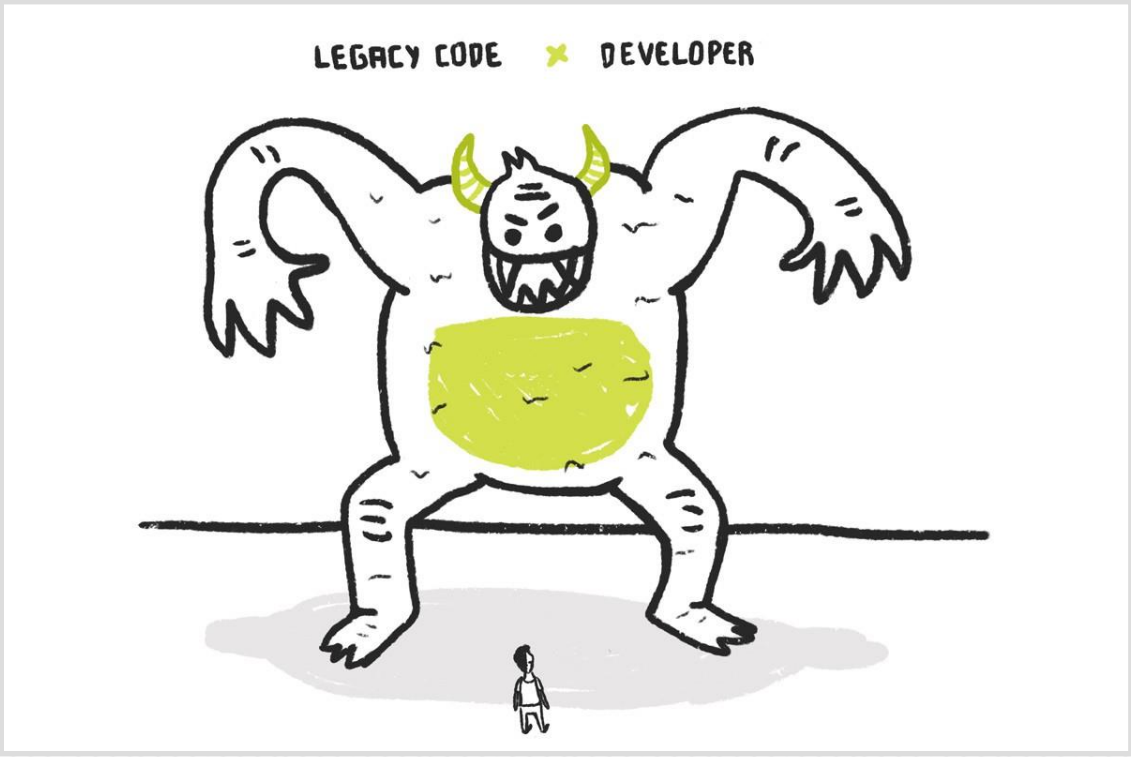
NAVER

발표자

- 이경일 / leekyoungil@gmail.com
- 네이버쇼핑데이터, 비즈OCR개발
- 노가다 전문 개발자 (잡부)

레거시 코드란 무엇일까요?

- 어디부터 봐야 할지 막막한 코드?



이미지 출처 : https://village.oyaore.com

레거시 코드란 무엇일까요?

- 복잡한 코드? (?)

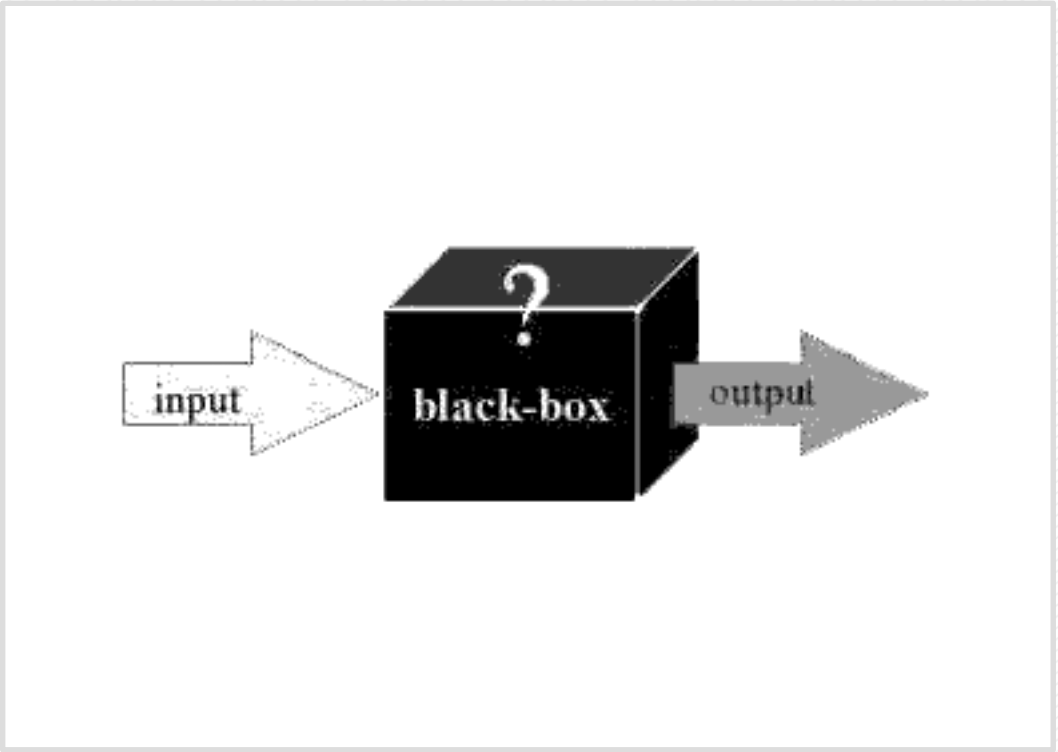
```
function register()
{
    if (!empty($_POST)) {
        $msg = '';
        if ($_POST['user_name']) {
            if ($_POST['user_password_new']) {
                if ($_POST['user_password_new'] === $_POST['user_password_repeat']) {
                    if (strlen($_POST['user_password_new']) > 5) {
                        if (strlen($_POST['user_name']) < 65 && strlen($_POST['user_name']) > 1) {
                            if (preg_match('/^[a-z\d]{2,64}$/i', $_POST['user_name'])) {
                                $user = read_user($_POST['user_name']);
                                if (!isset($user['user_name'])) {
                                    if ($_POST['user_email']) {
                                        if (strlen($_POST['user_email']) < 65) {
                                            if (filter_var($_POST['user_email'], FILTER_VALIDATE_EMAIL)) {
                                                create_user();
                                                $_SESSION['msg'] = 'You are now registered so please login';
                                                header('Location: ' . $_SERVER['PHP_SELF']);
                                                exit();
                                            } else $msg = 'You must provide a valid email address';
                                        } else $msg = 'Email must be less than 64 characters';
                                    } else $msg = 'Email cannot be empty';
                                } else $msg = 'Username already exists';
                            } else $msg = 'Username must be only a-z, A-Z, 0-9';
                        } else $msg = 'Username must be between 2 and 64 characters';
                    } else $msg = 'Password must be at least 6 characters';
                } else $msg = 'Passwords do not match';
            } else $msg = 'Empty Password';
        } else $msg = 'Empty Username';
        $_SESSION['msg'] = $msg;
    }
    return register_form();
}
```

레거시 코드란 무엇일까요?

- 남이(주로 퇴사자) 짠 코드? ~~(나똥을왜내가...)~~

레거시 코드란

- 테스트 코드로 커버되지 않으며
- 유지 보수가 되고 있지 않은 코드 (유지 보수가 어려운 코드)



개발자들이 레거시 코드를 외면하는 이유?

- 다른 사람이 짠 코드는 수정하기 싫다.
- 신규 프로젝트가 훨씬 재미있다.
- 조직에서 인정받기 어렵다??

레거시 코드를 다시 한번 보자!

- 오랜 시간 자신의 역할을 실행한 안정적인 코드
- 조직의 축적된 비즈니스 로직의 구현체

레거시 코드를 다시 한번 보자!

- 누군가 말했다. "The developers speak by the code."
- 앞서 운영했던 개발자들의 오랜 Know-How 가 있다.

결국 핵심은...

- 코드를 개선하고 내 것으로 만들어야 한다는 것!

레거시 코드 개선...

- 레거시 코드를 어떻게 개선해야 할까요?
  - DDD로 바꾸세요. -> Domain Driven Development
  - MSA 하세요.

(후훗 전 저자 싸인 받았...)

레거시 코드 개선...

- 오늘 이야기는 이론적인 내용이 아닙니다. (사실저도몰...)
- 그냥 제가 책임감을 가지고 (먹고살기 위해) 서비스의 운영방식, 코드를 개선한 이야기를 하고자 합니다.

오늘 할 이야기...

- 내 편으로 만들기
- 급한 불부터 끄기
- 한 걸음씩 가기
- 아픈 곳 고치기
- 조금 더 다듬기

그전에...

- BizOCR Project 를 소개합니다.

BizOCR 프로젝트는요...

- 영수증이나, 사업자 등록증 인식 (이미지)
- OCR을 이용한 이미지 안의 문자를 추출하여 정제함
- Json Document 로 출력함 (HTTP REST API 제공)

BizOCR 프로젝트는요...

- Naver : Myplace 방문 인증 (영수증 이용)
- Line, Naver : CONOMI (일본 맛 집, 영수증 이용)
- Naver Business Platform : 사업자 등록증
- Naver Smart Place : 사업자 등록증

내 편으로 만들기...

- If you are not with me, then you are my enemy.
  (잘리기 싫으면 나를 빨리 개선해라)

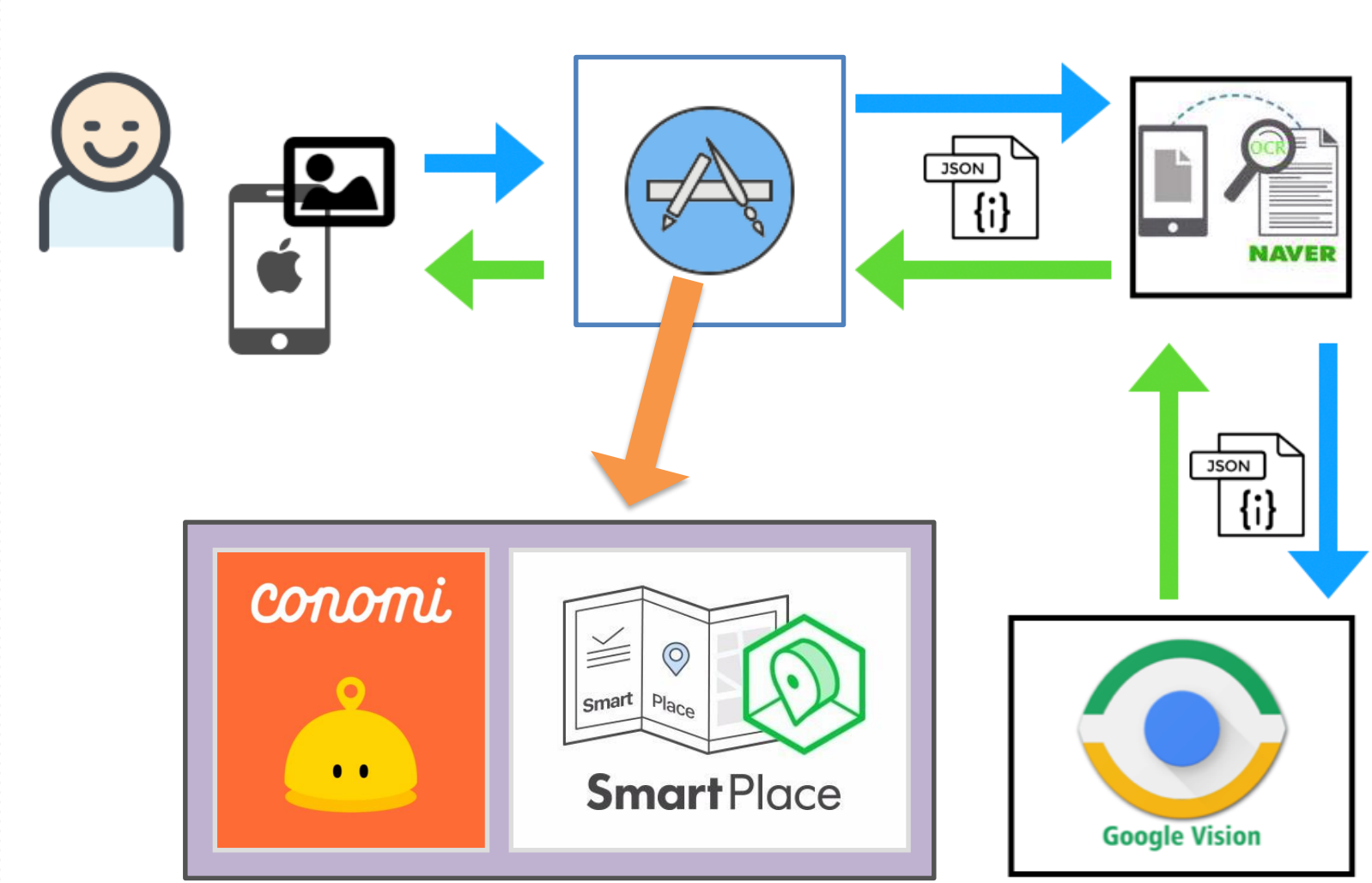
내 편으로 만들기...

- 첫 만남
- XX대의 운영 서버와 1대의 개발서버 (엄청난 사양의)

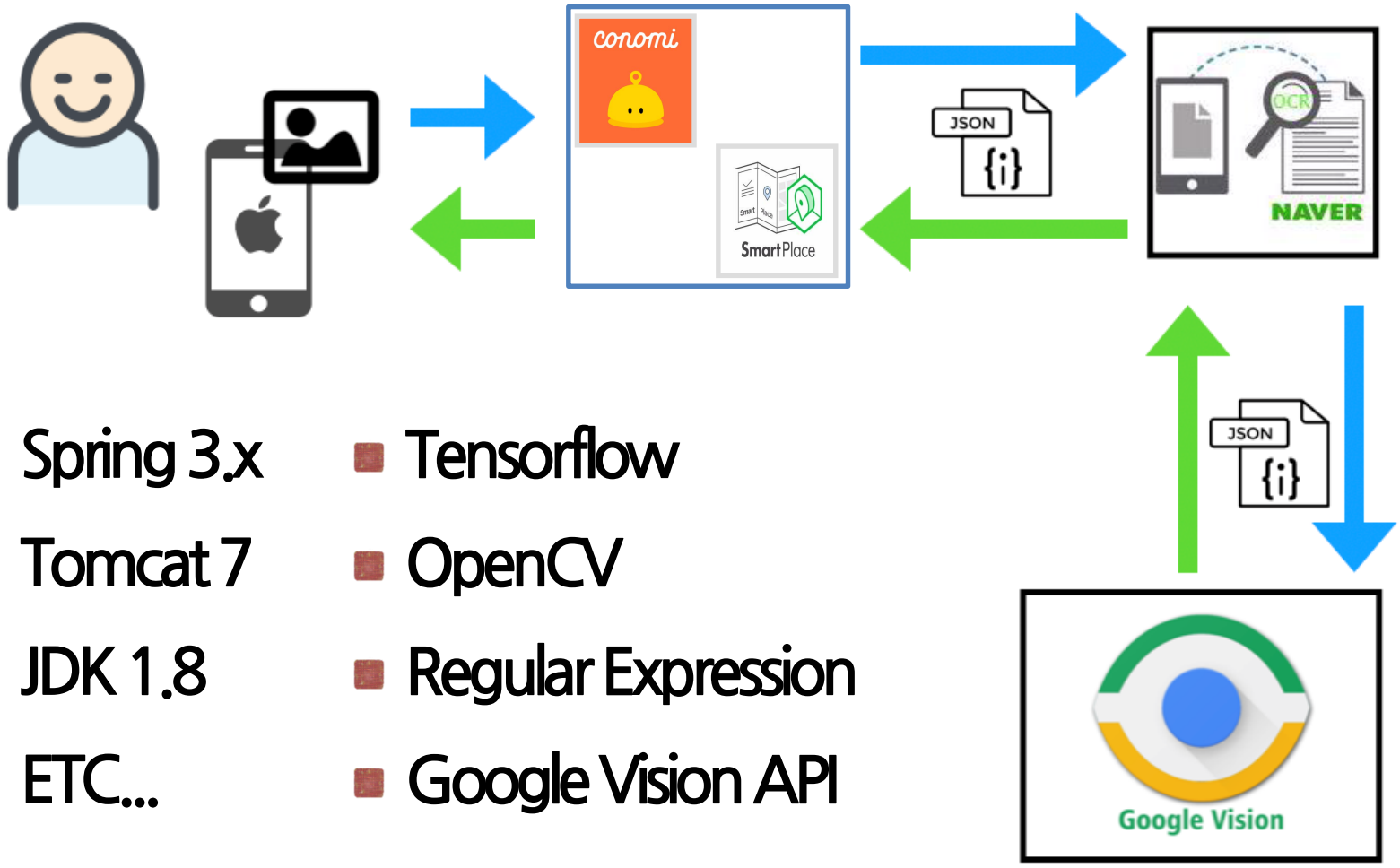
내 편으로 만들기...

- 당시 BizOCR을 사용하는 운영 중인 서비스는 2가지
- 일본 맛집 서비스 코노미, 한국 사업자 등록증

내 편으로 만들기...



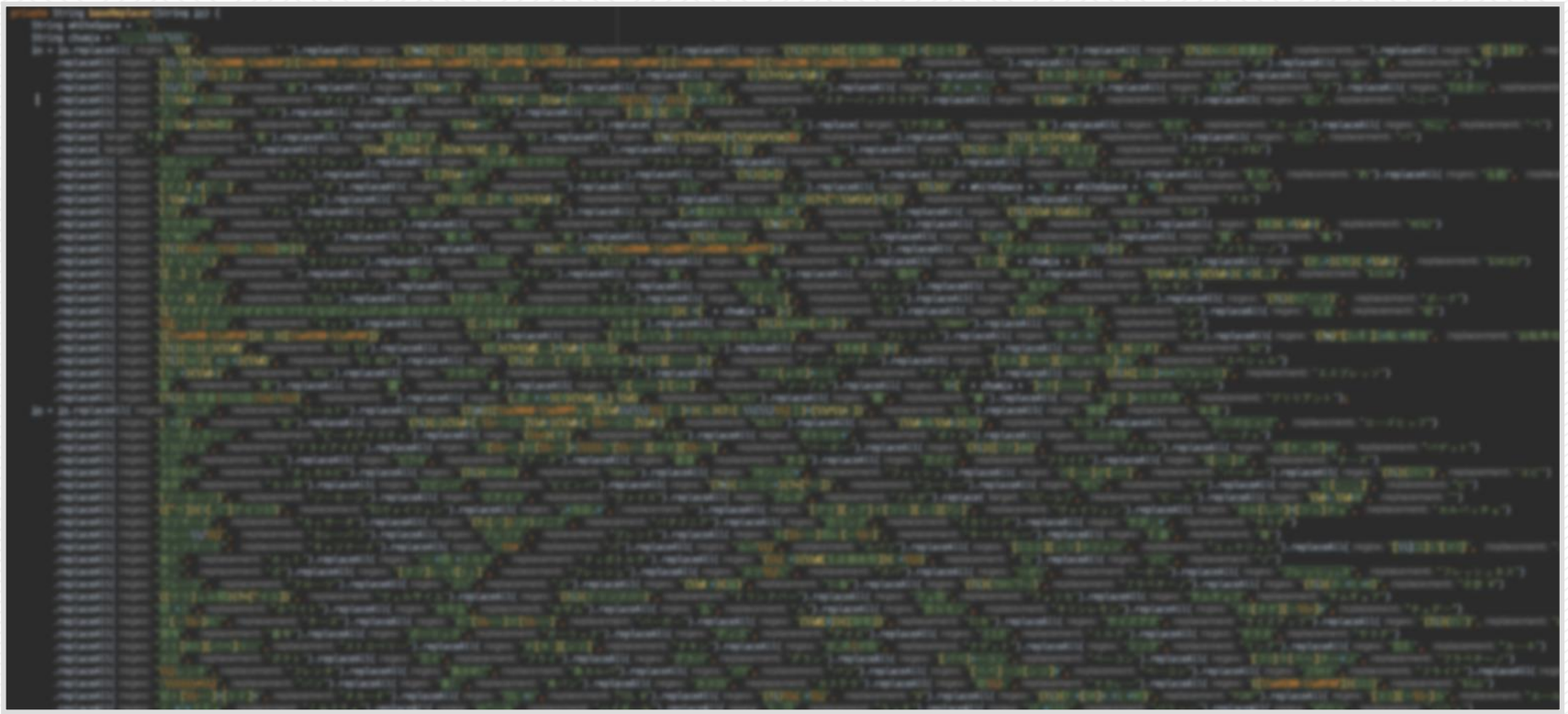
내 편으로 만들기...



- Spring 3.x
- Tomcat 7
- JDK 1.8
- ETC...
- Tensorflow
- OpenCV
- Regular Expression
- Google Vision API

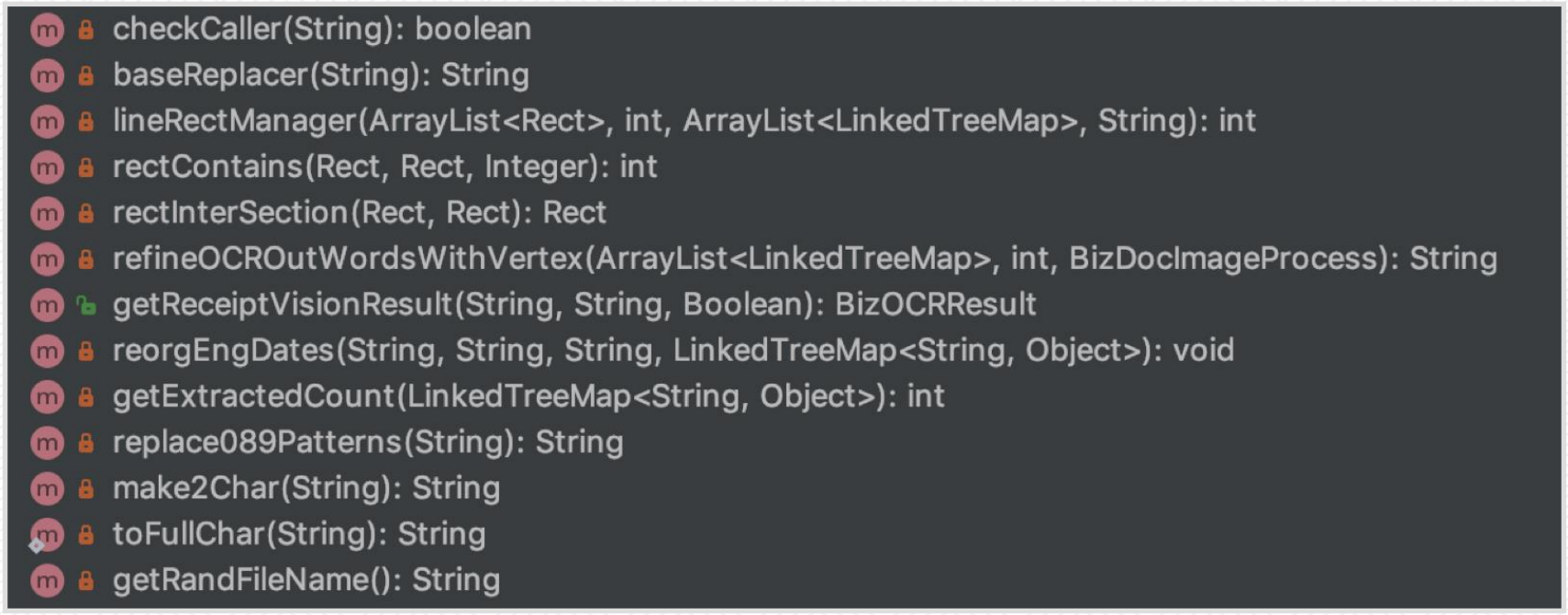
내 편으로 만들기...

- 그리고 소스는 정규식 지옥...



내 편으로 만들기...

- Transaction script 방식의 Service Class.
- 모든 로직이 한 Class 안에 있는 구조

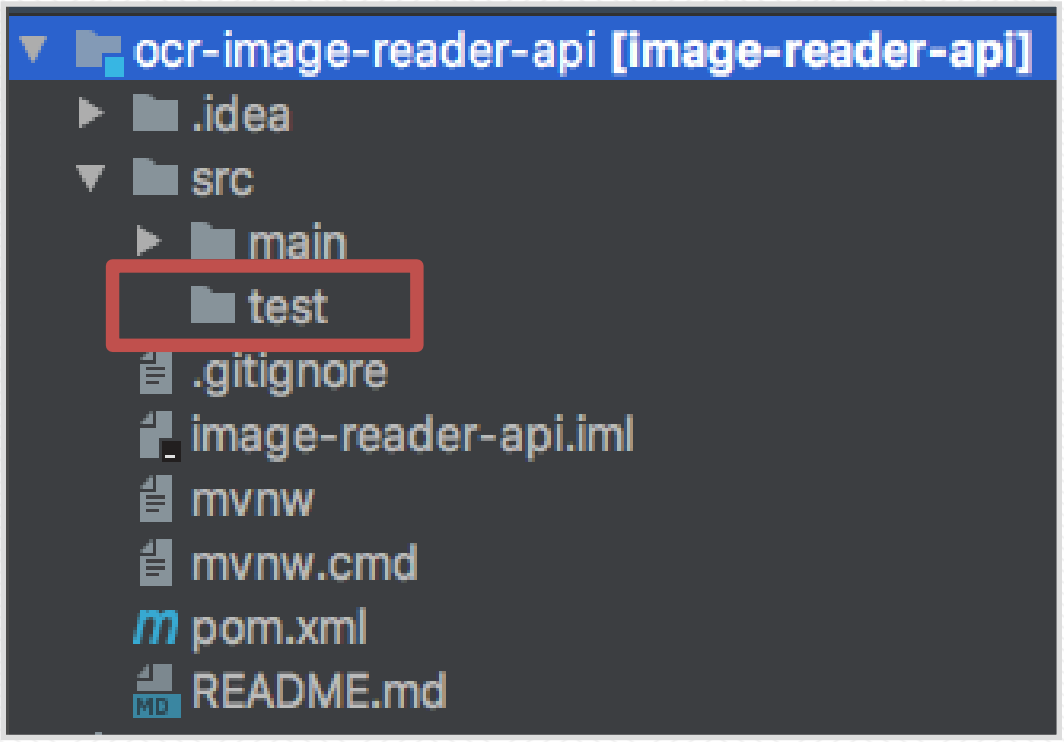


내 편으로 만들기...

