**주문 앱 영수증에서 데이터 추출 프로그램 개발**

설계 및 운영 매뉴얼

목차

[1. 추출서버 설정 3](#_Toc120746493)

[1.1 포트 설정 3](#_Toc120746494)

[1.2 수집서버 url 설정 3](#_Toc120746495)

[2. 추출서버 프로토콜 4](#_Toc120746496)

[2.1 수집 서버 연동 API 4](#_Toc120746497)

[2.2 테스트 프로트콜 4](#_Toc120746498)

[3. 설계 – 추출 로직 5](#_Toc120746499)

[3.1 주문 App 종류 구분 5](#_Toc120746500)

[3.2 개행문자로 문자열 자르기 5](#_Toc120746501)

[3.3 루프 및 디코딩 및 특수문자 제거 6](#_Toc120746502)

[3.4 단일 행 파싱 7](#_Toc120746503)

[3.5 다중 행 파싱 7](#_Toc120746504)

[3.6 메뉴 파싱 8](#_Toc120746505)

[3.6.1 배민 메뉴 파싱 8](#_Toc120746506)

[3.6.2 요기요 메뉴 파싱 8](#_Toc120746507)

[3.6.3 땡겨요 메뉴 파싱 9](#_Toc120746508)

# 추출서버 설정

## 포트 설정

* application.yml

application.yml에 server.port 부분을 수정한다.

|  |
| --- |
| server:  port: 9000 |

## 수집서버 url 설정

* Application.yml

application.yml에 agent.server.url 부분을 수정한다.

|  |
| --- |
| agent:  server:  url: https://210.116.92.72:9280 |

# 추출서버 프로토콜

## 수집 서버 연동 API

POS 단말에서 HexaDecimal 인 RawData를 받아서 파싱 후에 수집서버에 추출 결과를 전달한다.

|  |
| --- |
| /api/neworder |

## 테스트 프로트콜

POS 단말에서 HexaDecimal 인 RawData를 받아서 파싱 후에 파싱 결과를 수집서버에 전달하지 않고 파싱 결과를 그래도 리턴해준다.

테스트 용도로 사용한다.

|  |
| --- |
| /api/order |

# 설계 – 추출 로직

## 주문 App 종류 구분

주문 App마다 파싱 알고리즘이 다르므로 POS 단말에서 HexaDecimal 데이터를 받으면 맨 먼저 어떤 주문 App인지 구분해야 한다.

|  |  |
| --- | --- |
| 요기요 | 전체 HexaDecimal에서 요기요라는 단어가 들어가면 요기요로 판단한다.  만일 [요기요 익스프레스]라는 단어가 들어가면 YE로 가정한다.  [요기요 포장] 이 들어가면 YG에 주문종류는 P로 판단.  위 두가지 경우가 아니면 YG에 주문종류 D로 판단. |
| 배민라이더스 | [배민라이더스] 가 포함되면 BR로 판단.  [배달 주문전표] 가 들어가 있으면 배달  [포장 주문전표] 가 들어가 있으면 포장이다. |
| 배민원 | [배민1 주문전표]가 들어가 있으면 신배민이다.  [배민1주문] 이 들어가 있으면 구배민이다. |
| 신배민 | [주문번호:B1] 이 포함되어있고,  [배달 주문전표] 가 들어가 있으면 신배민/배달이고  [포장 주문전표] 가 들어가 있으면 신배민/포장이다. |
| 구배민 | [주문번호:B1] 이 포함되어있고,  [사전결제 여부:] 가 들어가 있다.  [포장 접수번호] 가 추가적으로 들어가 있으면 구배민/포장이고, 없으면 구배민/배달이다. |
| 땡겨요 | [땡겨요]가 들어간 경우로 판단. |

## 개행문자로 문자열 자르기

HexaDecimal 전체로 주문 App 종류를 구분한 다음 개행문자로 잘라서 ArrayList로 만든다.

주문 앱마다 개행문자가 다르므로 종류별로 구분하여 자른다.

|  |  |  |
| --- | --- | --- |
|  | 개행문자 | 비고 |
| 요기요 | 0D0A |  |
| 배민 | 0A0D |  |
| 땡겨요 | 0A | 0A 로 개행시 한 문자가 4비트이므로 0A를 찾아서 개행을 하게 되면 홀수번째 인덱스가 변환될 경우의 수가 있다.  예를 들어 B0A1 이렇게 되면, 두 문자가 묶여서 한 문자를 이루므로 파싱 오류가 발생한다  따라서, 짝수번째 인덱스에서만 0A 인 경우를 찾아서 잘라야 한다. |

## 루프 및 디코딩 및 특수문자 제거

개행문자로 잘라서 저장된 ArrayList에는 인코딩되어있는 Hexa 문자열이 들어가 있다.