- 어떻게 내 편으로 만들어야 하나?
- 로직을 파악하는 것이 최우선 (뭘 알아야 하던가 말던가)
- 직접 돌려 보는 것이 역시 가장 좋음

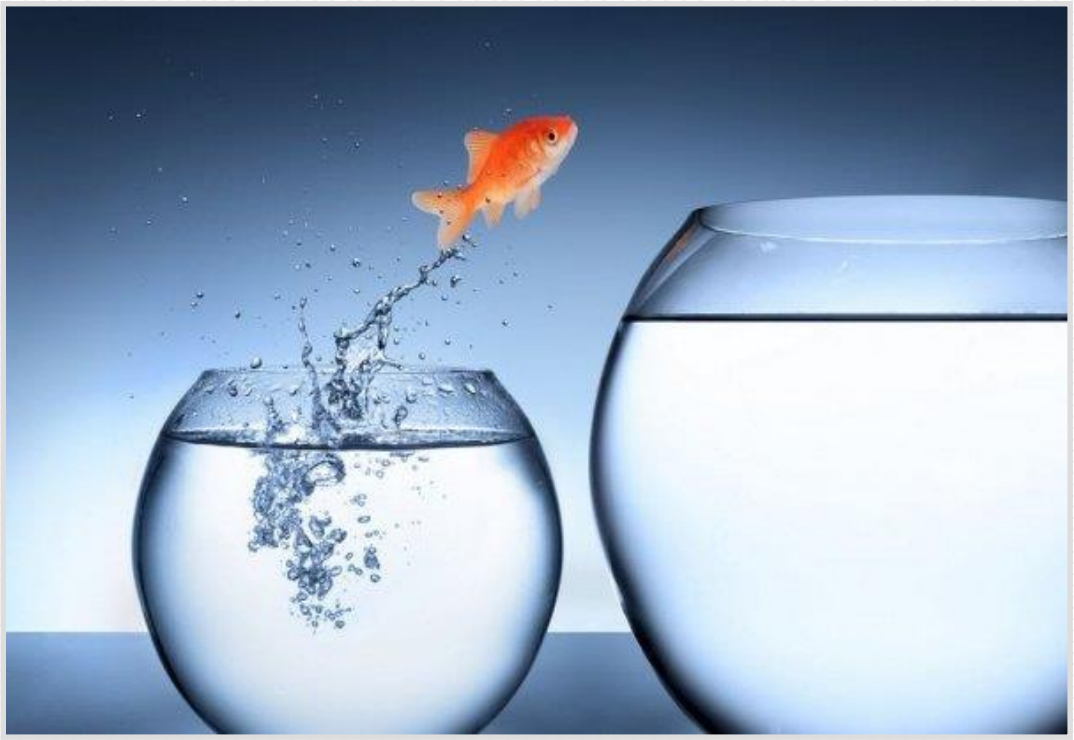
내 편으로 만들기...

- 로직을 파악하기 위해서는 역시 TestCase를 보는 것!
- 하지만 테스트 케이스가 없었음...



내 편으로 만들기...

- 성장(self landing)을 위한 기회... (라고 정말로 생각했음)



내 편으로 만들기...

- 테스트 코드를 짜기 시작했습니다.
- JUnit이 아닌 Spock Framework을 사용했습니다.

내 편으로 만들기...

- 자연스럽게 레가시 코드를 파악할 수 있음
- Refactoring 을 할만한 부분이 보이기 시작함
- 처음으로 PR 을 날렸음

내 편으로 만들기...

- 베프까진 아니더라도 사친 정도까진 친해짐

  별로 친해지고 싶진 않았지만 (먹고살아야 하니) 어쩔 수 없었음

급한 불부터 끄기... (운영 중이기 때문에)

- 당장 해결해야 할 우선순위를 정해야 합니다.

급한 불부터 끄기... (운영 중이기 때문에)

- 같이 일하는 사람은 총 3명 (저 포함)

이미지처리

감독

잡부

급한 불부터 끄기... (운영 중이기 때문에)

- 이들에게 급한 불은 총 2가지
  - 정규식을 유지 보수하기 힘들다.
  - Application 구동 시간이 너무 오래 걸린다.

급한 불부터 끄기... (운영 중이기 때문에)

- 정규식을 유지 보수하기 힘든 이유...

급한 불부터 끄기... (운영 중이기 때문에)

- 하나를 수정하고 다시 빌드 해서 검증... (무한 반복)
  - 이런 식으로 치환 & 추출하는 케이스는 5개 이상

급한 불부터 끄기... (운영 중이기 때문에)

- OCR 작업 중 약 50%는 정규식 작업
  - 소스코드의 수정이 필요 없이 정규식만 수정을 해야 할 상황이 생각보다 많다는 소리 (새로운 패턴을 추가한다든지)

급한 불부터 끄기... (운영 중이기 때문에)

- OCR 작업 중 약 50%는 정규식 작업
- 게다가 순서가 중요함
  - 2018年5月23日(水) 20時24分000101₩n重さ(g
    - 특수문자, 의미없는 숫자, 특수문자 ( ) $#% 등등 제거
    - 필요 없는 水 같은 요일 TEXT 제거
    - 날짜 패턴 (20₩d{2}₩s?[₩-₩/年₩s]₩s?(?:0[1-9]|1[1-2… 사용해서 값 추출

급한 불부터 끄기... (운영 중이기 때문에)

- 코드 안에 있는 모든 정규식을 추출함

급한 불부터 끄기... (운영 중이기 때문에)

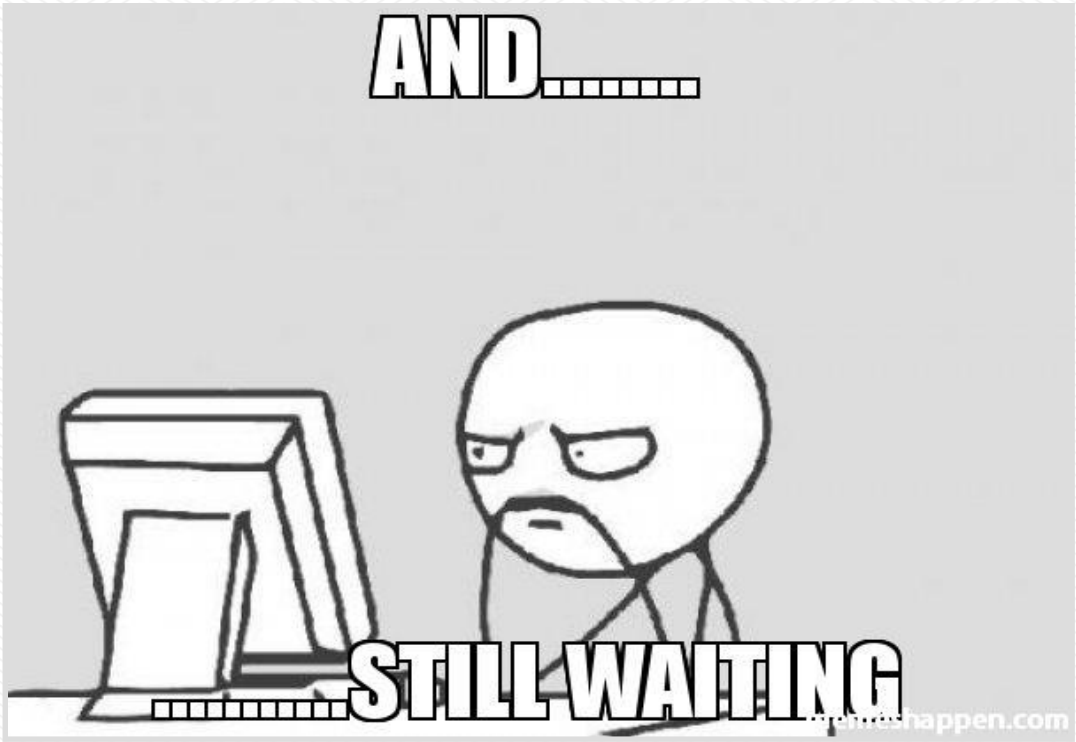
- 정규식을 YML로 (즉 설정으로서) 분리함

급한 불부터 끄기... (운영 중이기 때문에)

- YML -> MAP 변환 후 Loop 문으로 처리함
- 소스코드량 대폭적인 감소 & 유지 보수가 수월해짐
  - 대신 그 코드가 설정으로 갔다는....

급한 불부터 끄기... (운영 중이기 때문에)

- 다음 급한 문제는 구동 시간이 너무 오래 걸린다는 것
- 약 1분 이상...



급한 불부터 끄기... (운영 중이기 때문에)

- 왜 이렇게 오래 걸리는지 살펴보니...
- Oracle, ElasticSearch, 각종 캐싱, 각종 Bean...초기화

급한 불부터 끄기... (운영 중이기 때문에)

- 쓰지도 않는 불필요한 초기화가 너무 많음
- 원인은 Common library

```
<dependency>
    <groupId>com.naver [redacted]</groupId>
    <artifactId>land-common</artifactId>
    <version> [redacted] SNAPSHOT</version>
</dependency>
```

급한 불부터 끄기... (운영 중이기 때문에)

- Common library 소스 분석 후 필요한 부분을 찾음

급한 불부터 끄기... (운영 중이기 때문에)

- 크게 3가지를 사용하고 있음
  - API 통신 관련 설정들 (주소, 호출 파라미터 정의)
  - 상황별 Exception, Json 관련로직, 각종 DTO 객체들
  - XML로 Inject 되고 있던 Bean 객체들

급한 불부터 끄기... (운영 중이기 때문에)

- 과감하게 Dependency 삭제
- Common Library에서 꼭 필요한 부분만 복사함
- Custom 된 Class 대신 Spring의 Feature를 사용함

  주로 RestTemplate 같은...

급한 불부터 끄기... (운영 중이기 때문에)

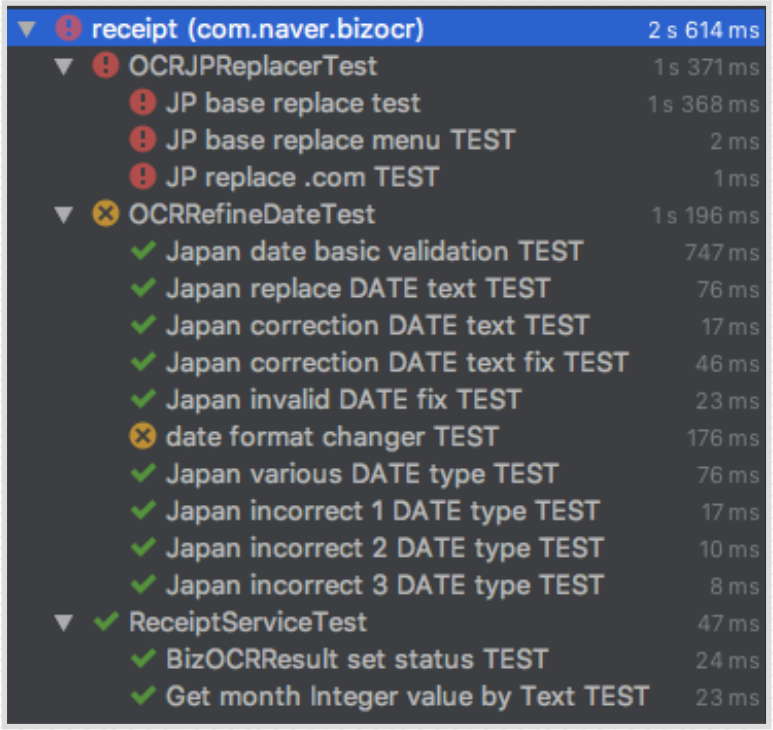
- XML 설정 부분을 properties + Java config로 수정

```
papago.requestInfo.target = PapagoTrans
papago.requestInfo.se : UTF-8
papago.requestInfo.re : UTF-8
papago.requestInfo.timeout : 1000
papago.requestInfo.address : http://test
papago.requestInfo.templateUse : N
```

```
@Slf4j
@Configuration
@PropertySource("classpath:config/external/papago/papago_configuration-${spring.profiles.active}.properties")
@ConfigurationProperties(prefix = "papago")
public class PapagoRequestConfigration {

    @Setter
    private PapagoRequestInfo requestInfo;

    @Bean
    public PapagoRequestInfo papagoRequestInfo () { return this.requestInfo; }
}
```

급한 불부터 끄기... (운영 중이기 때문에)

- 오류 나는 부분 전부 찾아서 수정후...
- 미리 짜둔 테스트 케이스로 검증함

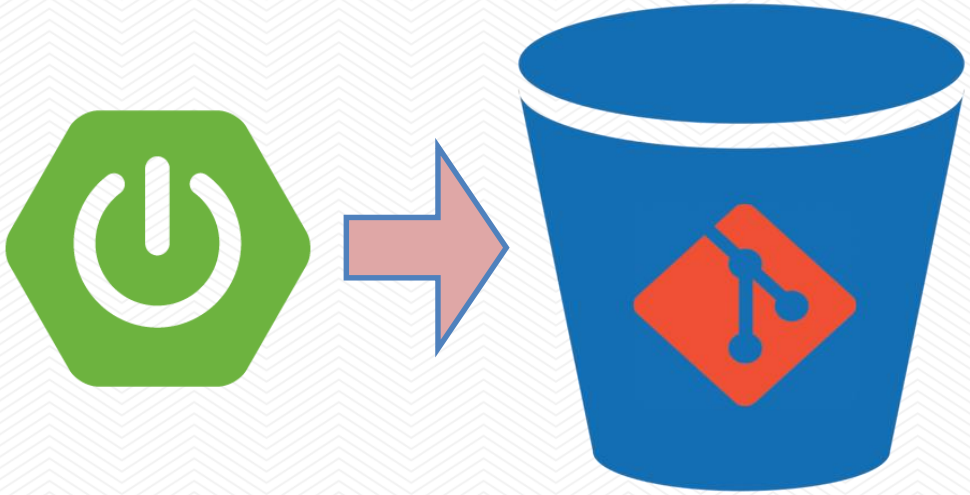


급한 불부터 끄기... (운영 중이기 때문에)

- 내친김에 Spring 3.x를 버리고 Spring Boot로...

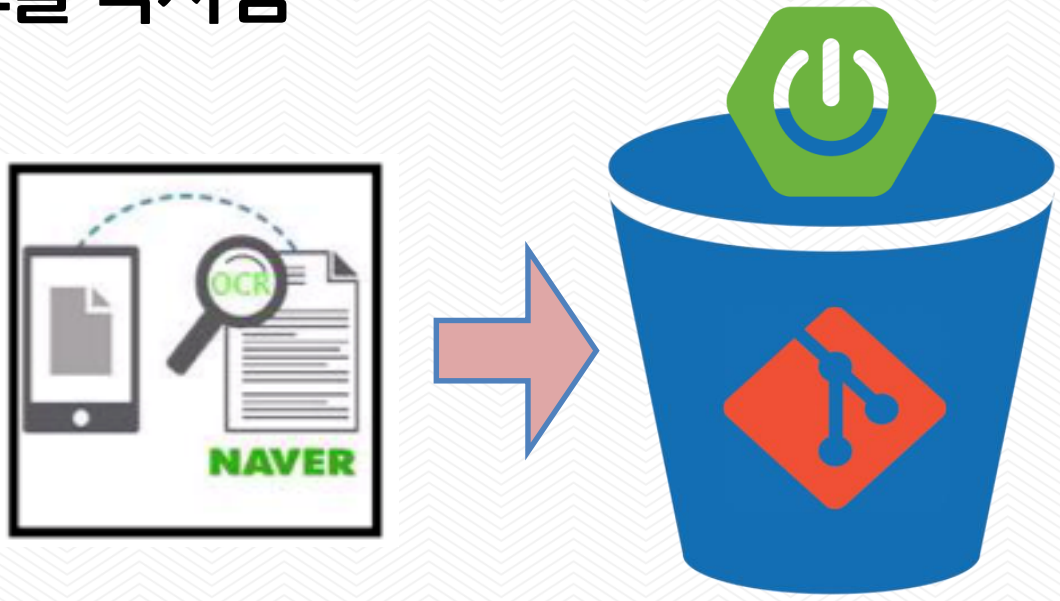
급한 불부터 끄기... (운영 중이기 때문에)

- 새로운 Repository 생성 후
- 기본적인 Spring Boot MVC 프로젝트 세팅함



급한 불부터 끄기... (운영 중이기 때문에)

- 새로운 Repository 생성 후
- 기본적인 Spring Boot MVC 프로젝트 세팅함
- 여기에 기존 소스를 복사함



급한 불부터 끄기... (운영 중이기 때문에)

- 팀장님께 코드 리뷰 를(라고 쓰고 허락이라고 읽는다) 받고
- 일부 서버에 배포를 해서 검증을 했음

급한 불부터 끄기... (운영 중이기 때문에)

- 문제가 없어서 전부 배포함
- 다 같이 소고기 먹었음 (삼합... 또 먹고 싶...)

급한 불부터 끄기... (운영 중이기 때문에)

- 구동시간 감축으로 인한 Local 작업 효율 향상
  - ☑ 약 1분 ~ 2분 걸리던 시간을 3~5초로 감소
- Spring 3.x에서 Spring Boot로 Framework 버전업
- Common Library 제거로 인한 프로젝트 독립

한 걸음씩 가기

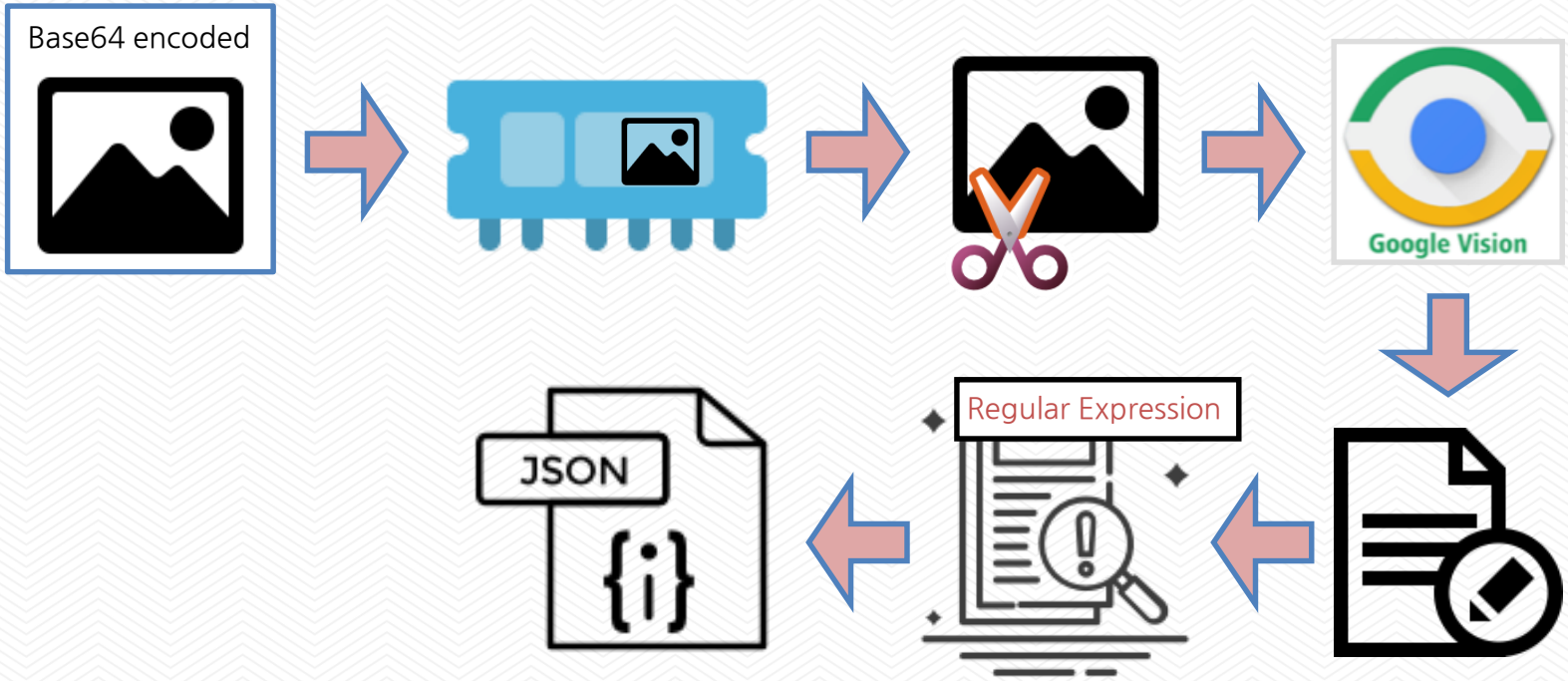
- A journey of a thousand miles
  begins with a single step.

한 걸음씩 가기

- 코드 패키지 분리

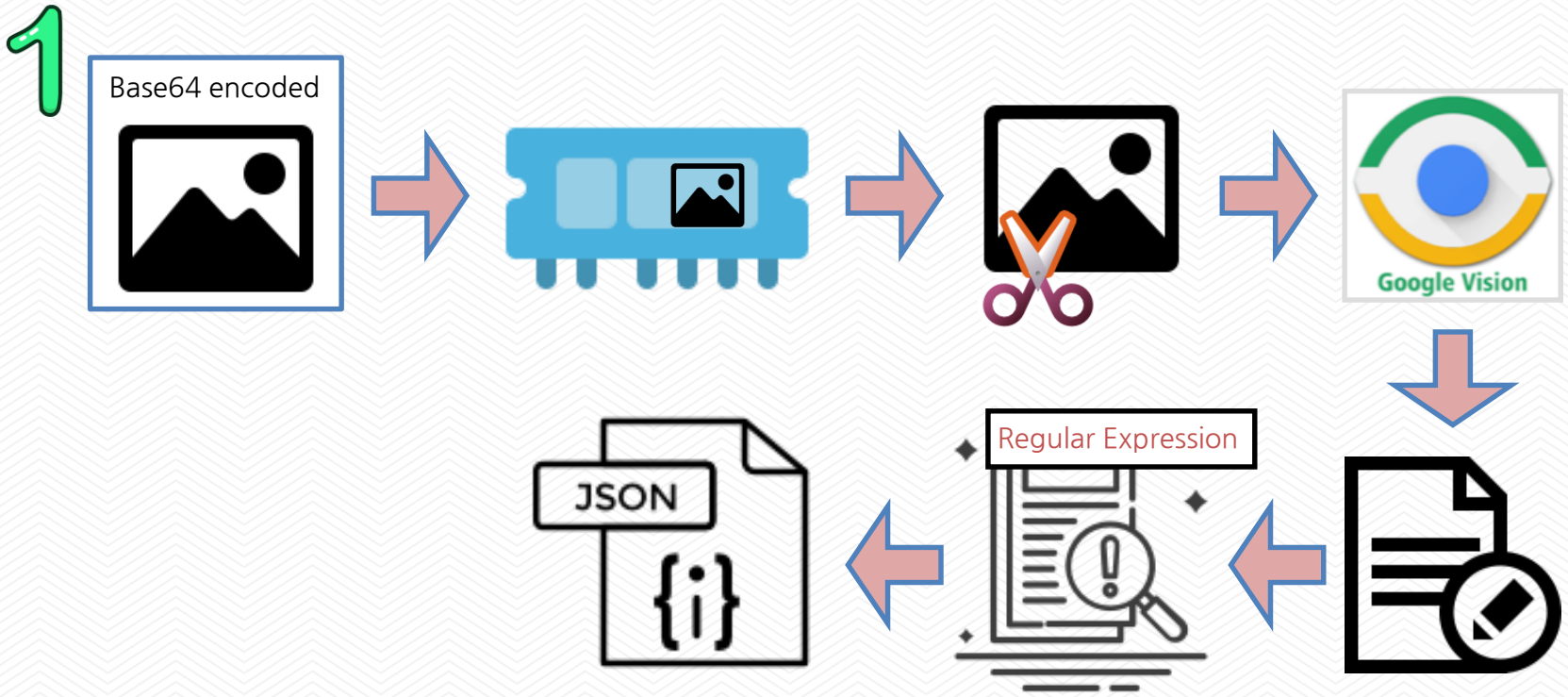
한 걸음씩 가기

- 프로세스의 흐름을 정리하면...



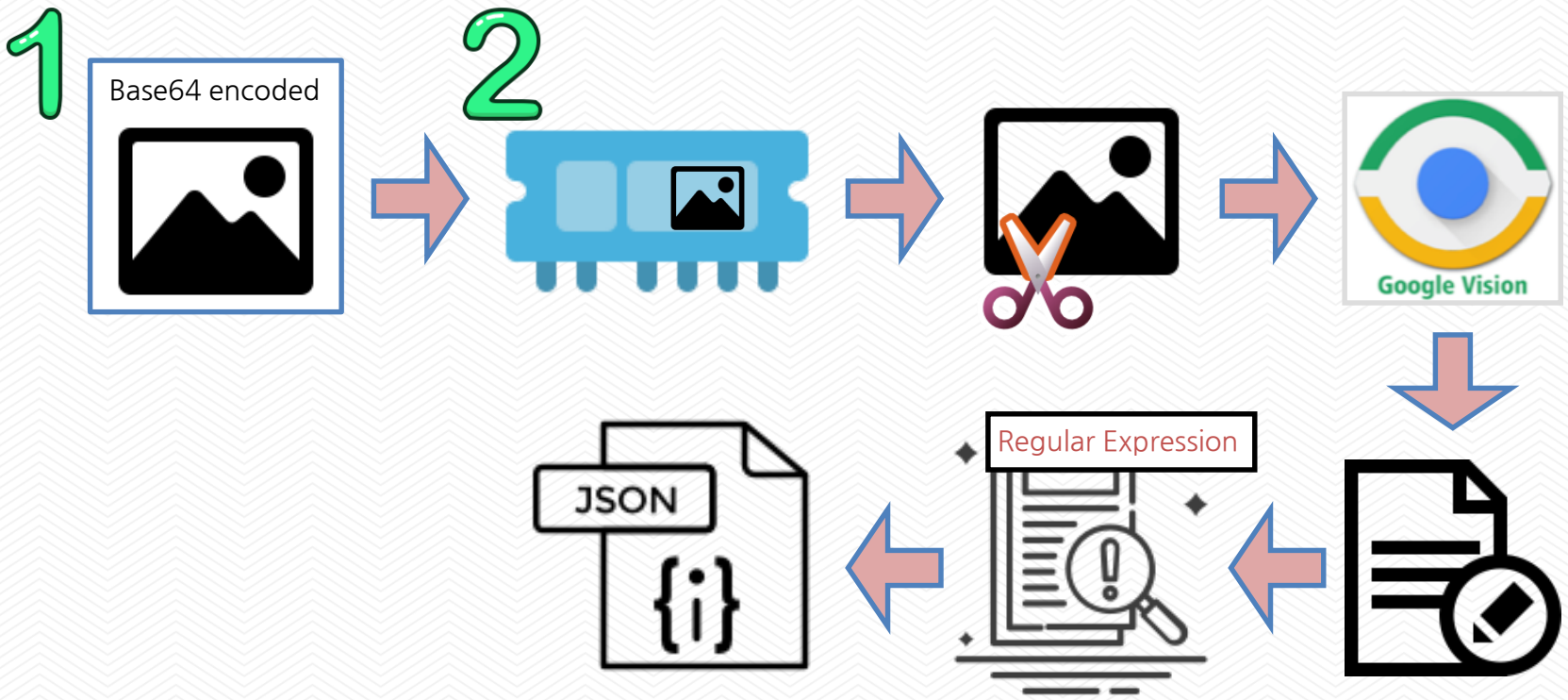
한 걸음씩 가기

- 이미지가 들어오면 구분, Validation



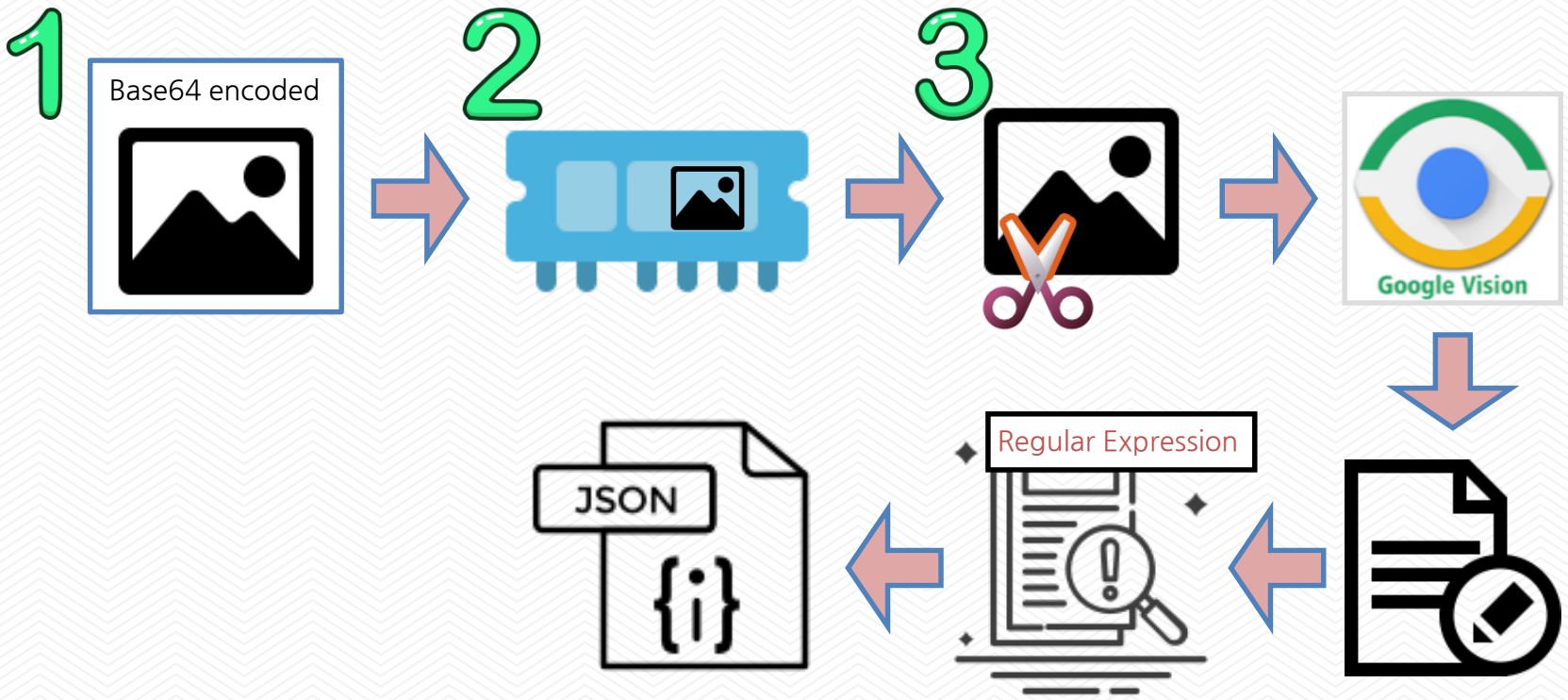
한 걸음씩 가기

- 이미지 데이터를 메모리에 올림



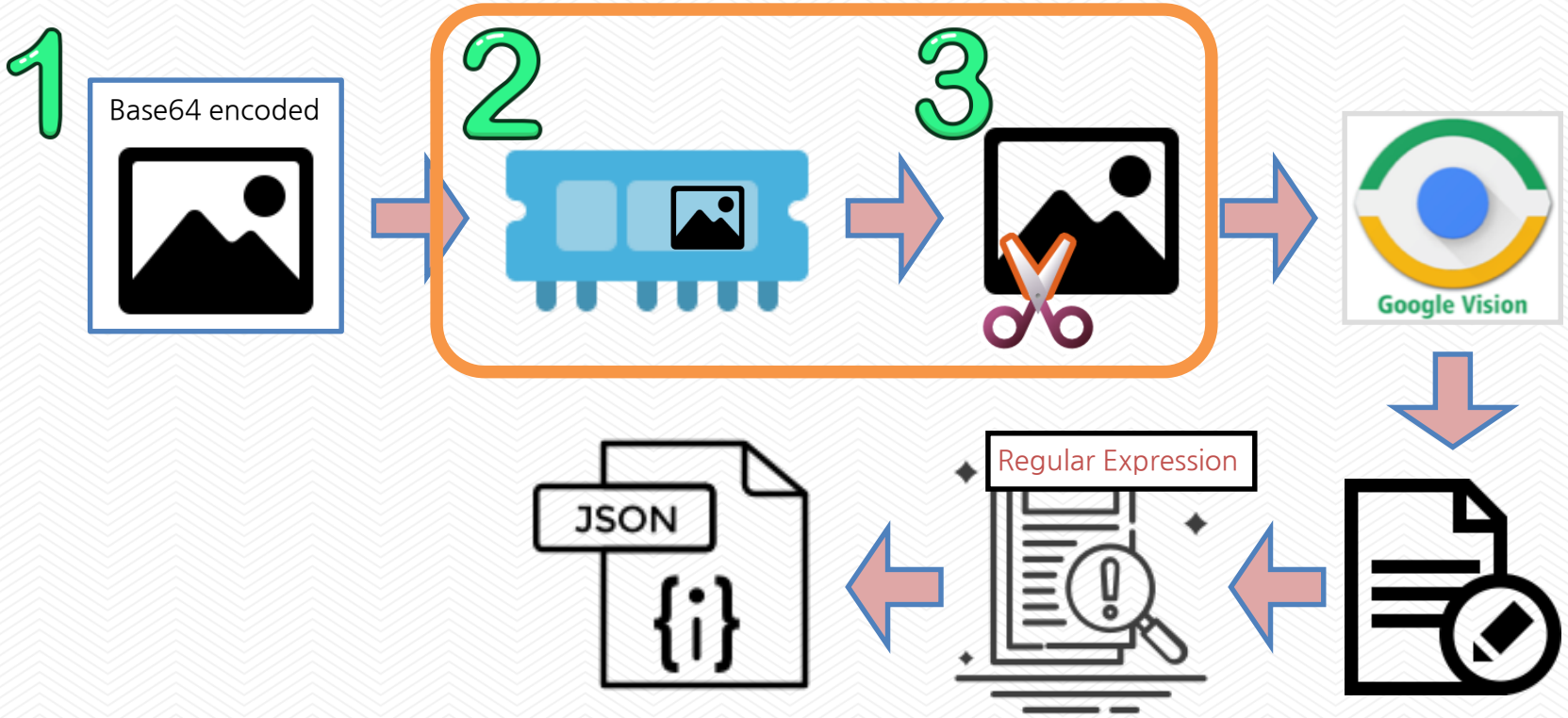
한 걸음씩 가기

- 이미지를 편집함



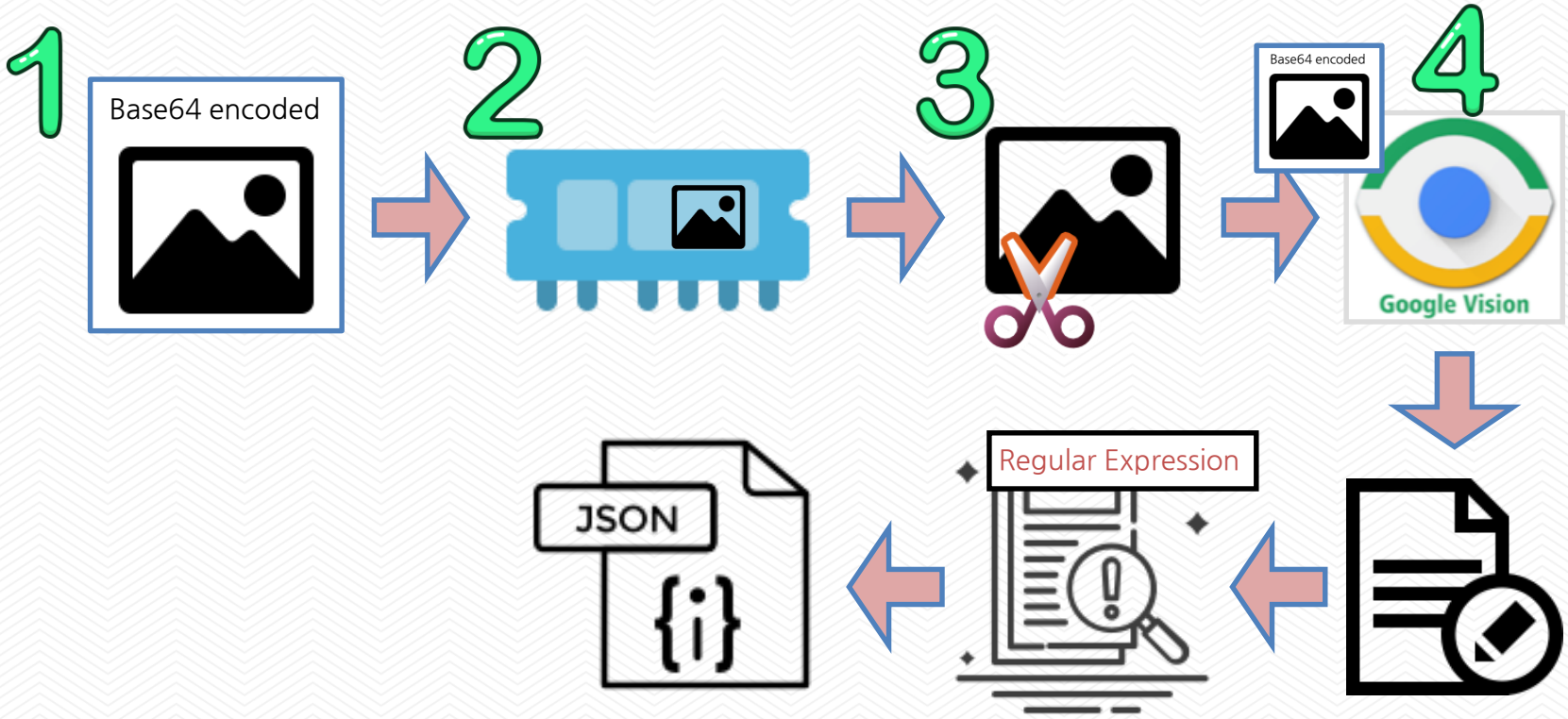
한 걸음씩 가기

- OpenCV와, Tensorflow 사용



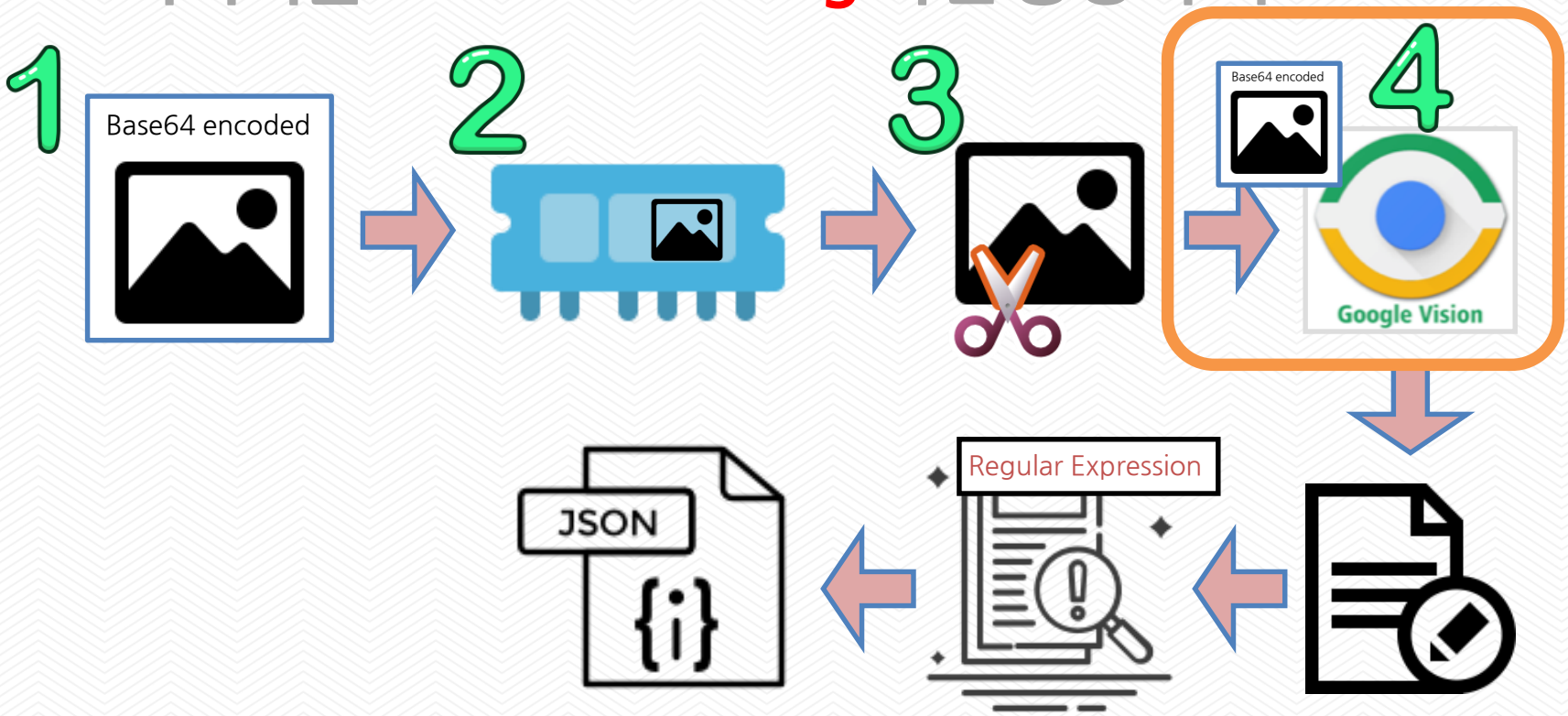
한 걸음씩 가기

- 편집한 이미지를 Vision API 호출, 결과 정제



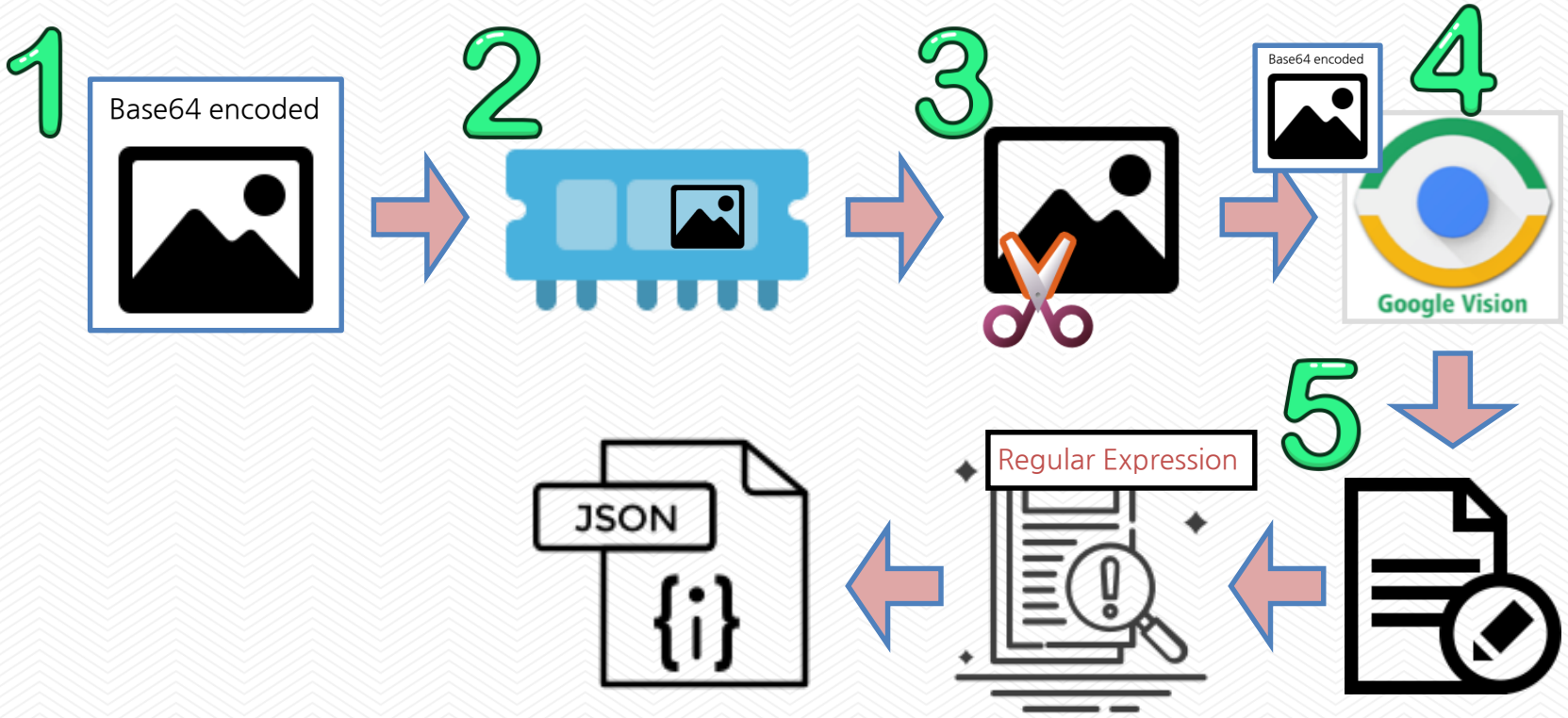
한 걸음씩 가기

- 이미지를 **Base64 encoding** 하면 용량이 약 **25%** UP



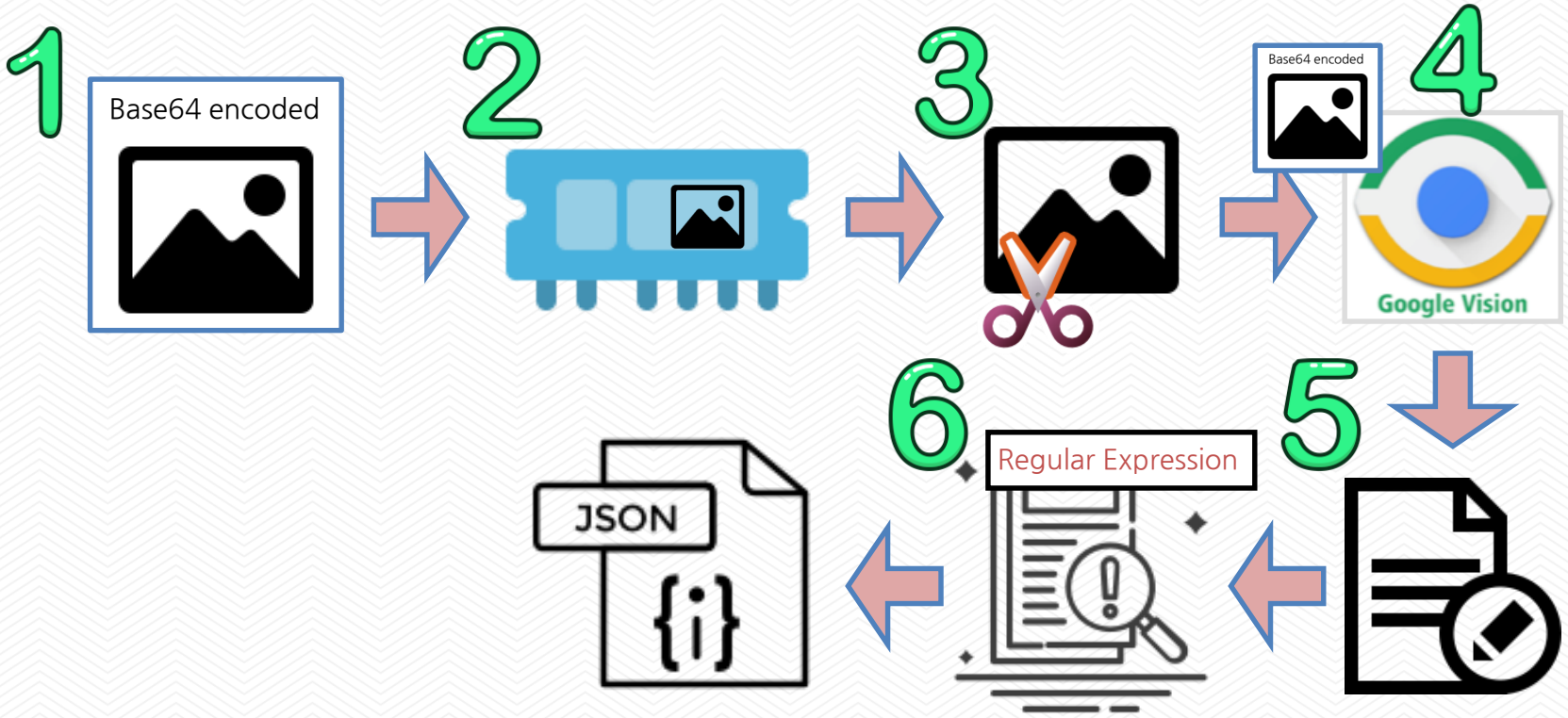
한 걸음씩 가기

- 좌표와 데이터를 이용해서 라인 보정



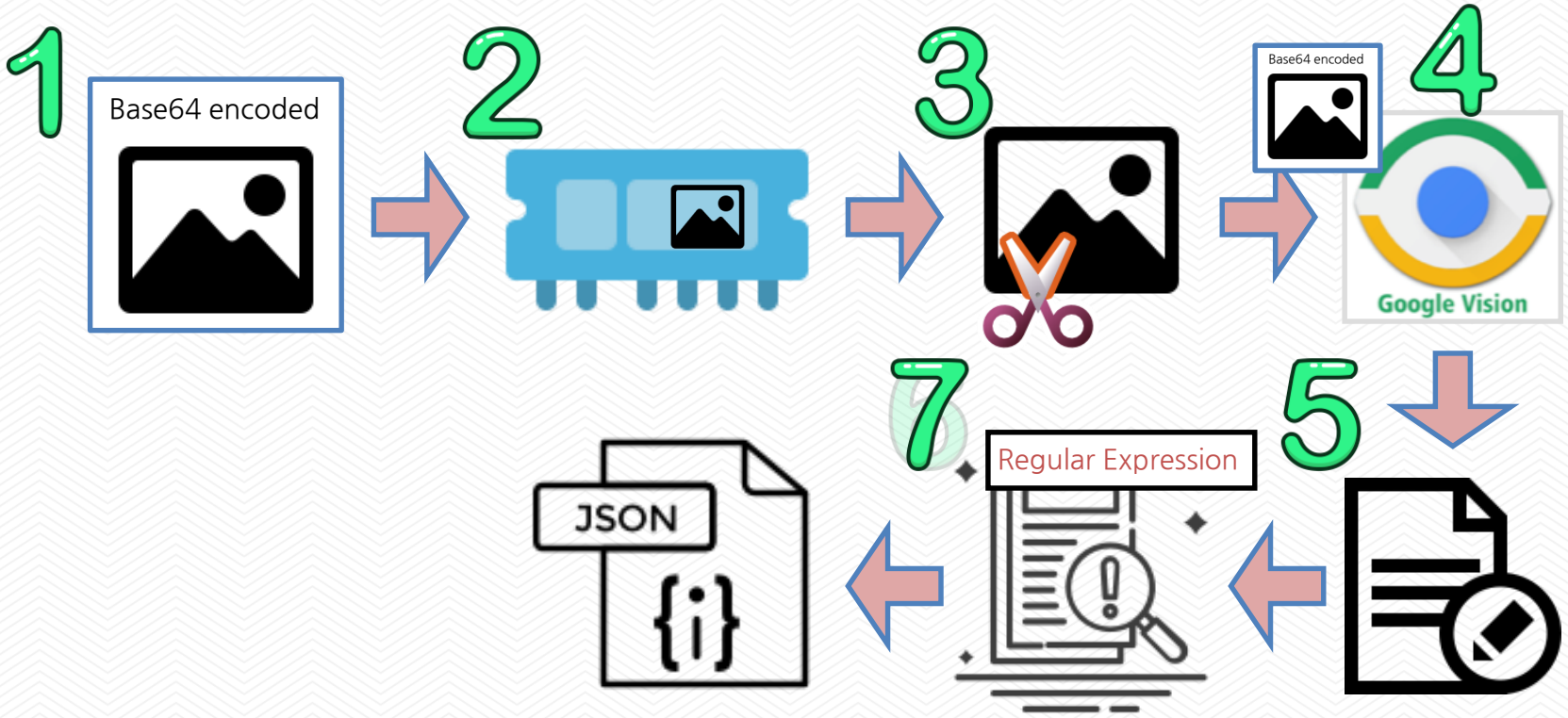
한 걸음씩 가기

- 기본 정규식을 이용한 TEXT 제거, 치환



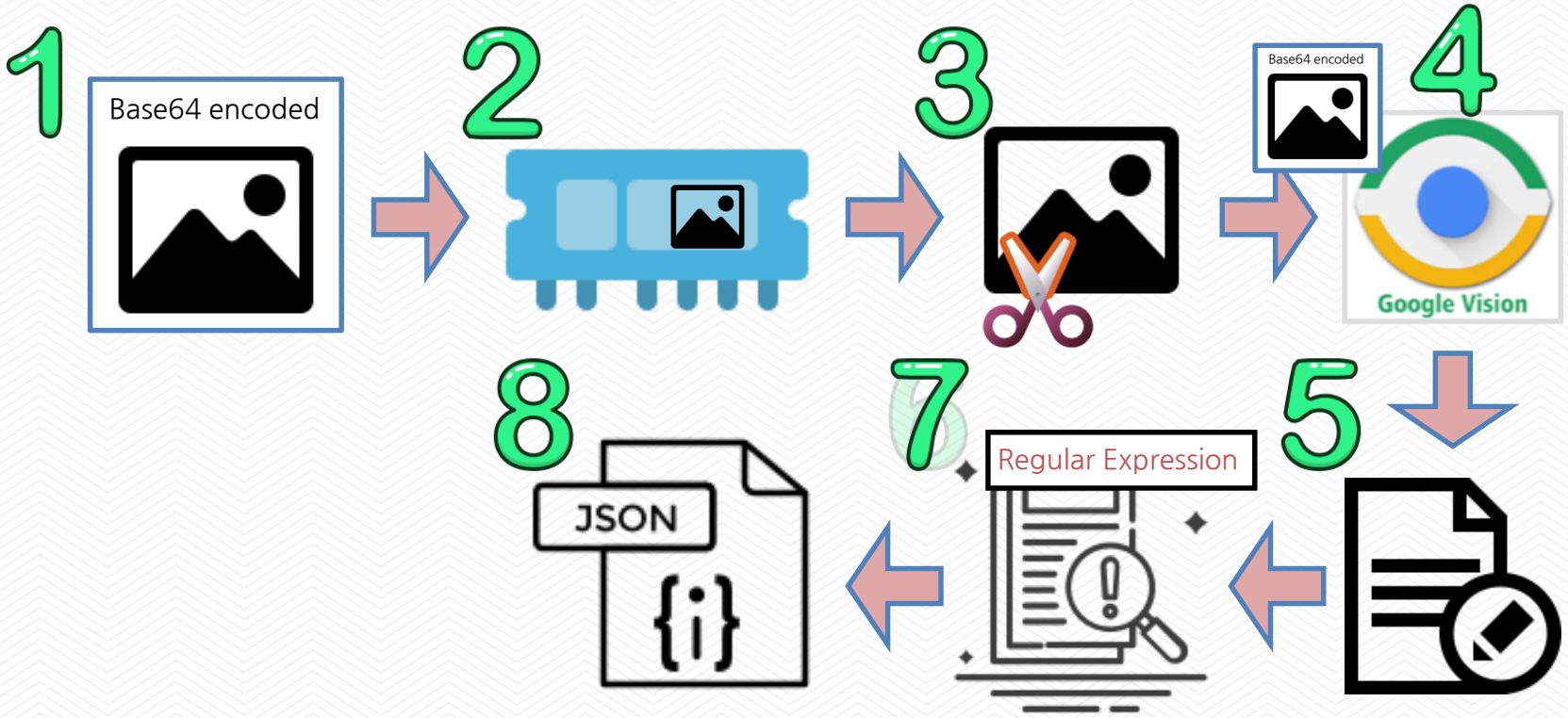
한 걸음씩 가기

- **정규식을 이용한 정보 추출** (상호, 날짜, 전화번호, 주소, 메뉴, 금액)