이것을 루프를 돌리면서 디코딩한다. 즉, 한 줄씩 파싱한다. Hexa 문자가 혹시 필요할 수 있으므로 Hexa문자로 자른다음에 Hexa문자를 디코딩하여 인코딩 문자와 디코딩 문자를 같이 사용하도록 한다.

디코딩시에는 특수문자를 제거한다. 특수문자는 POS에서 정한 영수증 규칙입지만 파싱하는 입장에서는 무슨 규칙이 있는지는 알 수 없으면 주문 App마다 종류가 모두 다르므로 하나씩 눈으로 보고 걸러내어야 하는 어려움이 있다.

세문자 혹은 두 문자의 규칙을 찾아서 제거한다. 한 문자인 경우는 홀수번째 인덱스에서 발생할 수 있으므로 짝수번째 인지 확인하고 제거해야 한다.

|  |  |
| --- | --- |
| 배민 | hexadecimal = hexadecimal.replace("1B21001D2400001D7630302C0004000A0D", "");  hexadecimal = hexadecimal.replace("1B401D427920", ""); // 구배민  hexadecimal = hexadecimal.replace("1B21001B2118", ""); // 신배민  hexadecimal = hexadecimal.replace("1B401B2118", ""); // 신배민  hexadecimal = hexadecimal.replace("1B2100", ""); // 신배민: 일반텍스트  hexadecimal = hexadecimal.replace("1B2118", ""); // 신배민: 인쇄 모드 설정  hexadecimal = hexadecimal.replace("1B4500", ""); // 구배민  hexadecimal = hexadecimal.replace("1B4501", ""); // 구배민  hexadecimal = hexadecimal.replace("1B40", ""); // 구배민: 프린트 초기화  hexadecimal = hexadecimal.replace("1B61", ""); // 구배민: 프린트 초기화  hexadecimal = hexadecimal.replace("201D42", ""); // 구배민: 사전결부 여부 오른쪽 특수문자 |
| 요기요 | hexadecimal = hexadecimal.replace("1B4501", "");  hexadecimal = hexadecimal.replace("1B45", "");  hexadecimal = hexadecimal.replace("1B32", "");  hexadecimal = hexadecimal.replace("1B61", "");  hexadecimal = hexadecimal.replace("1B21", "");  hexadecimal = hexadecimal.replace("1B4D", "");  hexadecimal = hexadecimal.replace("1D42", "");  hexadecimal = hexadecimal.replace("1D2101", "");  hexadecimal = hexadecimal.replace("1D21", "");  요기요의 경우 글자마다 사이 사이 00이 들어가 있다.  00을 모두 제거해야 하는데 한 문자이므로 00을 찾으면 안되고 짝수번째 인덱스 인경우만 모두 00을 제거한다. |
| 땡겨요 | hexadecimal = hexadecimal.replace("1B6130", "");  hexadecimal = hexadecimal.replace("1B6131", "");  hexadecimal = hexadecimal.replace("1B6132", "");  hexadecimal = hexadecimal.replace("1B40", "");  hexadecimal = hexadecimal.replace("1B2108", "");  hexadecimal = hexadecimal.replace("1B2100", "");  hexadecimal = hexadecimal.replace("1B21", "");  hexadecimal = hexadecimal.replace("1D2100", "");  hexadecimal = hexadecimal.replace("1D2101", "");  hexadecimal = hexadecimal.replace("1D21", "");  hexadecimal = hexadecimal.replace("011B", "");  hexadecimal = hexadecimal.replace("6131", ""); |

## 단일 행 파싱

라벨이 포함된 단일행의 경우는 루프를 돌면서 바로 파싱이 가능한 가장 쉬운 경우이다.

예를 들어, [주문 일자: xxxx] 이렇게 라벨이 포함된 경우 해당 라벨을 제거하고 바로 파싱이 가능하다.

그러나, [주문 일자] 와 같은 라벨이 없는 경우는 아래 위로 고정 라벨이 있는지를 파악해야 한다. 고정 라벨이 있으면 해당 고정 라벨의 행라인 번호를 기억하고 있다가 루프가 끝난 다음 해당 행라인에서 몇번째인지를 파악해서 파싱한다. 이때 중간에 다중 행이 있는 경우 파싱이 안될수 있다.

## 다중 행 파싱

주소, 원산지와 한 줄에 표시가 안되고 1줄이상 표시가 되는 경우이다.

이 때는 시작되는 행라인과 끝나는 행라인을 기억하고 그 사이에 있는 부분을 모두 가져가서 파싱해야 한다.

## 메뉴 파싱

### 신배민 메뉴 파싱

메뉴인 경우와 옵션의 경우 모두 한 줄을 파싱할 때 스페이스 두 칸으로 자른다.