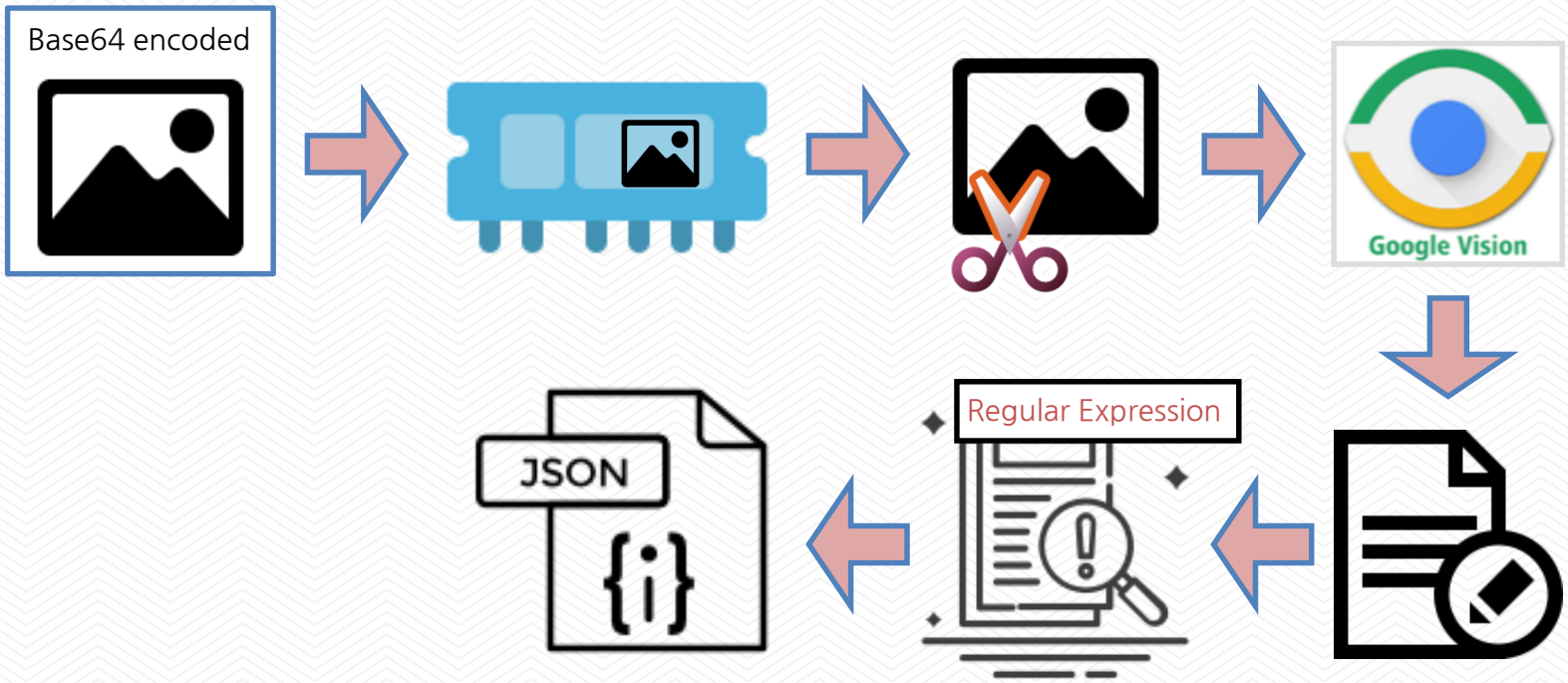
한 걸음씩 가기

- 요구 사항에 따른 Json Document 생성

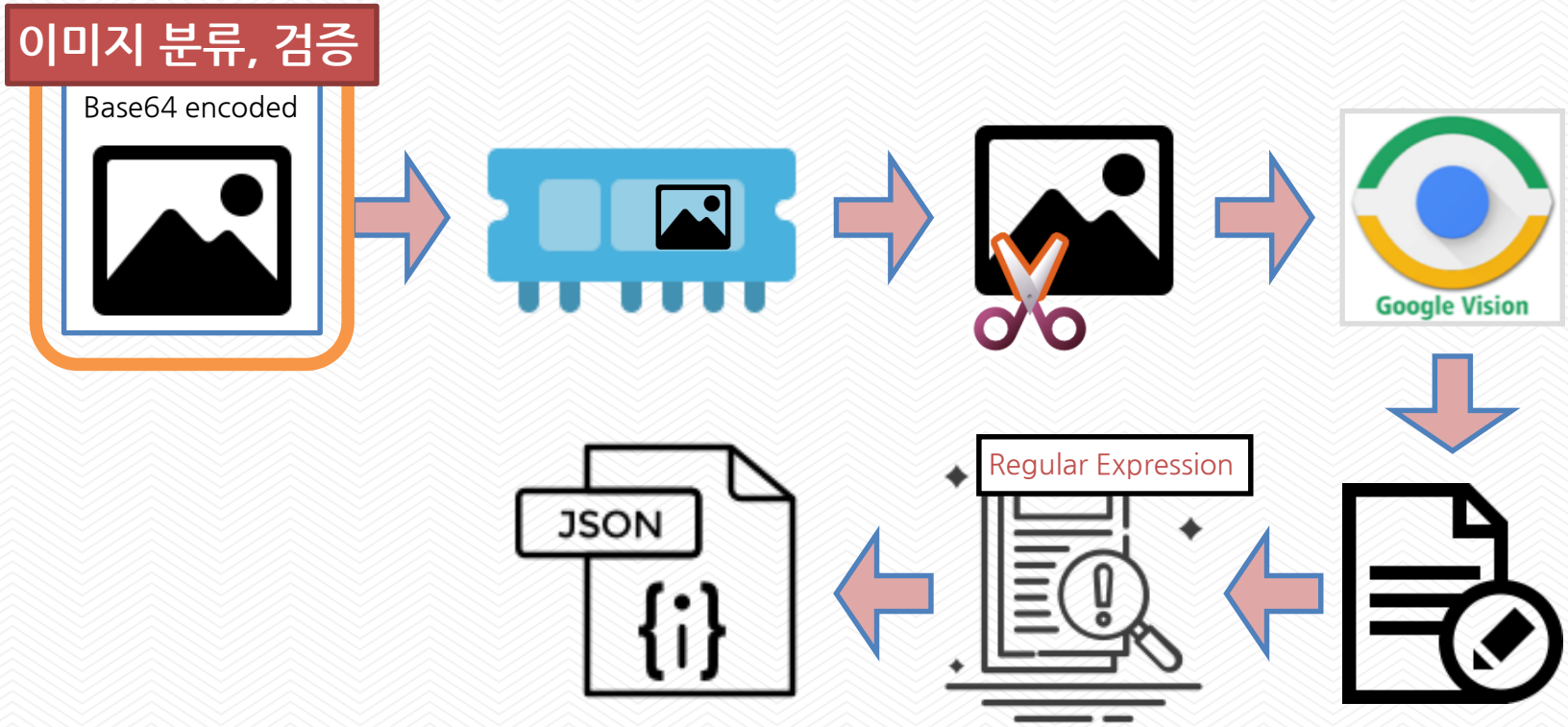


한 걸음씩 가기

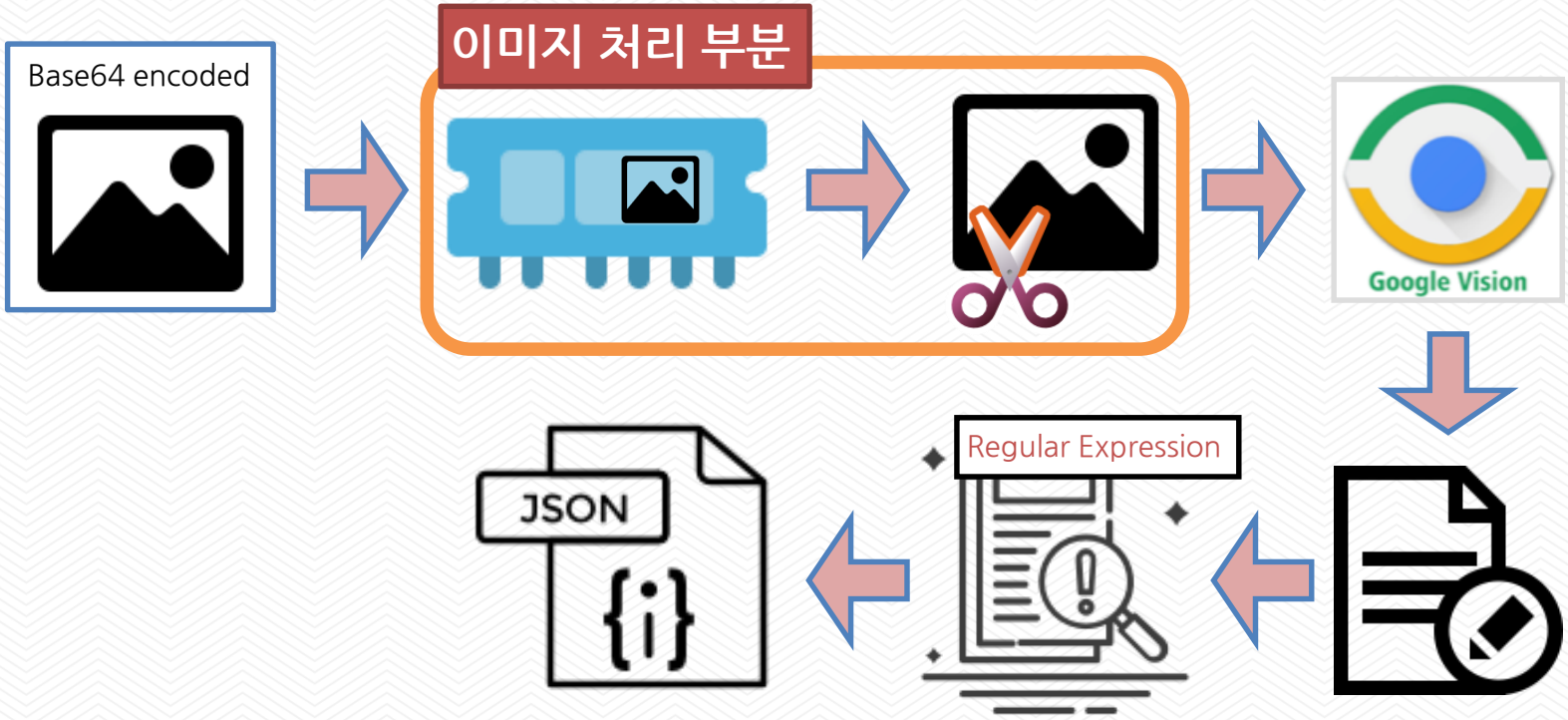
- 어떻게 분리를 할지 고민



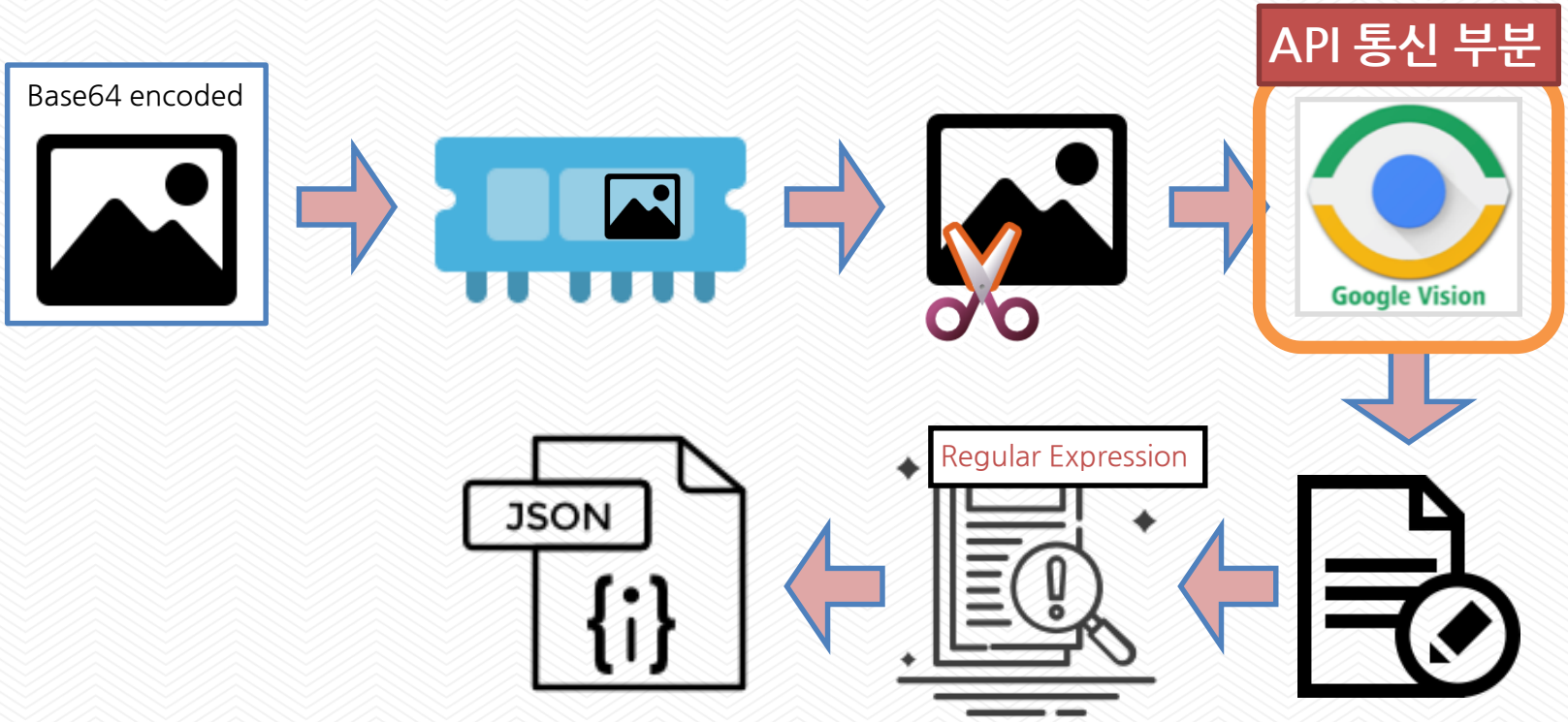
한 걸음씩 가기



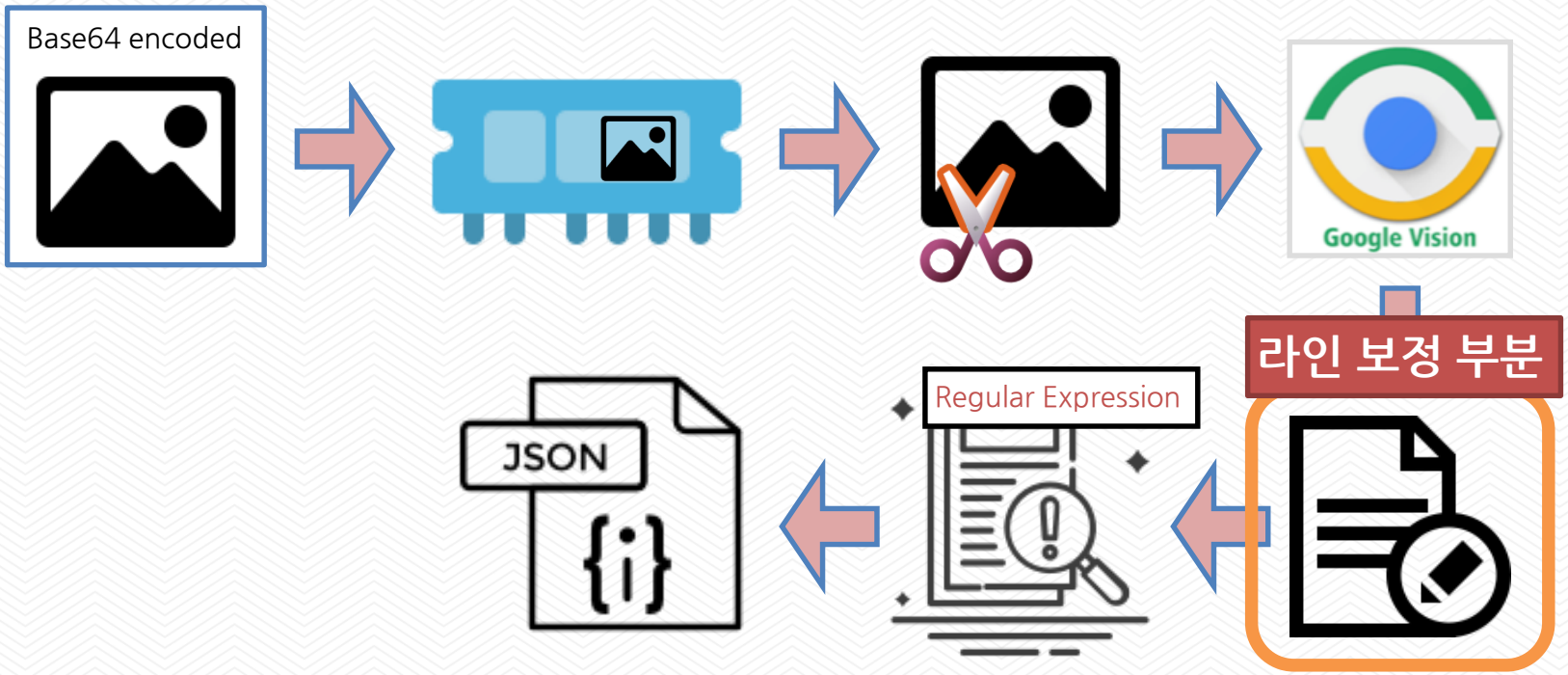
한 걸음씩 가기



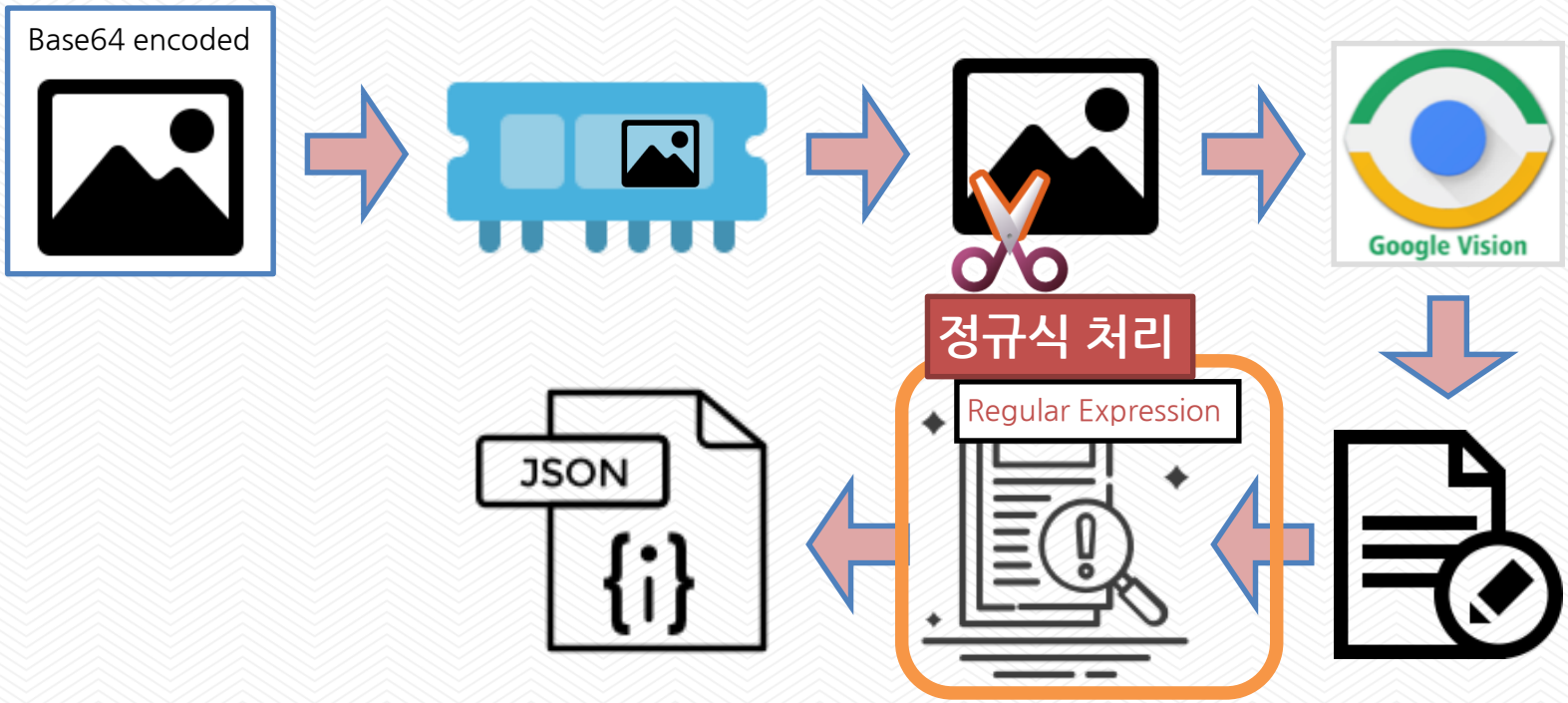
한 걸음씩 가기



한 걸음씩 가기

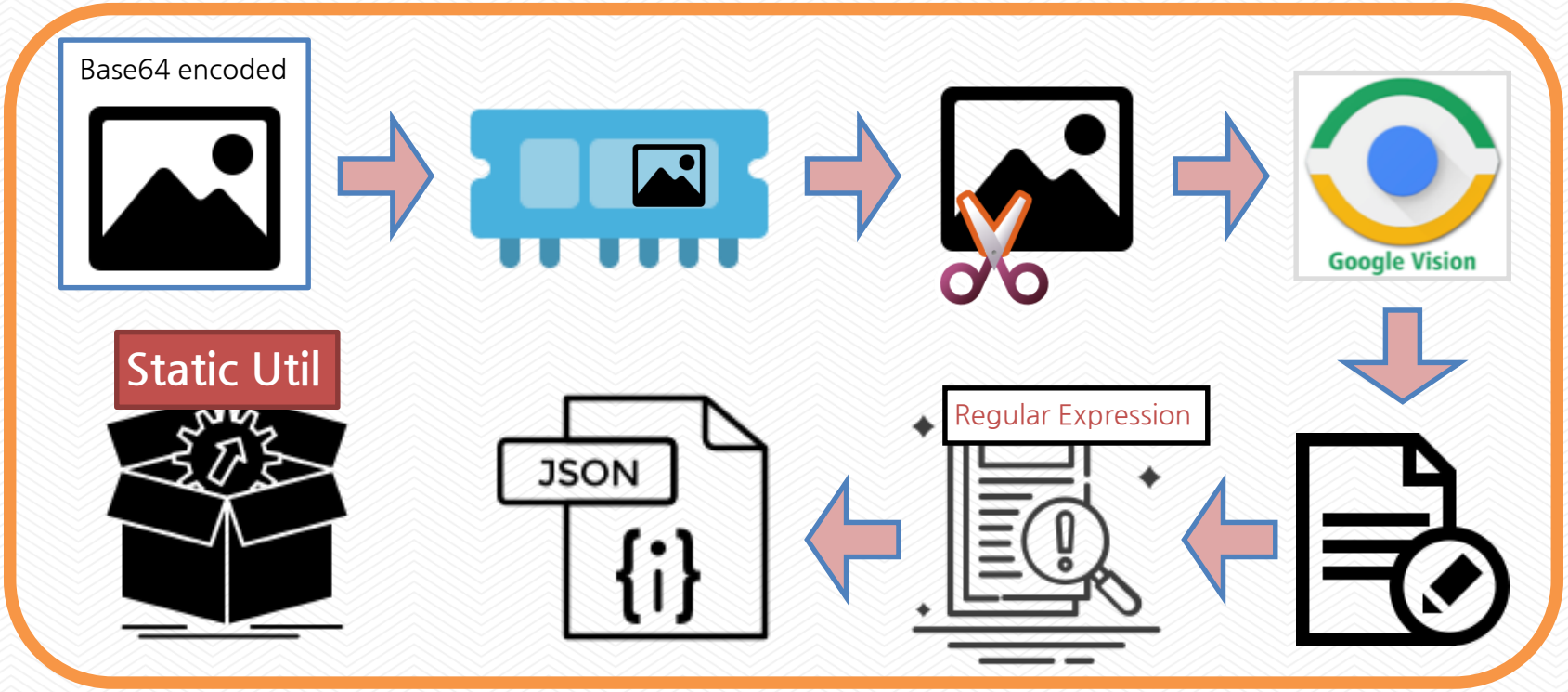


한 걸음씩 가기



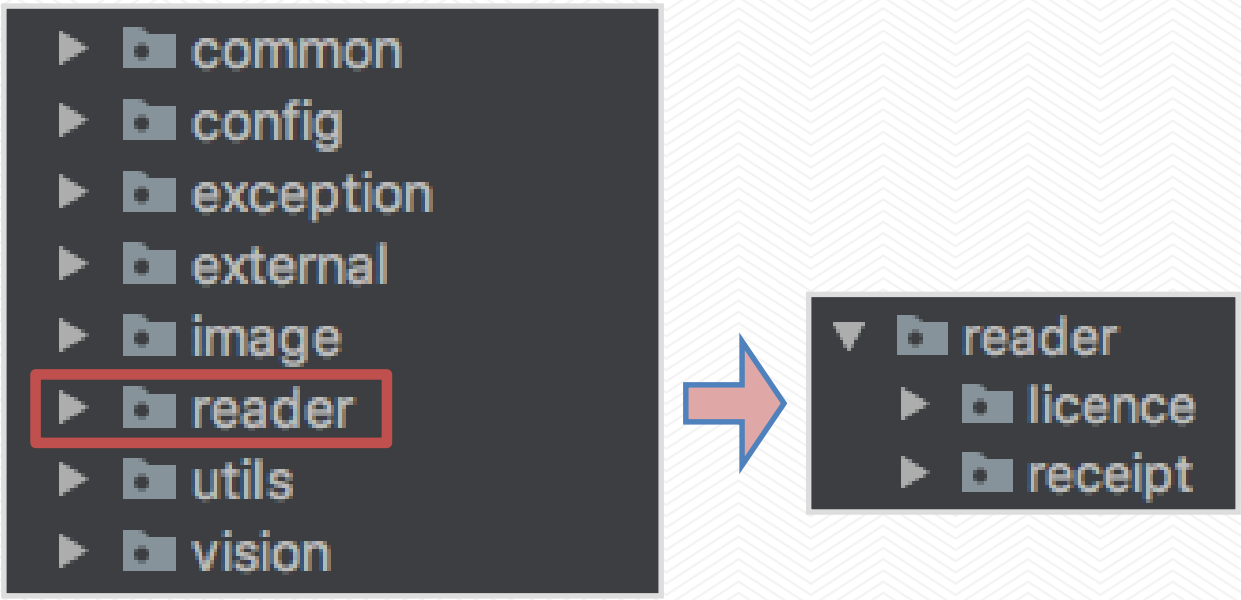
한 걸음씩 가기

- 그리고 모든 곳에서 사용되는 Static Util 부분



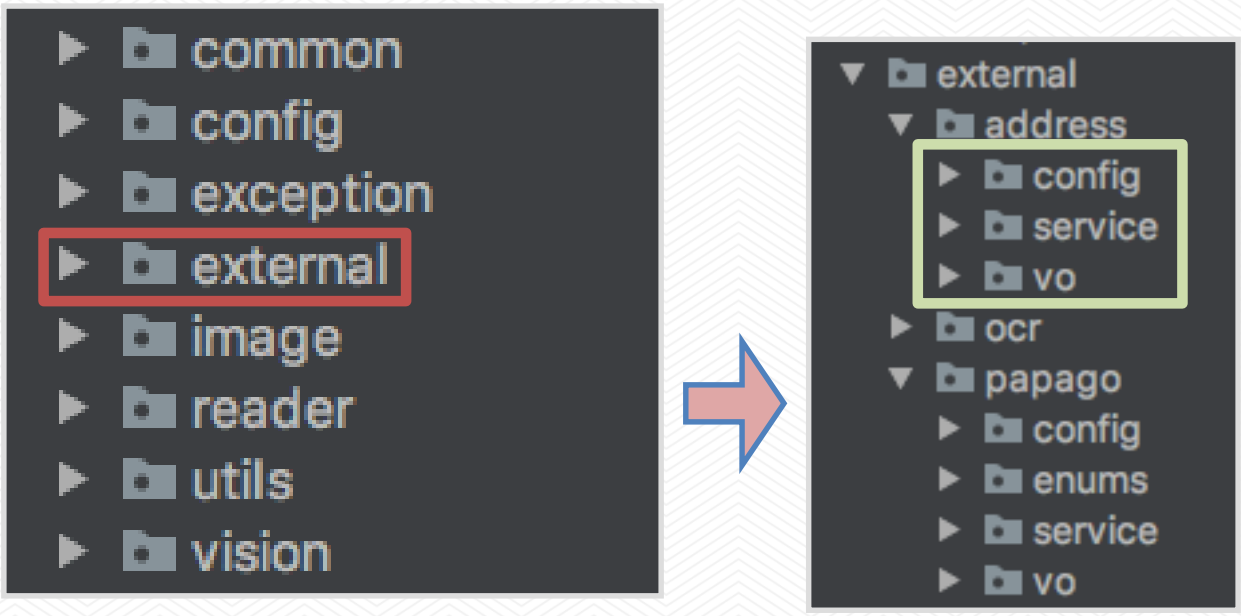
한 걸음씩 가기

- Package 구조를 결정함
- Reader 부분은 언제든 분리가 가능하도록 설계함



한 걸음씩 가기

- Package 구조를 결정함
- 모든 부분을 언제나 쉽게 분리를 하기 위해 신경을 씀



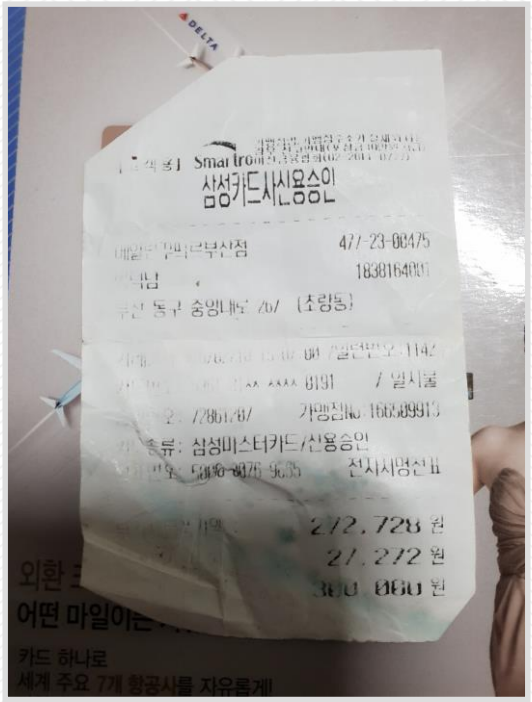
한 걸음씩 가기

- OCR의 핵심 기능...
  - ☑ 이미지 전처리
  - ☑ 정규식 후처리

이미지 출처 : https://economiology.com/핵심감사제도-

한 걸음씩 가기

- image : 이미지 전처리 모듈
  - ☑ 다양한 상태의 영수증이 들어옴



한 걸음씩 가기

- image : 이미지 전처리 모듈
  - OpenCV를 이용한 영수증 영역 검출, 이미지 편집
  - Tensorflow 학습 데이터를 이용한 각도 보정
  - 학습데이터 수집을 위한 이미지 로깅

박사님

한 걸음씩 가기

- image : 이미지 전처리 모듈이 왜 필요할까요?
  - OCR 엔진이 이미지를 더 잘 읽을 수 있게 하기 위해
  - 반복적인 노가다를 통해 Magic Number를 찾아야 함

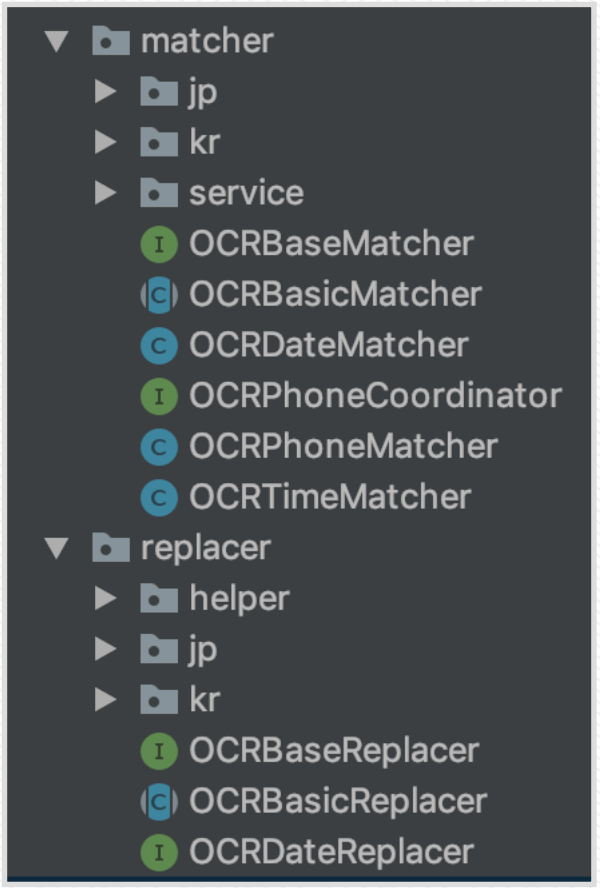
한 걸음씩 가기

- OCR의 핵심 기능...
  - ☑ 이미지 전처리
  - ☑ 정규식 후처리

이미지 출처 : https://economiology.com/핵심감사제도-

한 걸음씩 가기

- 정규식을 사용한 후처리 부분
  - 텍스트 보정 - replacer
  - 텍스트 검출 - matcher



한 걸음씩 가기

- 정규식을 사용한 후처리 부분 - Replace
  - ☑ 매우 심플한 케이스인데요...

GARDEN HOUSE

GARDEN HOUSE 新宿
2018/08/30 (木) 12:15 店No-0000001000002-0001
伝票No-00026 ﾃｰﾌﾞﾙ B03 1名

EAT-IN
L /ﾊﾟﾝｹｰｷｺﾝﾎﾞ ¥1,350
Lunch Drink ¥0
L/ﾗﾃ ¥300

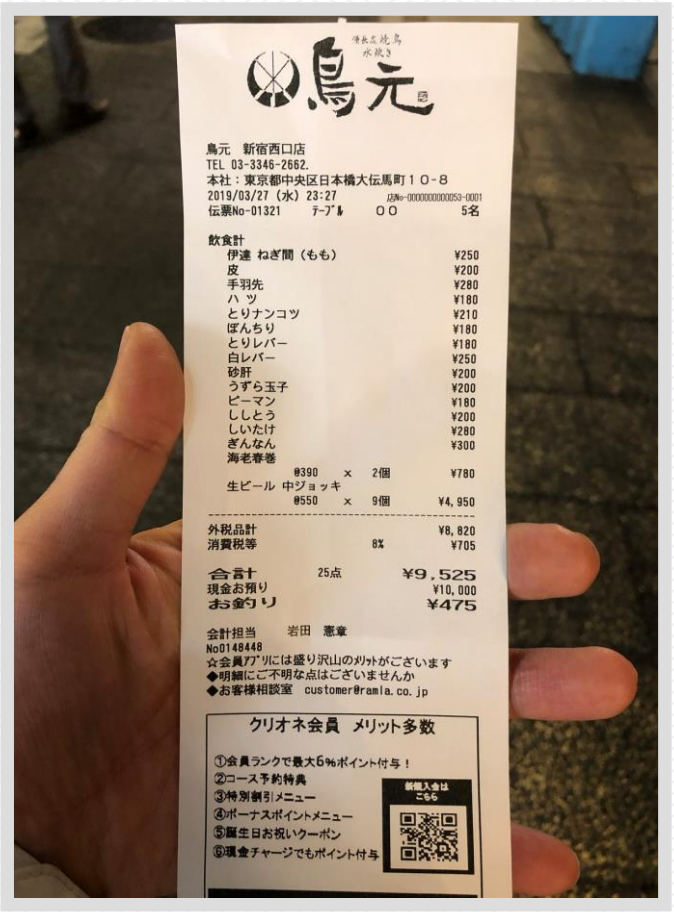
EAT-IN ¥1,650
従業員１０% 10% -¥165

合計 3点 ¥1,485
(消費税等 8% ¥110)
交通系ＩＣカード 1 ¥1,485

会計担当 牧
No0173940

한 걸음씩 가기

- 정규식을 사용한 후처리 부분 - Replace
  - ☑ 대부분 이렇게 복잡해요.



한 걸음씩 가기

- 정규식을 사용한 후처리 부분 - Replace
  - ☑ 이미지 전처리 과정 - OCR 엔진으로 호출

GARDEN HOUSE

GARDEN HOUSE 新宿
2018/08/30 (木) 12:15 店No-0000001000002-0001
伝票No-00026 ﾃｰﾌﾞﾙ B03 1名

EAT-IN
L /ﾊﾟﾝｹｰｷｺﾝﾎﾞ ¥1,350
Lunch Drink ¥0
L/ﾗﾃ ¥300

EAT-IN ¥1,650
従業員１０% 10% -¥165

合計 3点 ¥1,485
(消費税等 8% ¥110)
交通系ＩＣカード 1 ¥1,485

会計担当 牧
No0173940

한 걸음씩 가기

- 정규식을 사용한 후처리 부분 - Replace
  - ☑ 원본 텍스트 검출

```
GARDENHOUSE
GARDENHOUSE新宿
2018/08/30(木)12:15
伝票No-00026テーブルB031名
EAT-IN
L/パンケーキコンボ¥1,350
LunchDrink¥0
L/ラテ¥300
EAT-IN¥1,650
従業員10%10%-¥165
合計3点¥1,485
(消費税等8%¥110)
交通系ICカード1¥1,485
会計担当牧
No0173940
```

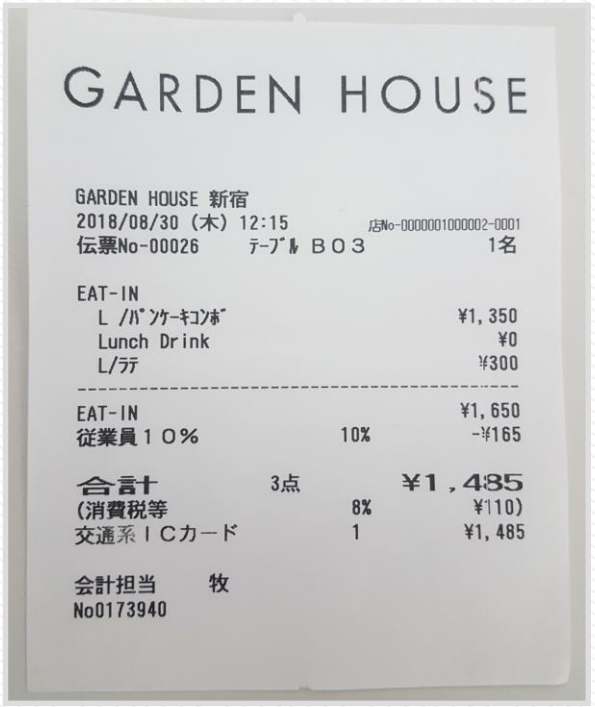
한 걸음씩 가기

정규식을 사용한 후처리 부분 - Replace

☑ 라인 보정

```
GARDENHOUSE
GARDENHOUSE新宿
2018/08/30(木)12:15
伝票No-00026テーブルB031名
EAT-IN
L/パンケーキコンボ¥1,350
LunchDrink¥0
L/ラテ¥300
EAT-IN¥1,650
従業員10%10%-¥165
合計3点¥1,485
(消費税等8%¥110)
交通系ICカード1¥1,485
会計担当牧
No0173940
```

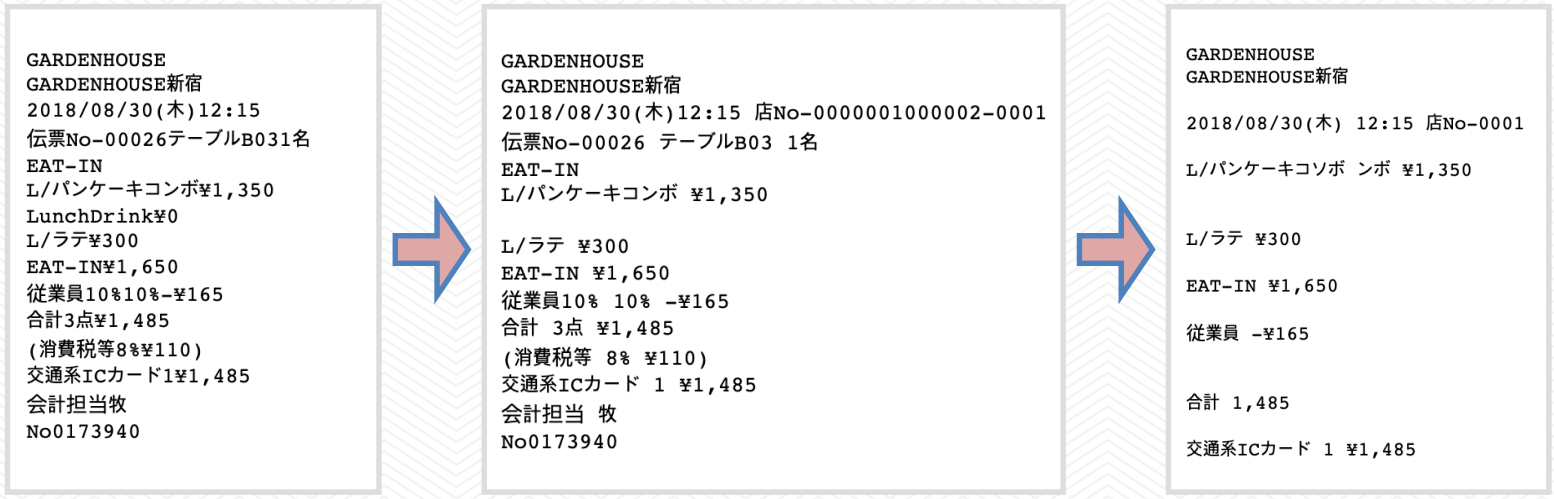
```
GARDENHOUSE
GARDENHOUSE新宿
2018/08/30(木)12:15 店No-0000001000002-0001
伝票No-00026 テーブルB03 1名
EAT-IN
L/パンケーキコンボ ¥1,350

L/ラテ ¥300
EAT-IN ¥1,650
従業員10% 10% -¥165
合計 3点 ¥1,485
(消費税等 8% ¥110)
交通系ICカード 1 ¥1,485
会計担当 牧
No0173940
```



한 걸음씩 가기

정규식을 사용한 후처리 부분 - Replace

- 필요 없는 텍스트 제거



한 걸음씩 가기

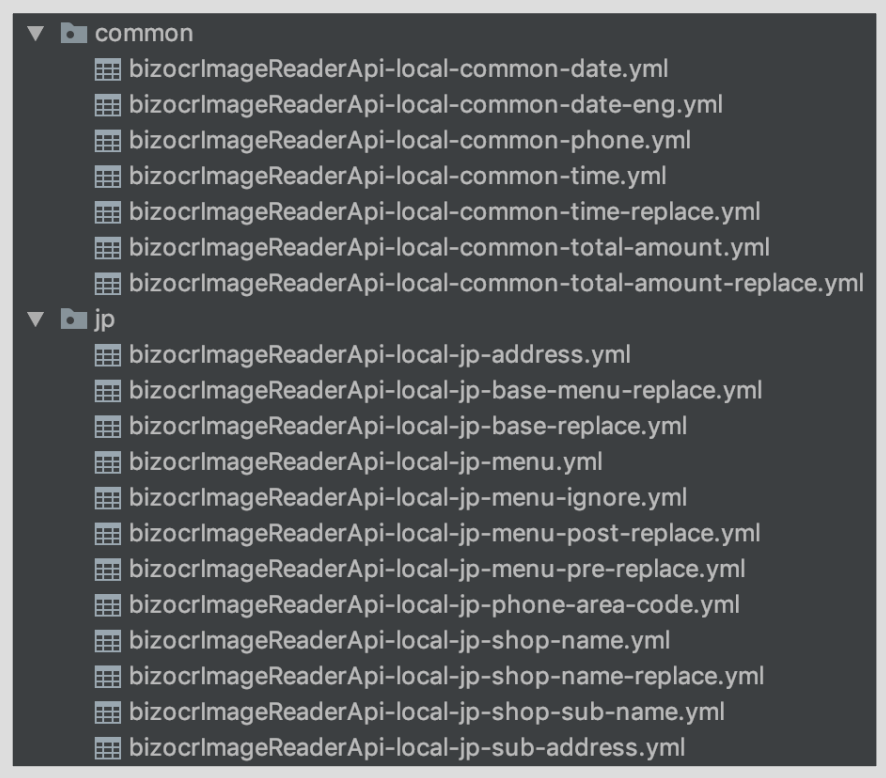
- **정규식을 사용한 검출 부분 - Matcher**
  - ☑ 숍 이름, 일시, 주소, 전화번호, 메뉴, 가격 등을 추출

```
GARDENHOUSE
GARDENHOUSE新宿

2018/08/30(木) 12:15 店No-0001

L/パンケーキコソボ ンボ ¥1,350

L/ラテ ¥300

EAT-IN ¥1,650

従業員 -¥165

合計 1,485

交通系ICカード 1 ¥1,485
```

result data	
name	GARDENHOUSE
date	2018-08-30
time	12:15
totalAmount	1485
menuItem	L/パンケーキコソボ ンボ : 1350 (count : 1) ,L/ラテ : 300 (count : 1)

한 걸음씩 가기

- 정규식을 분야별로 구분하여 관리함



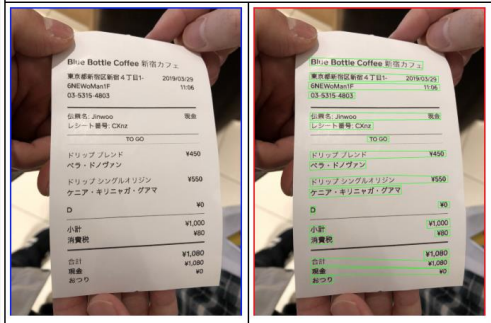
한 걸음씩 가기

- 편한 유지 보수와 개발을 위해 디버깅 툴을 만듬

일본 식당 영수증 OCR 판독 및 보정

앵글 보정 / 줄맞춤 보정 / 줄단위 그룹 / 띄어쓰기 보정 / 단어 보정(예: 言十 -> 計)이 들어갑니다.

전송 이미지



OCR 인식 (clova)	라인 보정	문구 보정	한국어 번역 결과(papago)	추출 항목
BlueBottleCoffee新宿カフェ 東京都新宿区新宿4丁目1-2019/03/29 6NEWoMan1F11:06 03-5315-4803 伝票名:Jinwoo現金 レシート番号:CXnz TOGO ドリップブレンド¥450 ベラ·ドノヴァン ドリップシングルオリジン¥550 ケニア·キリニャガ·グアマ ¥0 D ¥1,000 小計¥80 消費税 ¥1,080 合計¥1,080 現金¥0 おつり	BlueBottleCoffee新宿カフェ 東京都新宿区新宿4丁目1- 2019/03/29 6NEWoMan1F 11:06 03-5315-4803 伝票名:Jinwoo 現金 レシート番号:CXnz TOGO ドリップブレンド ¥450 ベラ·ドノヴァン ドリップシングルオリジン ¥550 ケニア·キリニャガ·グアマ ¥1,000 小計 ¥80 消費税 ¥1,080 合計 ¥1,080 おつり	BlueBottleCoffee新宿カフェ 東京都新宿区新宿4丁目1 2019/03/29 6 NEWoMan1F 11:06 TEL03-5315-4803 レシート番号:CXnz TOGO ドリップブレンド ¥450 ベラ·ドノヴァン ドリップシングルオリジン ¥550 ケニア·キリニャガ·グアマ 小計 ¥80 ¥1,000 合計 1,080 おつり	BlueBottleCoffee신주쿠 카페 도쿄도 신쥬쿠 구 신주쿠 4가 1 2019/03/29 6 NEWoMan1F 11:06 TEL03-5315-4803 영수증 번호:CXnz TOGO 드립 블렌드 ¥ 450 벨라 도노우아은 드립 싱글 오리진 ¥ 550 케냐 키리나가·그아마 소계 ¥ 80 ¥ 1,000 합계 1,080 거스름 돈	Rank is 88.89 result data name: BlueBottleCoffee新宿カフェ addr1: 東京都新宿区新宿4丁目1 addr2: 6 NEWoMan1F phone: 03-5315-4803 date: 2019-03-29 time: 11:06 totalAmount: 1080 menuItem: ドリップブレンド : 450 (count : 1) ,ドリップシングルオリジン : 550 (count : 1)

한 걸음씩 가기

- 정규식 Loop 문으로 돌려 체크하는 방식
  - 어떤 정규식에 걸려서 텍스트가 변경되었는지 추적이 어려움

한 걸음씩 가기

- Local에서 복잡한 정규식 디버깅을 위해 로그를 만듬

```
@@ ##################################################################
@@ 매칭된 OCRJPBaseReplacer 패턴이 있습니다.
@@ 1. yml 타입 : replaceAll
@@ 1. 패턴 : (.*伝票.*)
@@ 1. 변경할 텍스트 : 제거대상
@@ 1. 매칭된 카운트 갯수 : 1
@@ 1. 매칭된 그룹 카운트 갯수 : 1
@@ 1. 매칭된 원본 텍스트 : 伝票名:Jinwoo{공백}現金
@@ 1. 변경된 텍스트 : 제거됨
@@ ##################################################################
@@ 매칭된 OCRJPBaseReplacer 패턴이 있습니다.
@@ 1. yml 타입 : replaceAll
@@ 1. 패턴 : \n(\d)([^\d\s\,\.pn].*)
@@ 1. 변경할 텍스트 : {줄바꿈}$1{공백}$2
@@ 1. 매칭된 카운트 갯수 : 1
@@ 1. 매칭된 그룹 카운트 갯수 : 2
@@ 1. 매칭된 원본 텍스트 : {줄바꿈}6NEWoMan1F{공백}11:06
@@ 1. 변경된 텍스트 : {줄바꿈}6{공백}NEWoMan1F{공백}11:06
@@ ##################################################################
```

한 걸음씩 가기

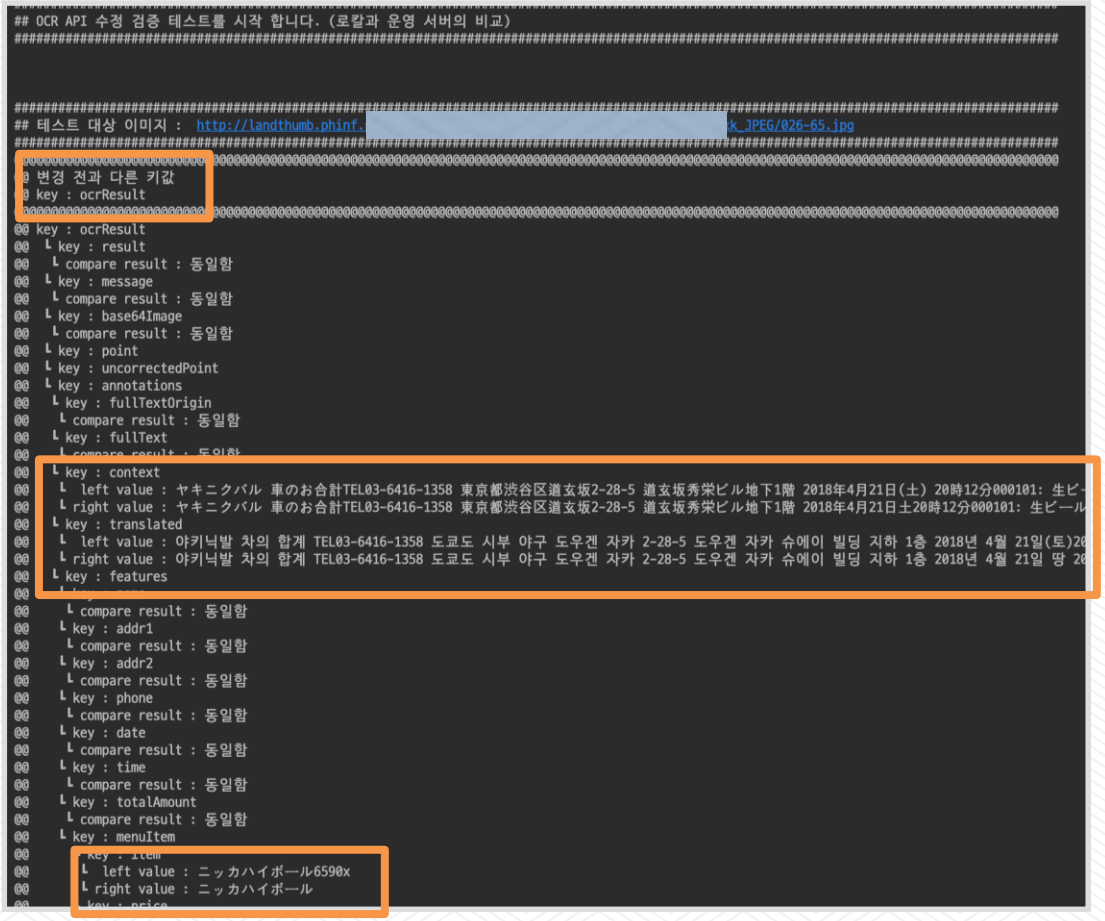
- **정규식을 수정했을 때 종종 있는 상황**
  - ☑ 전에는 잘 나왔는데 지금은 안 나와요 ~ !

한 걸음씩 가기

- Local, Stage 동일 요청 결과 비교하는 툴 개발

한 걸음씩 가기

- Local, Stage 동일 요청 결과 비교하는 툴 개발



한 걸음씩 가기

- 공통으로 사용하는 부분
  - 전화번호, 일자, 시간

한 걸음씩 가기

- 전화번호는 전 세계가 비슷한 포맷을 사용함
  - 하나의 클래스를 같이 쓰려고 만들었지만...
  - 생각보다 클래스 크기가 너무 커졌음
  - 그래서 동일한 부분은 같이 쓰고 다른 부분은 주입으로 해결

한 걸음씩 가기

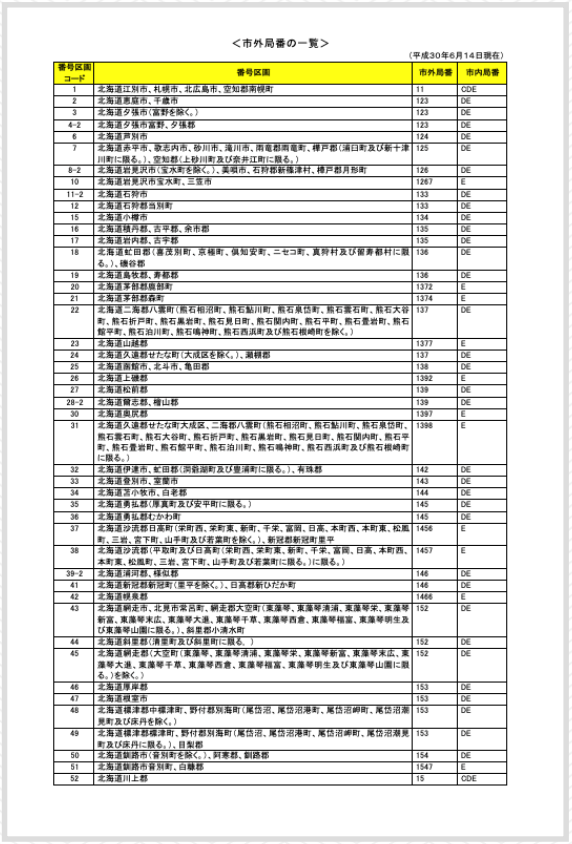
- 전화번호 중 다른 부분은 국번 (지역번호)

한 걸음씩 가기

- 한국은 생각보다 심플함
  - 지역번호가 몇 개 없음

한 걸음씩 가기

- 헐...그런데 일본은?
  - ☑ A4로 16페이지...
  - ☑ 중복제거하니 388개



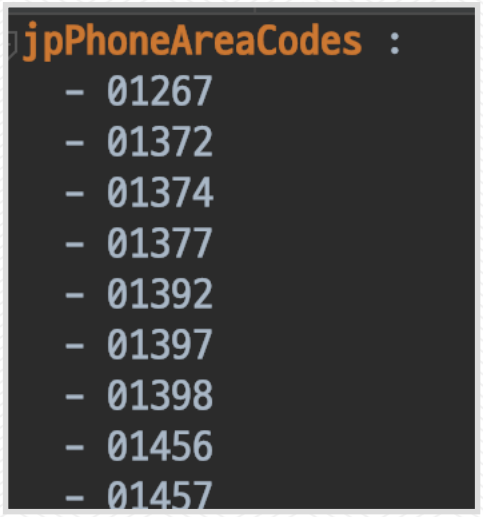
＜市外局番の一覧＞

（平成30年6月14日現在）

番号区画コード	番号区画	市外局番	市内局番
1	北海道江別市、札幌市、北広島市、空知郡南幌町	11	CDE
2	北海道恵庭市、千歳市	123	DE
3	北海道夕張市（富野を除く。）	123	DE
4-2	北海道夕張市富野、夕張郡	123	DE
6	北海道芦別市	124	DE
7	北海道赤平市、歌志内市、砂川市、滝川市、雨竜郡雨竜町、樺戸郡（浦臼町及び新十津川町に限る。）、空知郡（上砂川町及び奈井江町に限る。）	125	DE
8-2	北海道岩見沢市（宝水町を除く。）、美唄市、石狩郡新篠津村、樺戸郡月形町	126	DE
10	北海道岩見沢市宝水町、三笠市	1267	E
11-2	北海道石狩市	133	DE
12	北海道石狩郡当別町	133	DE
15	北海道小樽市	134	DE
16	北海道積丹郡、古平郡、余市郡	135	DE
17	北海道岩内郡、古宇郡	135	DE
18	北海道虻田郡（喜茂別町、京極町、倶知安町、ニセコ町、真狩村及び留寿都村に限る。）、磯谷郡	136	DE
19	北海道虻田郡、寿都郡	136	DE
20	北海道茅部郡鹿部町	1372	E
21	北海道茅部郡森町	1374	E
22	北海道二海郡八雲町（熊石相沼町、熊石鮎川町、熊石泉沢町、熊石雲石町、熊石大谷町、熊石折戸町、熊石黒岩町、熊石見日町、熊石関内町、熊石平町、熊石畳岩町、熊石館平町、熊石泊川町、熊石鳴神町、熊石西浜町及び熊石根崎町を除く。）	137	DE
23	北海道山越郡	1377	E
24	北海道久遠郡せたな町（大成区を除く。）、瀬棚郡	137	DE
25	北海道函館市、北斗市、亀田郡	138	DE
26	北海道上磯郡	1392	E
27	北海道松前郡	139	DE
28-2	北海道檜山郡、檜山郡	139	DE
30	北海道爾志郡	1397	E
31	北海道久遠郡せたな町大成区、二海郡八雲町（熊石相沼町、熊石鮎川町、熊石泉沢町、熊石雲石町、熊石大谷町、熊石折戸町、熊石黒岩町、熊石見日町、熊石関内町、熊石平町、熊石畳岩町、熊石館平町、熊石泊川町、熊石鳴神町、熊石西浜町及び熊石根崎町に限る。）	1398	E
32	北海道伊達市、虻田郡（洞爺湖町及び豊浦町に限る。）、有珠郡	142	DE
33	北海道登別市、室蘭市	143	DE
34	北海道苫小牧市、白老郡	144	DE
35	北海道勇払郡（厚真町及び安平町に限る。）	145	DE
36	北海道勇払郡むかわ町	145	DE
37	北海道沙流郡日高町（栄町西、栄町東、新町、千栄、富岡、日高、本町西、本町東、松風町、三岩、宮下町、山手町及び若葉町を除く。）、新冠郡新冠町里平	1456	E
38	北海道沙流郡（平取町及び日高町（栄町西、栄町東、新町、千栄、富岡、日高、本町西、本町東、松風町、三岩、宮下町、山手町及び若葉町に限る。）に限る。）	1457	E
39-2	北海道浦河郡、様似郡	146	DE
41	北海道新冠郡新冠町（里平を除く。）、日高郡新ひだか町	146	DE
42	北海道幌泉郡	1466	E
43	北海道網走市、北見市常呂町、網走郡大空町（東藻琴、東藻琴清浦、東藻琴栄、東藻琴新富、東藻琴末広、東藻琴大通、東藻琴千草、東藻琴西倉、東藻琴稲富、東藻琴明生及び東藻琴山園に限る。）、斜里郡小清水町	152	DE
44	北海道斜里郡（清里町及び斜里町に限る。）	152	DE
45	北海道網走郡（大空町（東藻琴、東藻琴清浦、東藻琴栄、東藻琴新富、東藻琴末広、東藻琴大通、東藻琴千草、東藻琴西倉、東藻琴稲富、東藻琴明生及び東藻琴山園に限る。）を除く。）	152	DE
46	北海道厚岸郡	153	DE
47	北海道根室市	153	DE
48	北海道標津郡中標津町、野付郡別海町（尾岱沼、尾岱沼港町、尾岱沼岬町、尾岱沼潮見町及び床丹に限る。）を除く。）	153	DE
49	北海道標津郡標津町、野付郡別海町（尾岱沼、尾岱沼港町、尾岱沼岬町、尾岱沼潮見町及び床丹に限る。）、目梨郡	153	DE
50	北海道釧路市（音別町を除く。）、阿寒郡、釧路郡	154	DE
51	北海道釧路市音別町、白糠郡	1547	E
52	北海道川上郡	15	CDE

한 걸음씩 가기

- YML 파일로 독립 후 사용

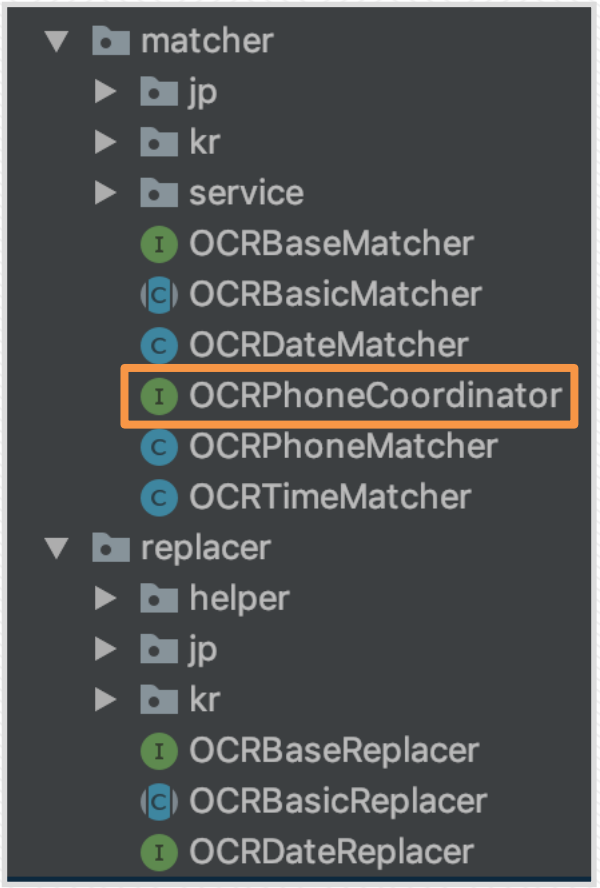


한 걸음씩 가기

- 전화번호 Class의 역할을 분리함
  - MainClass - 전화번호의 패턴만 확인 후 추출함

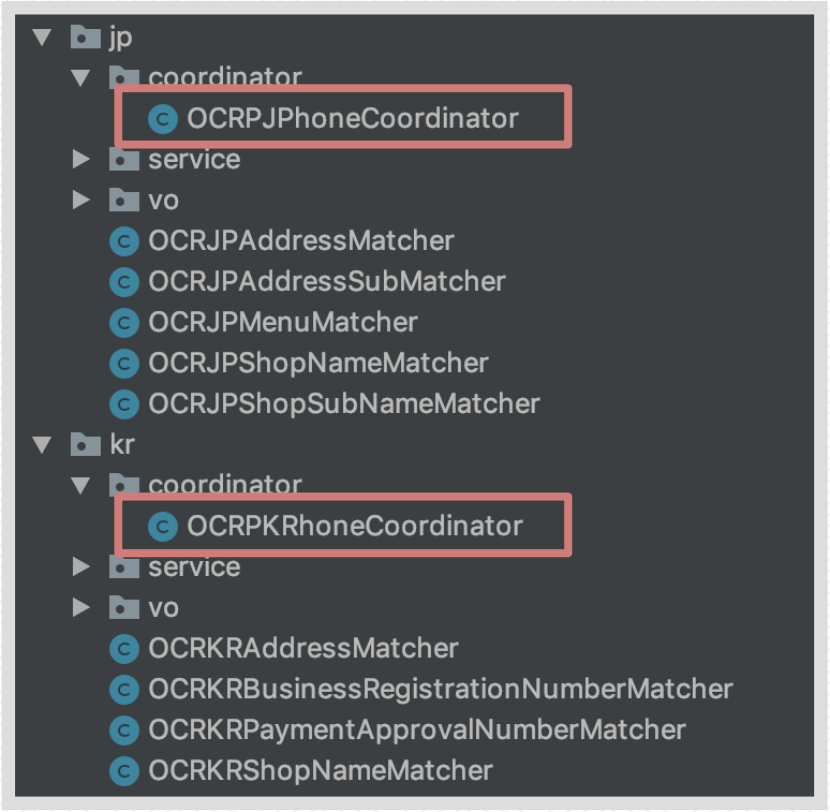
한 걸음씩 가기

- Interface를 하나 만들고...



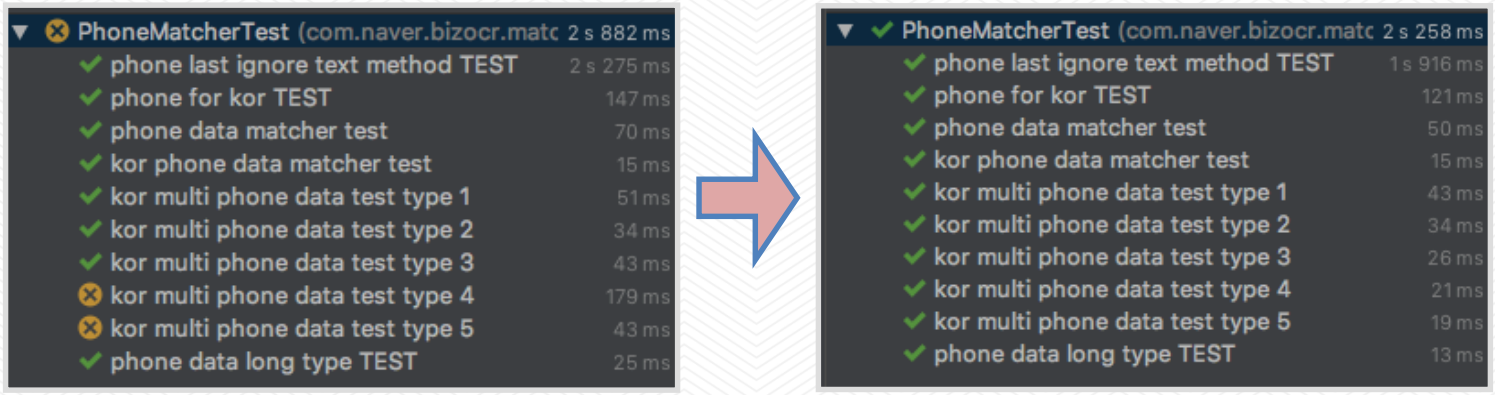
한 걸음씩 가기

- 구현 Class를 만들어줌
  - ☑ 국가별로 다른 부분을 구현하여 주입



한 걸음씩 가기

- 전화번호 Class의 역할을 분리함
  - MainClass - 전화번호의 패턴만 확인 후 추출함
  - CodinatorClass - 추출된 전화번호를 국가에 맞춰 검증 & 보정
- Testcase 작성 후 검증



한 걸음씩 가기

- 전화번호가 잘 인식되었는지 (기획자가) 검증 방법은?
  - ☑ 전화번호의 앞자리 (지역번호)가 잘 인식되었는지
  - ☑ 전화번호 포맷에 맞는지
  - ☑ 정답셋과 비교 (알바군단이 작성한...)

한 걸음씩 가기

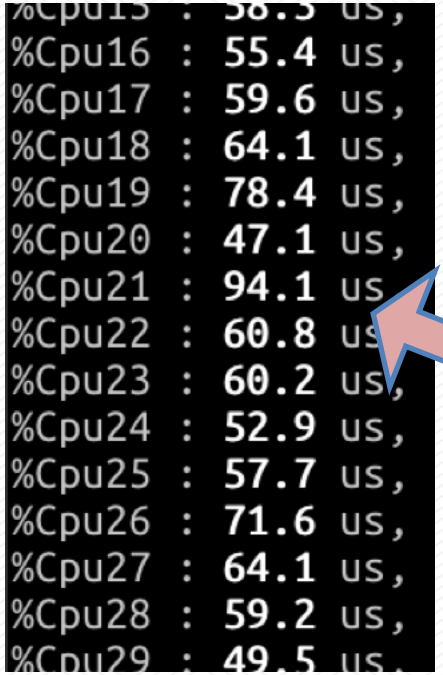
- 전화번호 인식률
  - ☑ 98% 이상
- 또 다 같이 소고기 먹었음 (삼합... 또 먹었...)

아픈 곳 고치기

- CPU 절약하기

아픈 곳 고치기

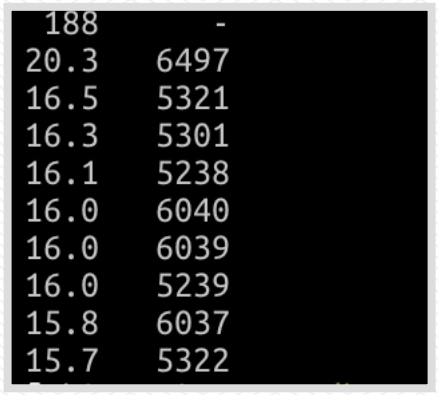
- CPU 사용량이 높은 성격의 로직




```
top - 15:11:30 up 102 days,  3:26,  1 user,  load average: 75.70, 56.83, 29.50
Tasks: 440 total,   3 running, 437 sleeping,   0 stopped,   0 zombie
%Cpu0  : 62.1 us,  8.7 sy,  0.0 ni, 29.1 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  : 57.8 us,  4.9 sy,  0.0 ni, 37.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  : 68.9 us, 10.7 sy,  0.0 ni, 20.4 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  : 51.5 us,  9.9 sy,  0.0 ni, 38.6 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  : 60.2 us,  9.7 sy,  0.0 ni, 30.1 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  : 58.3 us, 10.7 sy,  0.0 ni, 31.1 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  : 70.2 us,  4.8 sy,  0.0 ni, 24.0 id,  0.0 wa,  0.0 hi,  1.0 si,  0.0 st
%Cpu7  : 62.1 us,  8.7 sy,  0.0 ni, 29.1 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu8  : 60.2 us,  7.8 sy,  0.0 ni, 32.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu9  : 60.8 us, 10.8 sy,  0.0 ni, 28.4 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu10 : 50.5 us,  6.9 sy,  0.0 ni, 42.6 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu11 : 53.8 us,  6.7 sy,  0.0 ni, 39.4 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu12 : 56.9 us,  4.9 sy,  0.0 ni, 38.2 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu13 : 74.8 us,  4.9 sy,  0.0 ni, 20.4 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu14 : 54.4 us,  4.9 sy,  0.0 ni, 40.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu15 : 58.3 us,  5.8 sy,  0.0 ni, 35.9 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu16 : 55.4 us,  6.9 sy,  0.0 ni, 37.6 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu17 : 59.6 us,  5.8 sy,  0.0 ni, 34.6 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu18 : 64.1 us,  2.9 sy,  0.0 ni, 33.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu19 : 78.4 us,  3.9 sy,  0.0 ni, 17.6 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu20 : 47.1 us,  9.6 sy,  0.0 ni, 43.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu21 : 94.1 us,  2.9 sy,  0.0 ni,  2.9 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu22 : 60.8 us,  8.8 sy,  0.0 ni, 30.4 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu23 : 60.2 us, 10.7 sy,  0.0 ni, 29.1 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu24 : 52.9 us, 10.6 sy,  0.0 ni, 36.5 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu25 : 57.7 us,  8.7 sy,  0.0 ni, 33.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu26 : 71.6 us,  4.9 sy,  0.0 ni, 23.5 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu27 : 64.1 us,  7.8 sy,  0.0 ni, 28.2 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu28 : 59.2 us,  7.8 sy,  0.0 ni, 33.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu29 : 49.5 us,  9.5 sy,  0.0 ni, 41.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu30 : 63.5 us,  3.8 sy,  0.0 ni, 32.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu31 : 51.0 us,  4.9 sy,  0.0 ni, 44.1 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu32 : 69.6 us,  3.9 sy,  0.0 ni, 26.5 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu33 : 51.5 us,  6.9 sy,  0.0 ni, 41.6 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu34 : 58.7 us,  5.8 sy,  0.0 ni, 35.6 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu35 : 63.7 us,  4.9 sy,  0.0 ni, 31.4 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu36 : 52.5 us,  3.0 sy,  0.0 ni, 44.6 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu37 : 62.1 us,  2.9 sy,  0.0 ni, 35.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu38 : 89.3 us,  3.9 sy,  0.0 ni,  6.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu39 : 46.1 us,  4.9 sy,  0.0 ni, 49.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 13144617+total, 33662228 free, 76488496 used, 21295456 buff/cache
KiB Swap:  4195068 total,  4195068 free,        0 used. 54308520 avail Mem
```

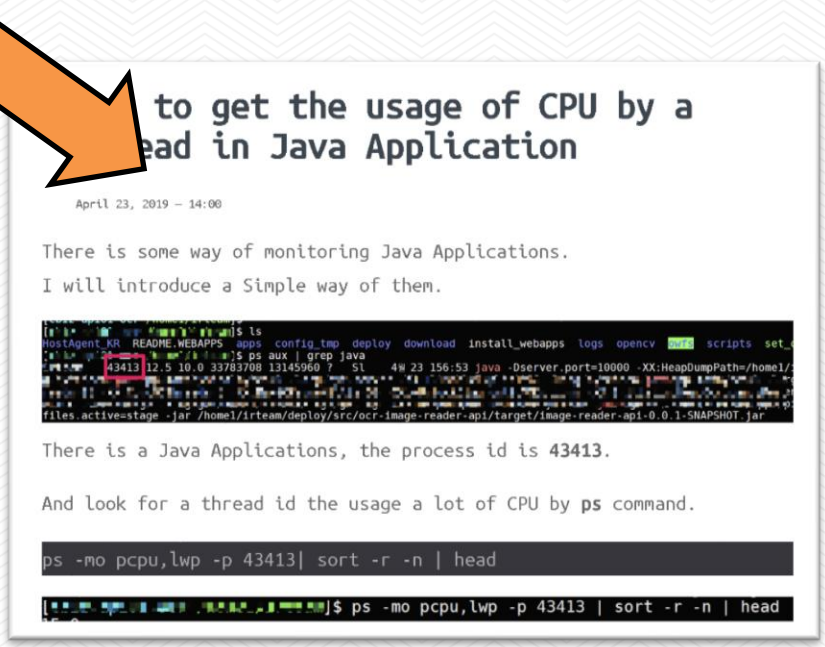
아픈 곳 고치기

- CPU 사용량이 높은 성격의 로직 (왜 이렇게 CPU를 많이 쓰지?)
  - Thread 중 CPU를 가장 많이 쓰는 순으로 추출



아픈 곳 고치기

- CPU 사용량이 높은 성격의 로직 (왜 이렇게 CPU를 많이 쓰지?)
  - Thread 중 CPU를 가장 많이 쓰는 순으로 추출
  - http://bit.ly/2Uwyhz <- 추출방법 정리해 보았습니다.



아픈 곳 고치기

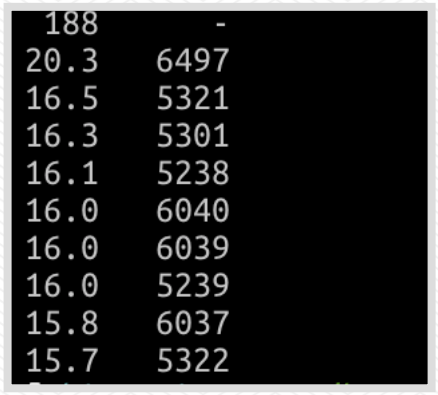
- CPU 사용량이 높은 성격의 로직 (왜 이렇게 CPU를 많이 쓰지?)
  - ☑ ThreadDump 받아서 확인

```
 188      -
20.3   6497
16.5   5321
16.3   5301
16.1   5238
16.0   6040
16.0   6039
16.0   5239
15.8   6037
15.7   5322
```

```
"http-nio-20000-exec-81" #252 daemon prio=5 os_prio=0 tid=0x00007f96c4013000 nid=
   java.lang.Thread.State: BLOCKED (on object monitor)
        at org.bytedeco.javacpp.Pointer.deallocator(Pointer.java:555)
        - waiting to lock <0x00007f99c89d3758> (a java.lang.Class for org.bytedec
        at org.bytedeco.javacpp.Pointer.init(Pointer.java:124)
        at org.bytedeco.javacpp.opencv_core$Point.allocate(Native Method)
        at org.bytedeco.javacpp.opencv_core$Point.<init>(opencv_core.java:11233)
```

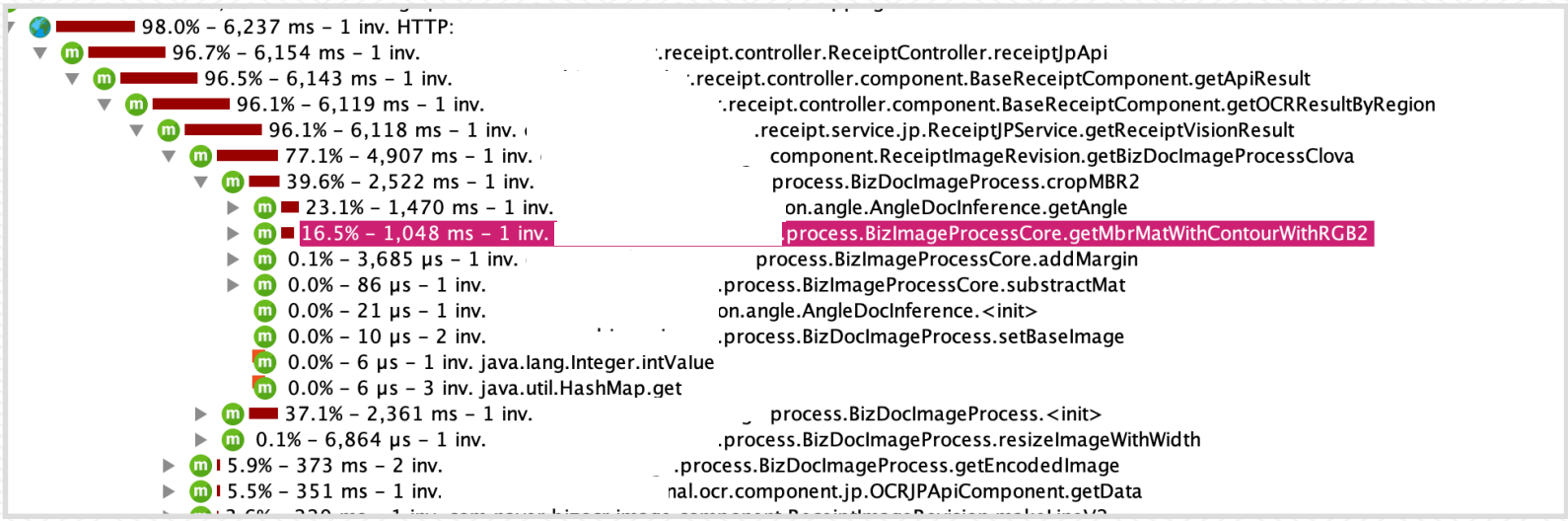
아픈 곳 고치기

- CPU 사용량이 높은 성격의 로직 (왜 이렇게 CPU를 많이 쓰지?)
  - OpenCV + Tensorflow 의 이미지 편집



아픈 곳 고치기

- CPU 사용량이 높은 성격의 로직 (왜 이렇게 CPU를 많이 쓰지?)
  - ☑ 보다 정확하게 Profiling을 해봅니다.



아픈 곳 고치기

- CPU 사용량이 높은 성격의 로직 (왜 이렇게 CPU를 많이 쓰지?)
  - ☑ 당장 할 수 있는 게 없어... (매우 안타까웠습니다.)

아픈 곳 고치기

- 제가 짠 코드를 다시 살펴보기 시작했죠.
  - ☑ 팀장님에게 CPU 떨어트리겠다고 호언장담을...

칼을 뽑았으면
무라도 썰자

아픈 곳 고치기

- 정규식을 순회하는 부분에서 Pattern 객체 생성 부분
  - 순회를 할 때마다 객체를 새로 만드는 문제

```
for (Map.Entry<String, String> entry : this.patternMap.entrySet(
    String patternString = entry.getKey();

    if (patternString.length() > 1 && "\\[".equals(patternString
    final Pattern pattern = Pattern.compile(patternString);

    String[] indexNoList = entry.getValue().split( regex: "\\$");
    for (int i=0; i<indexNoList.length; i++) {...}
```

아픈 곳 고치기

- Local Cache Map을 만들어서 해결함
  - 정규식 패턴은 변경되지 않기 때문에...
  - CPU 사용량 그다지 차이는 크지 않음 (티끌 모아 태산이라고 주장함)

	OCR_API_T...		이경일	ChunC...	2018-11-04 01:35	01:00:00	19.8
	OCR_API_T...		이경일	ChunC...	2018-11-03 23:40	01:00:00	19.9
	OCR_API_T...		이경일	ChunC...	2018-11-03 21:13	01:00:00	18.8
	OCR_API_T...		이경일	ChunC...	2018-11-03 21:05	00:05:00	18.1

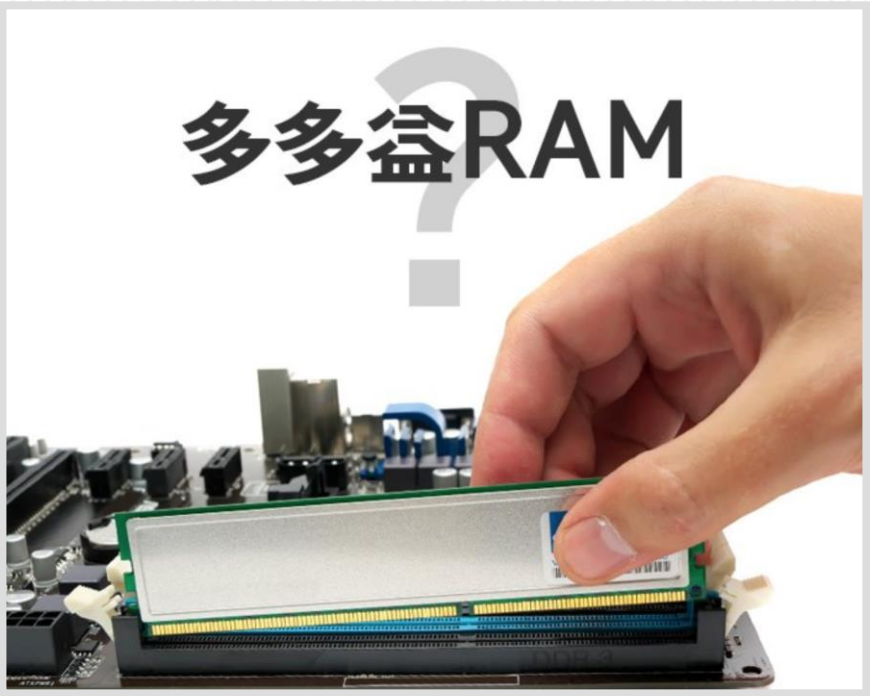
```
Pattern checkPattern = BASIC_MATCHER_PATTERN_CACHE_MAP.get(patternString);
if (checkPattern == null) {
    checkPattern = Pattern.compile(patternString);
    BASIC_MATCHER_PATTERN_CACHE_MAP.put(patternString, checkPattern);
}
final Pattern pattern = checkPattern;
```

아픈 곳 고치기

- Memory 절약하기

아픈 곳 고치기

- 서버의 램은 128GB를 장착한 고사양의 서버
  - JVM memory는 32G를 사용...



이미지 출처 : https://m.post.naver.com/viewer/postView.nhn?volumeNo=6647722&memberNo=36047368

아픈 곳 고치기

- 아니 왜 이렇게 램을 많이 쓰지?
  - ☑ HeapDump를 받아 분석
  - ☑ Memory Leak이 발견됨

Problem Suspect 1

853,771 instances of **"org.bytedeco.javacpp.Pointer$NativeDeallocator"** loaded by **"org.springframework.boot.loader.LaunchedURLClassLoader @ 0x7f44912870c0"** occupy **130,506,216 (71.44%)** bytes. These instances are referenced from one instance of **"java.lang.Object[]"**, loaded by **"<system class loader>"**

Keywords
org.springframework.boot.loader.LaunchedURLClassLoader @ 0x7f44912870c0
java.lang.Object[]
org.bytedeco.javacpp.Pointer$NativeDeallocator

Details »

아픈 곳 고치기

- 좀 더 디테일한 분석을 위해서 Profiling을 시작함
  - OpenCV의 객체가 많은 메모리 점유
  - Java 환경에서 처음 사용해봐서 발생한 문제

Class Name	Objects	Shallow Heap	Retained Heap
<Regex>	<Numeric>	<Numeric>	<Numeric>
org.bytedeco.javacpp.Pointer$NativeDeallocator	853,771	81,962,016	>= 130,506,216
org.bytedeco.javacpp.opencv_core$Point	290,238	16,253,328	>= 16,253,328
char[]	77,818	13,037,984	>= 13,037,984
byte[]	6,289	10,388,216	>= 10,388,216
org.bytedeco.javacpp.opencv_core$Rect	181,175	10,145,800	>= 10,145,800
org.bytedeco.javacpp.opencv_core$Size	169,322	9,482,032	>= 9,482,032
org.bytedeco.javacpp.opencv_core$Mat	92,013	5,888,832	>= 5,888,864
java.util.concurrent.ForkJoinTask[]	62	4,064,720	>= 4,064,720
org.bytedeco.javacpp.opencv_core$Moments	72,159	4,040,904	>= 4,040,904

아픈 곳 고치기

- OpenCV를 사용해서 생성한 객체는 (Mat 객체)
  명시적인 Release가 필요함... (아니 이게 무슨…)

```
for(int i = 0; i<contourCount; i++ ) {
    opencv_core.Mat contour = contours.get(i);
    opencv_core.Rect rect = opencv_imgproc.boundingRect(contour);
    if ((rect.size().width() > contourMinBoxSize || rect.size().heigh
            && (rect.area() < (slicedMat.size().width() * slicedMat.s
        // 중략
    }
    contour.release();
    rect.deallocate();
}
```

아픈 곳 고치기

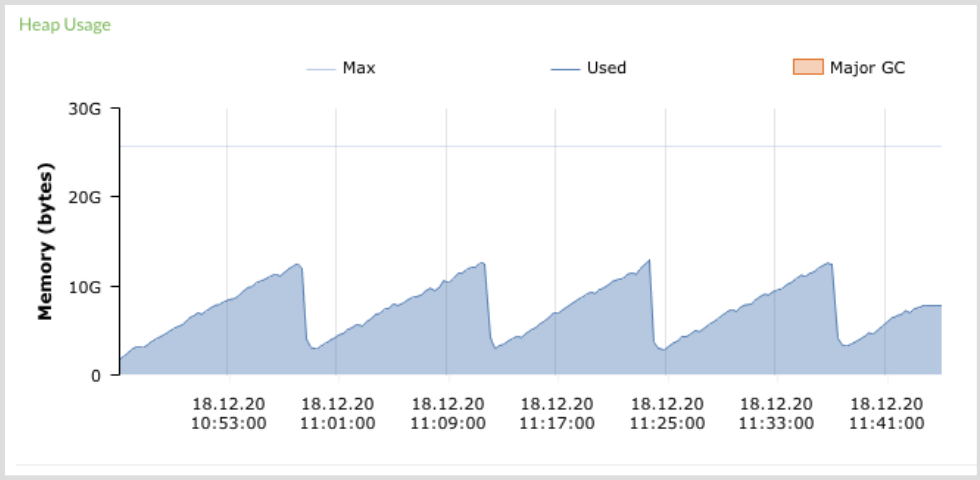
- 동일한 조건에서 테스트 한 결과
  - 많이 좋아짐...

Objects	Shallow Heap	Retained Heap
<Numeric>	<Numeric>	<Numeric>
853,771	81,962,016	>= 130,506,216
290,238	16,253,328	>= 16,253,328
77,818	13,037,984	>= 13,037,984
6,289	10,388,216	>= 10,388,216
181,175	10,145,800	>= 10,145,800
169,322	9,482,032	>= 9,482,032

Class Name	Objects	Shallow Heap	Retained Heap
<Regex>	<Numeric>	<Numeric>	<Numeric>
org.bytedeco.javacpp.Pointer$NativeDeallocator	420,824	26,932,736	>= 47,284,792
char[]	80,073	11,831,128	>= 11,831,128
org.bytedeco.javacpp.opencv_core$Size	179,288	8,605,824	>= 8,605,824
byte[]	6,384	5,453,200	>= 5,453,200
org.bytedeco.javacpp.opencv_core$Rect	113,494	5,447,712	>= 5,447,712
org.bytedeco.javacpp.opencv_core$Point	85,341	4,096,368	>= 4,096,368

아픈 곳 고치기

- 여기까지 작업해서 32G -> 24G까지 낮춤...
  - 그래도 너무 Memory를 많이 쓰고 있음...
  - HeapMemory 그래프를 보면... (엄청 쓰고 있음 덜덜덜)



아픈 곳 고치기

- 다시 Profiling으로 해봄...
  - char, byte Array 용량이... (이건 뭐지?)

Name	Instance Count	Size
char[ ]	270,891	232 MB
byte[ ]	51,427	199 MB
int[ ]	82,089	43,199 kB
org.bytedeco.javacpp.Pointer$NativeDeallocator	276,842	17,717 kB
java.lang.Float	921,601	14,745 kB
float[ ]	1,812	7,402 kB
org.bytedeco.javacpp.opencv_core$Size	87,120	4,181 kB
org.bytedeco.javacpp.opencv_core$Point	82,490	3,959 kB
java.lang.String	164,591	3,950 kB
org.bytedeco.javacpp.opencv_core$Rect	69,710	3,346 kB
java.lang.Integer	187,239	2,995 kB
org.bytedeco.javacpp.opencv_core$Mat	50,242	2,813 kB
java.lang.Object[ ]	53,771	2,345 kB
boolean[ ]	7,296	1,950 kB
java.lang.reflect.Method	19,718	1,735 kB
java.lang.Class	12,258	1,370 kB
java.util.concurrent.ConcurrentHashMap$Node	40,036	1,281 kB
java.lang.StringBuilder	42,441	1,018 kB
java.util.concurrent.ForkJoinTask[ ]	30	983 kB

아픈 곳 고치기

- **문제가 있는 코드를 보면...** (별문제 없어 보이는 간단한 메서드)
  - Request 당 약 아래의 코드가 2xxx 번 정도 실행됨
  - String Size가 작지 않음
  - Return 할 때마다 새로운 문자열이 메모리에 만들어짐

```
public static String replaceAll (Pattern p, String t
    return p.matcher(txt).replaceAll(replacement);
}
```

아픈 곳 고치기

- 제가 생각하는 개선 포인트
  - 메모리 공간을 재사용 했으면 좋겠다. -> StringBuilder 사용
  - 그리고 빠르게 GC 되었으면 좋겠다.
  - Google, Github에서 쓸만한 라이브러리가 있나 검색...

아픈 곳 고치기

- benelog 님에게 코드 리뷰를 부탁드림 (Naver PaaS)

아픈 곳 고치기

- benelog 님에게 코드 리뷰를 부탁드림 (Naver PaaS)
  - 해결책이 나옴 !!! -> 등잔 밑이 어둡다.. 라고 하셨음..
  - Java의 기본 API 중 Matcher의 AppendReplacement

아픈 곳 고치기

- Matcher_AppendReplacement
  - 그런데 문제가 있음...
  - StringBuffer

```
public Matcher appendReplacement(StringBuffer sb, String replacement) {

    // If no match, return error
    if (first < 0)
        throw new IllegalStateException("No match available");

    // Process substitution string to replace group references with groups
    int cursor = 0;
    StringBuilder result = new StringBuilder();

    while (cursor < replacement.length()) {
        char nextChar = replacement.charAt(cursor);
        if (nextChar == '\\') {
            cursor++;
            if (cursor == replacement.length())
                throw new IllegalArgumentException(
                    "character to be escaped is missing");
```

아픈 곳 고치기

- Matcher_AppendReplacement
  - 그런데 문제가 있음...
  - StringBuffer -> Jdk9부터는 StringBuilder

```
public Matcher appendReplacement(StringBuilder sb, String replacement) {
    // If no match, return error
    if (first < 0)
        throw new IllegalStateException("No match available");
    StringBuilder result = new StringBuilder();
    appendExpandedReplacement(replacement, result);
    // Append the intervening text
    sb.append(text, lastAppendPosition, first);
    // Append the match substitution
    sb.append(result);
    lastAppendPosition = last;
    modCount++;
    return this;
}
```

아픈 곳 고치기

- 내친김에 OpenJDK 11로 업글... (남자라면 최신)

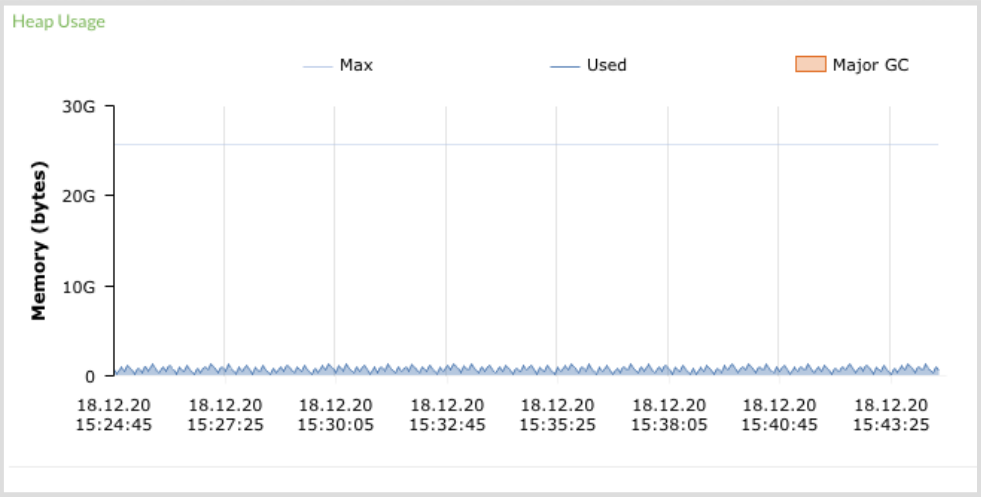
아픈 곳 고치기

- 최종 코드
  - Loop 문의 시작 전에 만들 StringBuilder를 buffer로 재사용
  - 아래의 코드는 예시 코드입니다.

```
final Matcher matcher = p.matcher(targetText);
StringBuilder tmpBufferString = new StringBuilder(bufferString.capacity());
while (matcher.find()) {
    matcher.appendReplacement(tmpBufferString, replacement);
}
matcher.appendTail(tmpBufferString);
bufferString.setLength(0);
bufferString.append(tmpBufferString);
```

아픈 곳 고치기

- 결과는?
  - ☑ 메모리 사용량 대폭 감소!!



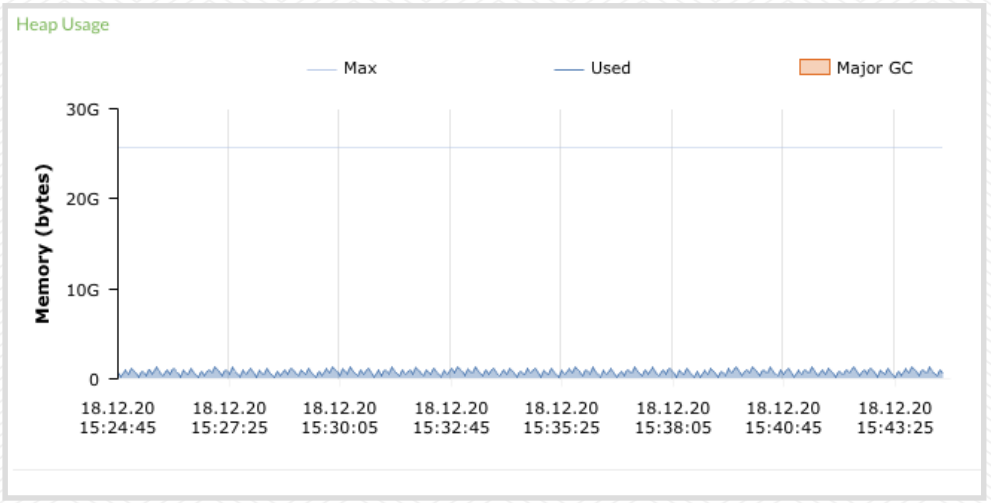
아픈 곳 고치기

- 결과는?
  - ☑ 메모리 사용량 대폭 감소!!
  - ☑ 관련해서 정리한글 -> http://bit.ly/2IOezox

아픈 곳 고치기

- 결과는?
  - ☑ 메모리 사용량 대폭 감소!!
- 또 다 같이 소고기 먹었음 (삼합... 또 먹었...)

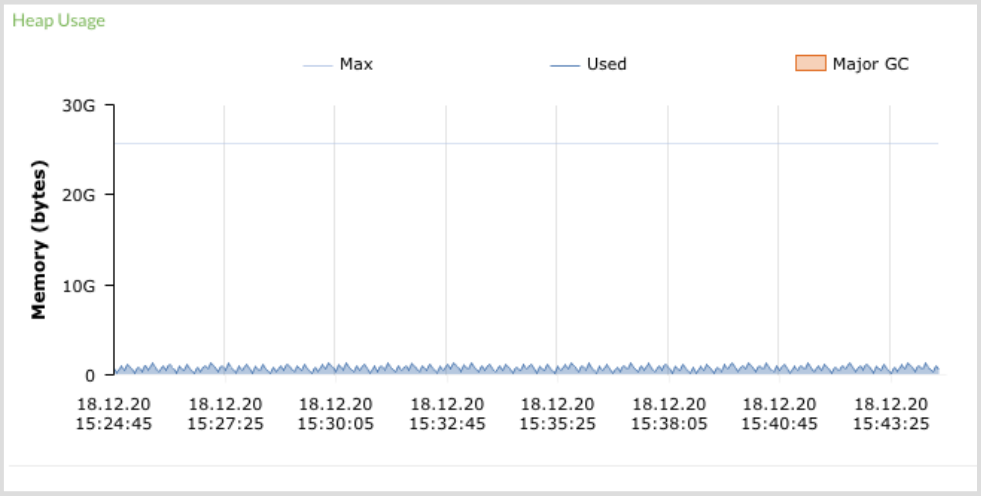




NAVER

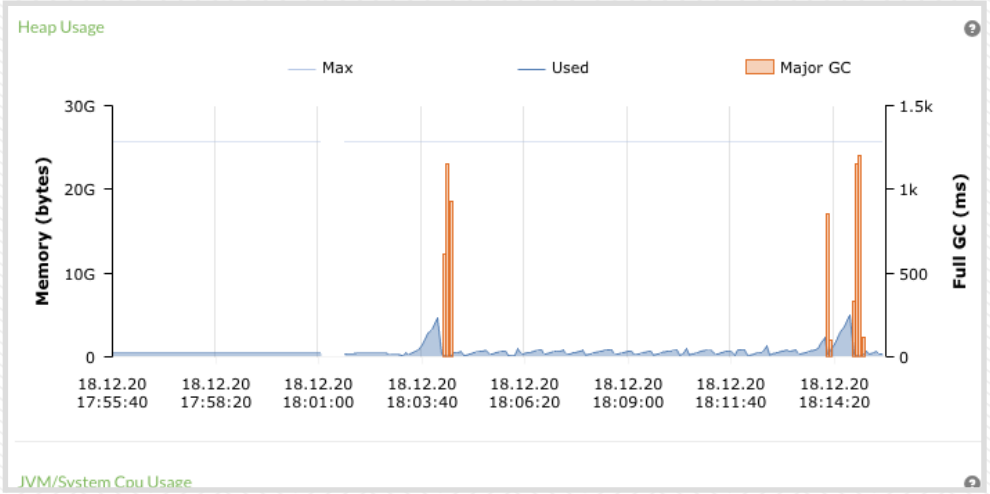
아픈 곳 고치기

- 결과는?
  - ☑ 메모리 사용량 대폭 감소!!
  - ☑ HeapMemory 24G -> 8G까지 조절



아픈 곳 고치기

- 어? 그런데 뭔가 이상함...
  - n-gringer를 이용한 부하 테스트 1시간 진행
  - FullGC가 나기 시작 하다가... 사망...
  - 그런데 특별한 오류 로그가 없음



아픈 곳 고치기

- 눈물을 흘리며 다시 JVM Memory 조정
  - HeapMemory 8G -> 24G
  - n-gringer를 이용한 부하 테스트 1시간 진행 -> 문제없음
  - HeapMemory 24G -> 16G
  - n-gringer를 이용한 부하 테스트 1시간 진행 -> 문제없음
  - n-gringer를 이용한 부하 테스트 2시간 진행 -> 사망

아픈 곳 고치기

- 단서를 찾기 시작함
  - ☑ Profiling, ThreadDump, HeapDump -> 별 이슈 없음...

아픈 곳 고치기

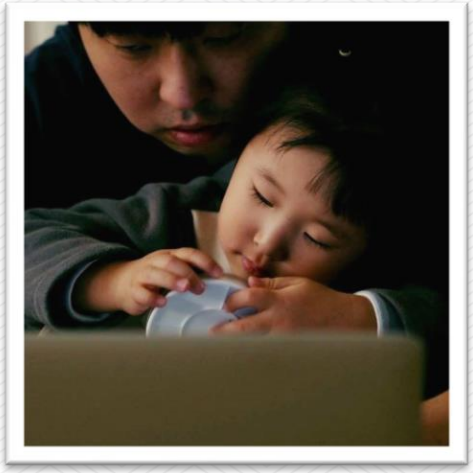
- 그래서 Linux system 로그를 확인함
  - ☑ OOM Killer 로그 확인!

아픈 곳 고치기

- JVM 때문에 사망한 것이 아니라고 판단함
  - 범인은 OpenCV 일 것으로 예상 (Native Memory Release)
  - OpenCV를 사용하기 위해 사용한 JavaCPP 라이브러리 에서 문제가 발생했다고 생각했음

아픈 곳 고치기

- 이번에는 이명현 님과 의논함 (Naver PaaS)



NAVER

아픈 곳 고치기

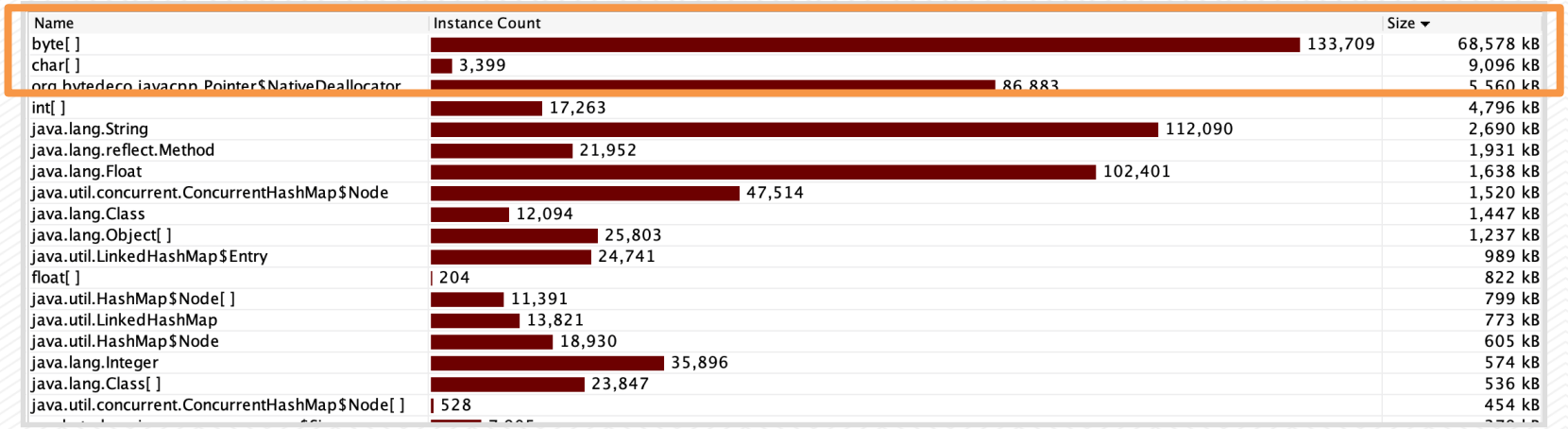
- 이번에는 이명현 님과 의논함 (Naver PaaS)
  - 당시 사용한 OpenJDK Version -> 11.0.1
  - OpenJDK Version -> 10.0.2
  - 3시간 이상 TEST 문제없음

아픈 곳 고치기

- 기쁨의 눈물을 흘리며 다시 JVM Memory 조정
  - HeapMemory 16G -> 8G
  - n-gringer를 이용한 부하 테스트 3시간 이상 진행 -> 문제없음

아픈 곳 고치기

- 다시 Profiling으로 해봄...
  - char[] 232MB -> 9MB
  - byte[] 199MB -> 68MB



Name	Instance Count	Size
byte[ ]	133,709	68,578 kB
char[ ]	3,399	9,096 kB
org.bytedeco.javacpp.Pointer$NativeDeallocator	86,883	5,560 kB
int[ ]	17,263	4,796 kB
java.lang.String	112,090	2,690 kB
java.lang.reflect.Method	21,952	1,931 kB
java.lang.Float	102,401	1,638 kB
java.util.concurrent.ConcurrentHashMap$Node	47,514	1,520 kB
java.lang.Class	12,094	1,447 kB
java.lang.Object[ ]	25,803	1,237 kB
java.util.LinkedHashMap$Entry	24,741	989 kB
float[ ]	204	822 kB
java.util.HashMap$Node[ ]	11,391	799 kB
java.util.LinkedHashMap	13,821	773 kB
java.util.HashMap$Node	18,930	605 kB
java.lang.Integer	35,896	574 kB
java.lang.Class[ ]	23,847	536 kB
java.util.concurrent.ConcurrentHashMap$Node[ ]	528	454 kB

아픈 곳 고치기

- OpenJDK 11.0.2가 나옴... (바로테스트함)

아픈 곳 고치기

- OpenJDK 11.0.2가 나옴...
  - OpenJDK 11.0.1과 동일하게 사망...

아픈 곳 고치기

- JDK 12에서 이 이슈인 것으로 예상되는 내용이 패치가 됨
  - ☑ 물론 (아직) 테스트는 해보지 못했음...
  - ☑ 조만간 해볼 예정...

infrastructure/build
→ **Linux Native Code Checks**
Additional safeguards to protect against buffer overruns in native code have been enabled on Linux. If a buffer overrun is encountered, the system will write the message "stack smashing detected" and the program will exit. Issues of this type should be reported to your vendor.

JDK-8199552 (not public)

https://www.oracle.com/technetwork/java/javase/12-relnote-issues-5211422.html

아픈 곳 고치기

- Heap Memory 32G -> 8G
- Oracle Java8 -> OpenJDK 10
- Boot Instance 1개 -> 2개

조금 더 다듬기

- OCR 엔진 바꾸기

조금 더 다듬기

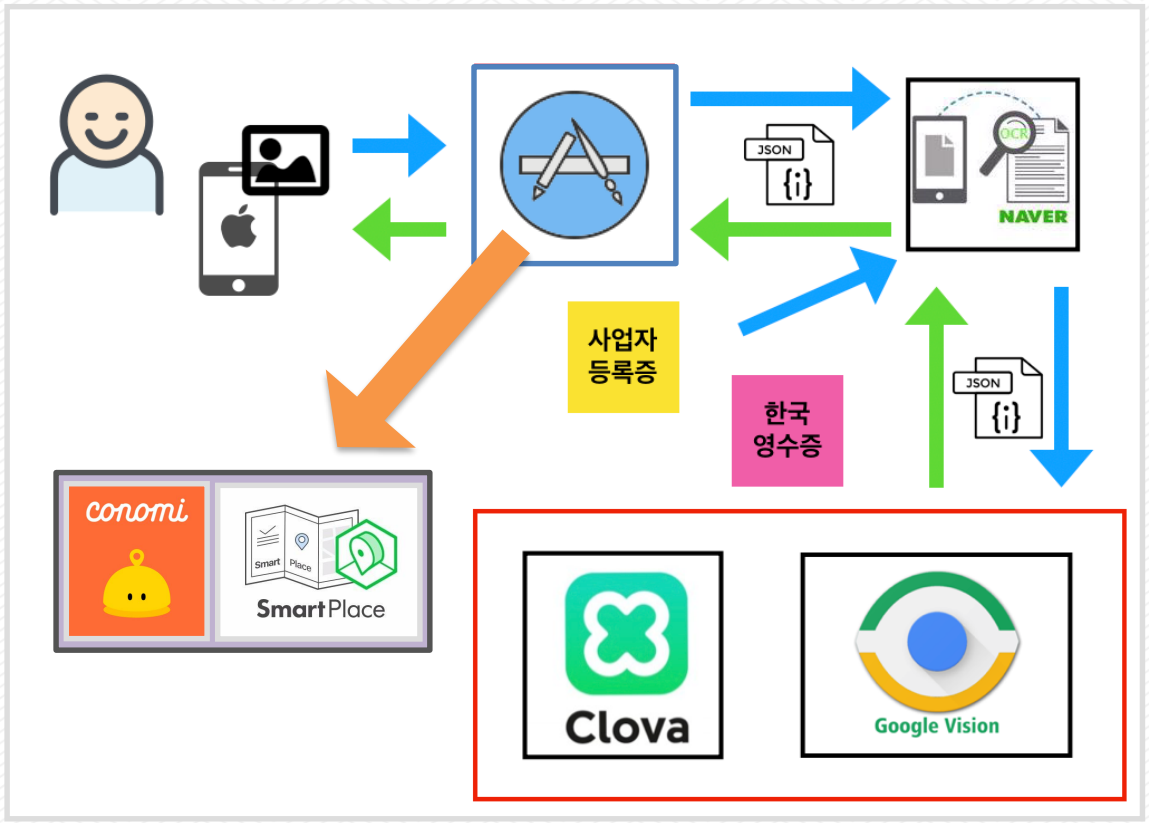
- Google Vision 개발되어 있는 프로젝트 (호출할 때마다 돈이..)
- Clova OCR 엔진이 개발될 예정 (사내 프로젝트)
  - ☑ 처음 이야기했던 일정보다 엄청 빨리 나올 상황...

조금 더 다듬기

- Clova가 나왔을 때 바로 갈아껴야 하는 상황
  - API 요청 interface를 만들고
  - API 호출한 결과 셋을 표준 결과 셋으로 만듬

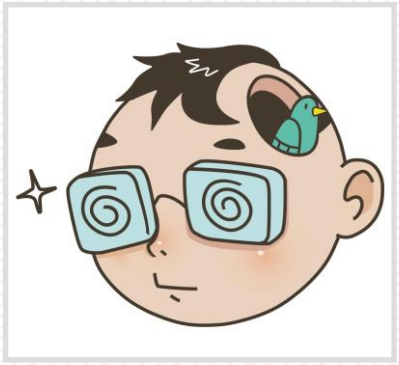
조금 더 다듬기

- 준비를 끝내고 마침내 Clova API 가 나왔습니다.
  - ☑ 코드 작성, 테스트 검증, 배포까지 딱 30분 걸림



조금 더 다듬기

- Google Vision에 비해 넘사벽으로 좋은 인식 성능
  - 오인식 보정을 위한 정규식의 10% 정도가 필요 없어짐



Clova

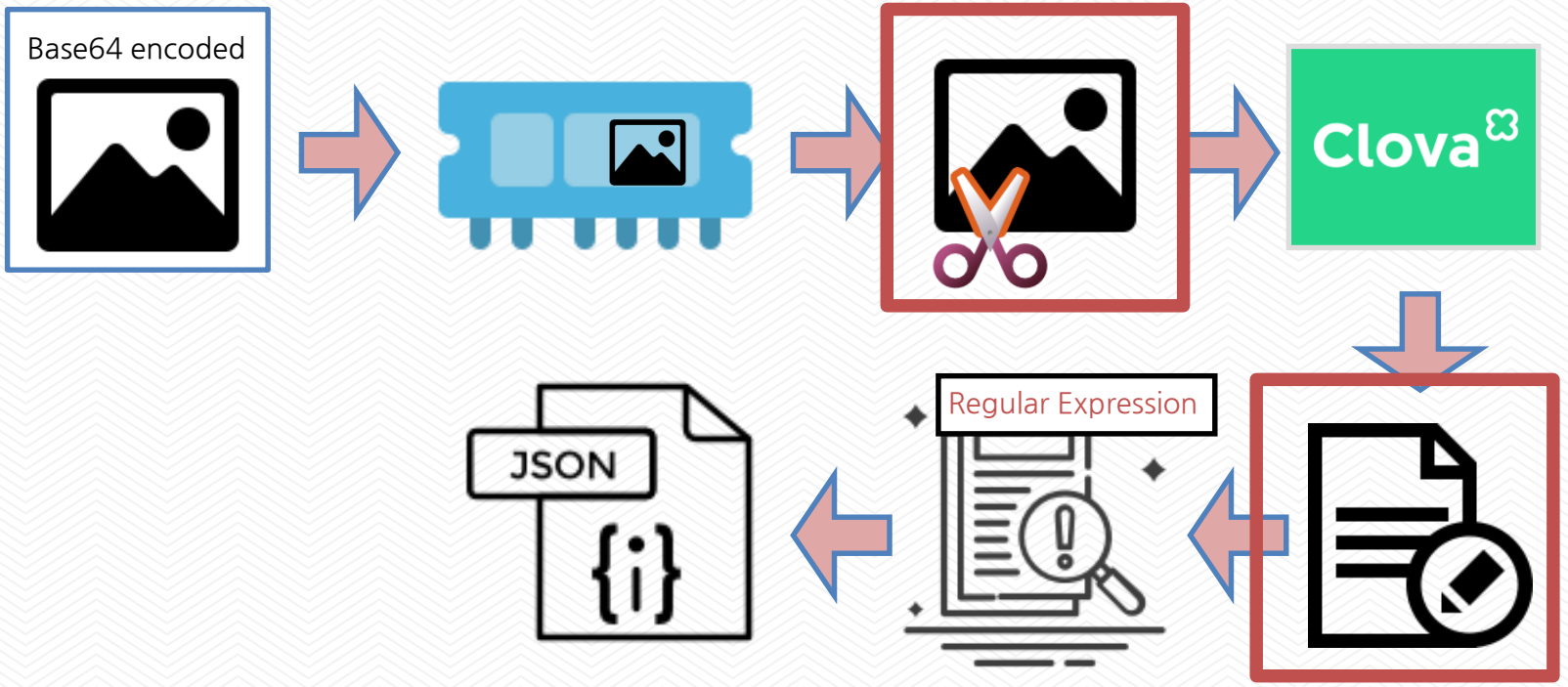
김진중,안종필님 감사합니다.

조금 더 다듬기

- I/O 줄이기

조금 더 다듬기

- I/O가 많이 발생하는 부분
  - 편집이 된 이미지 저장 -> DISK에 저장
  - OCR 엔진 결과 저장 -> DISK에 저장



조금 더 다듬기

- 서버에 RAM이 넘쳐흐르는 상황
  - 당연히 RAM Drive 사용
  - Linux /dev/shm 파티션 사용 (소스코드 수정 최소화)
  - /dev/shm에 저장된 image, text 파일을 주기적으로 network disk에 이동하는 스크립트 작성

조금 더 다듬기

- 결과는 약 20TPS 이상 향상!
  - ☑ 코드 수정 거의 없이...

	stg1	dev.api.bizo...	이경일	ChunC...	2019-02-17 13:55	00:10:00	91.7
	stg1	dev.api.bizo...	이경일	ChunC...	2019-02-17 13:35	00:10:00	68.7

조금 더 다듬기

- Spring Cloud Config 적용

조금 더 다듬기

- OCR의 운영 업무 중 정규식 작업이 큰 비중을 차지함
  - ☑ 정규식 하나 수정하고 배포를 하기에는 비 효율적
  - ☑ 정규식은 미리 YML로 분리된 상태

조금 더 다듬기

- Spring Cloud Config 서버 적용!
  - ☑ 갱신은 기존에 있는 Jenkins를 사용한 curl 호출!
  - ☑ 배포 없이 빠른 정규식 적용 가능
  - ☑ 다른 사람 일 시켜 먹기도 좋다!
  - ☐ 꼭 필요한 것만 사용하는 것도 방법

조금 더 다듬기

- Spring Cloud Config 운영할 때 이슈 발생!
  - 갑자기 YML 설정 파일을 찾을 수 없다고...

조금 더 다듬기

- Spring Cloud Config는 YML, Properties를 File로 저장
  - Linux의 /tmp에 저장하는데...
  - Linux는 특정 기간 access가 없으면 지워버림... (덜덜덜)
    당연히 파일이 없으면 새로 받아오긴 하는데...
    저장소가 느리다면? 저장소가 장애라면? (기본 timeout 5초)

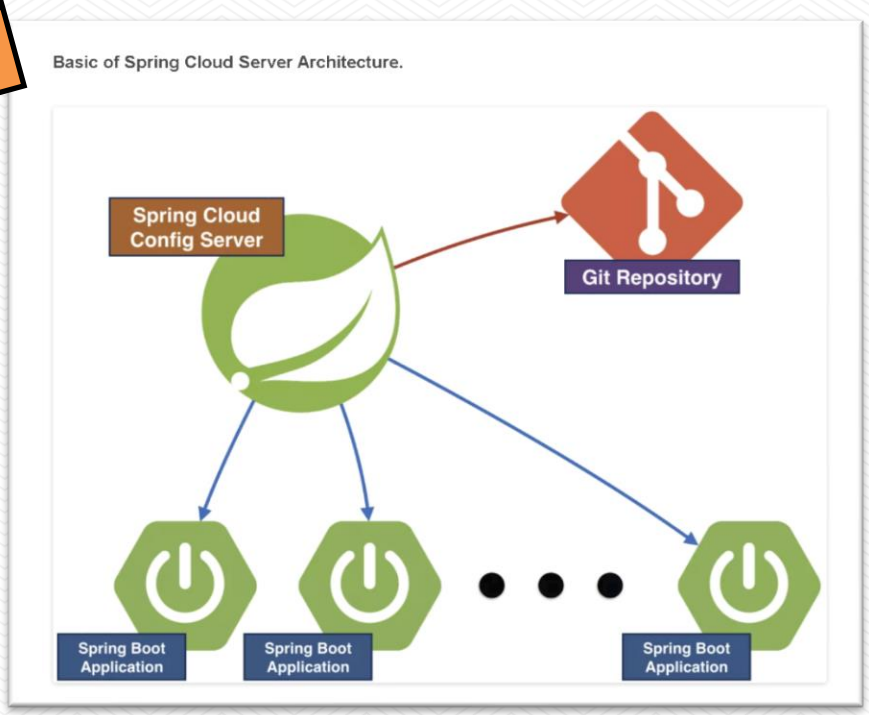
조금 더 다듬기

- Spring Cloud Config 설정을 바꾸는 것이 좋음

```
spring:
  application:
    name: bizocr-config
  cloud:
    config:
      server:
        git:
          basedir: [redacted]/config_tmp/
```

조금 더 다듬기

- Spring Cloud Config 관련 궁금하신 분들은
  - http://bit.ly/2vlbhRu <- 여기를 참고하세요.



결과를 정리해보면...

- 복잡한 정규식 코드 -> YML로 분리, 정규식 수련
- Spring3.X, Common lib -> SpringBoot, 프로젝트 독립
- Application Start Time 1분 이상 -> 3~5초
- Memory 사용량 32G -> 8G
- CPU, Memory 사용량 절약 -> 인스턴스 1개 더 추가
- 시간 관계상 이야기하지 못한 경험들...
- 도메인 지식 습득, 내 서비스가 됨

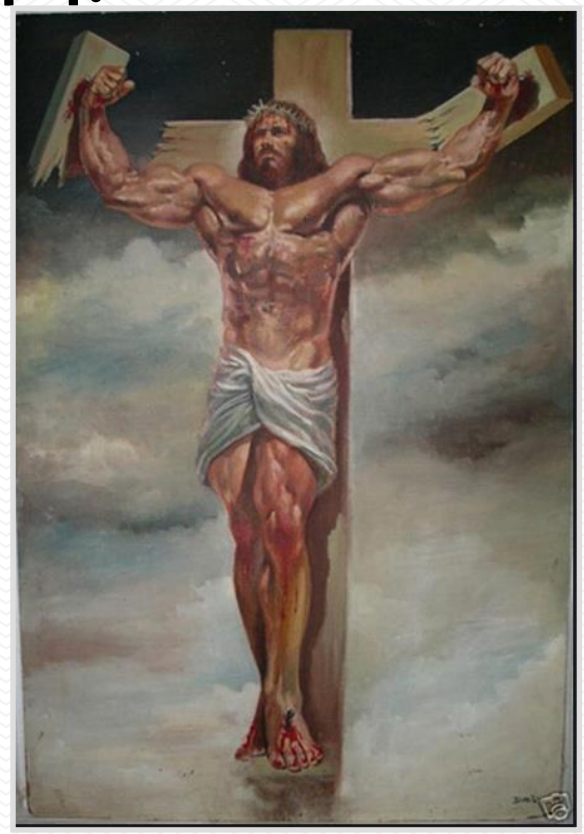
정리하며...

- 더 하고 싶은 이야기가 많지만 시간 관계상 생략…
  - ☑ Apache, Tomcat -> Nginx + Spring Boot
  - ☑ Nginx Traffic Throttling
  - ☑ Graceful Shutdown
  - ☑ ELK stack 적용
  - ☑ GC튜닝
  - ☑ 기타등등...
  - ☑ 또 기타등등...

정리하며...

- ~~타이쿤있다...~~ IT에서는 잘못된 말이다...
- 아직 저의 개선은 끝나지 않았습니다.

  저랑 같이하실분?



정리하며...

- 계속적인 관심과 업그레이드, 리팩토링이 필요함

정리하며...

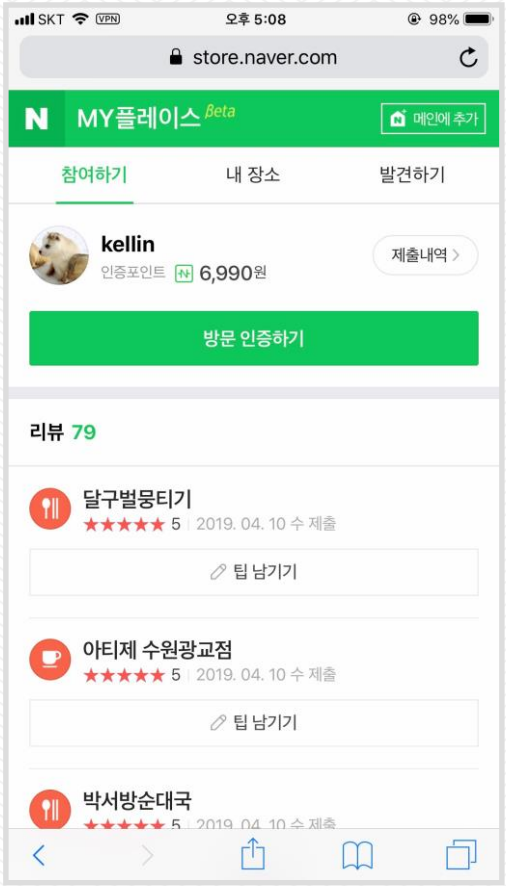
- 개밥 먹기를 하세요.

정리하며...

- 일본에서 직접 테스트해본 영수증의 흔적들...

정리하며...

- 한국에서도 열심히 구석구석 다니며 테스트



정리하며...

- 개밥 먹기를 하세요.
- TestCase는 TestCase일 뿐 필드에서 일어나는 일을 전부 커버할 순 없다.

정리하며...

- 피할 수 없다면 즐기자
- 코드에서 도메인 지식을 습득해 내 것으로 만들자

Naver 쇼핑데이터플랫폼, 플레이스는 개발자를 모집하고 있습니다. (특히 BizOCR같이 하실분?!!)

kellin.me@navercorp.com 으로 문의주셔도 됩니다.

NAVER

Thank You.

NAVER