* 옵션 파싱

옵션은 반드시 [ +] 로 스페이스와 플러스로 시작한다.

만일, 옵션이 개행될 경우는 [ ] 로 스페이스 두 칸으로 시작한다.

끝에 원으로 끝나는 경우는 맨 끝을 스페이스 자른 다음에 원을 제거하고 숫자로 변환한다. 만일 NumberFormat Exception이 발생하면 숫자가 아니므로 가격으로 파싱하지 않는다.

맨 끝에 () 안에 가격이 있는 경우가 있으므로 ()안에 있는 것을 꺼내서 원을 제거하고 숫자로 변환하여 에러가 나지 않으면 가격으로 파싱한다.

* 메뉴 파싱

메뉴는 메뉴명과 수량, 가격이 모두 표시된다. 그래서 [ ] 스페이스 두 칸으로 잘라서 비어있는 것들을 제거해서 개수가 3개가 되면 메뉴이다.

수량에 표시되는 숫자는 수량 바로 왼쪽에 정렬이 된다. 따라서 메뉴명이 길 경우 메뉴명과 수량사이에 스페이스가 한 칸이 되는 경우도 있다. 이 경우는 스페이스 두 칸으로 파싱하여 2개 인 경우이므로 파싱 개수가 2개이고 맨 앞이 스페이스 한 개 이하일 경우로 판단한다.

메뉴가 개행되는 경우는 메뉴명이 길 경우이다. 이 경우에 수량과 가격이 다음줄로 밀린다. 따라서 파싱갯수가 1일이면 긴 메뉴명이 들어간 경우이다. 그리고 앞쪽이 스페이스 5개 이상으로 시작하고 파싱갯수가 2개이면 수량과 가격이 들어간 경우로 판단한다.

### 구배민 메뉴 파싱

메뉴인 경우와 옵션의 경우 모두 한 줄을 파싱할 때 스페이스 두 칸으로 자른다.

* 옵션 파싱

옵션은 반드시 [ +] 로 스페이스와 플러스로 시작한다.

만일, 옵션이 개행될 경우는 [ ] 로 스페이스 두 칸으로 시작한다.

끝에 원으로 끝나는 경우는 맨 끝을 스페이스 자른 다음에 원을 제거하고 숫자로 변환한다. 만일 NumberFormat Exception이 발생하면 숫자가 아니므로 가격으로 파싱하지 않는다.

맨 끝에 () 안에 가격이 있는 경우가 있으므로 ()안에 있는 것을 꺼내서 원을 제거하고 숫자로 변환하여 에러가 나지 않으면 가격으로 파싱한다.

* 메뉴 파싱

메뉴는 메뉴명과 수량, 가격이 모두 표시된다. 그래서 [ ] 스페이스 두 칸으로 잘라서 비어있는 것들을 제거해서 개수가 3개가 되면 메뉴이다.

메뉴가 개행되는 경우는 메뉴명이 개행되지 수량과 가격은 개행되지 않는다. 그러므로 현재라인에서는 완전한 메뉴인지 메뉴명이 개행된건지 판단할 수 없다.

따라서 정상적을 처리를 한후에 파싱갯수가 1개가 올 경우 기존에 메뉴를 꺼내서 메뉴명을 뒤에 덧붙인다.

### 요기요 메뉴 파싱

스페이스 두 칸으로 자르는 로직은 동일하다.

* 옵션 파싱

[- ] 마이너스와 스페이스로 시작하면 옵션이다.

만일 옵션이 개행되면 옵션개행인지 메뉴 개행인지 알 수가 없다.

* 메뉴 파싱

개수가 3개이면 메뉴명, 수량, 가격이 모두 존재하는 메뉴 파싱로직을 수행한다.

배민의 경우처럼 수량과 가격이 개행되는 경우는 존재하지 않는다. 전부 메뉴명이 개행한다. 그러므로 파싱 개수가 2개인 경우는 존재하지 않는다.

파싱갯수가 1개인 경우는 옵션 개행과 메뉴 개행을 구분할 수 없다.

따라서 이전 라인이 옵션 파싱이였는지, 메뉴 파싱이였는지 기억해야 한다.

만일 이전 라인이 옵션 파싱여였다면 옵션 개행이며 메뉴 파싱이였다면 메뉴 개행이다.

### 땡겨요 메뉴 파싱

스페이스 두 칸으로 자르는 로직은 동일하다.

땡겨요는 개행문자가 0A 라서 그런지 개행문자로 개행하는 경우가 없다. 문자가 계속 연달아서 쓰면 영수증에서 줄이 밀려서 두 라인이 된다.

따라서 한 줄씩 파싱하는 이 로직에서는 개행이 발생하지 않는다.

* 옵션 파싱

[ + ] 로 시작하면 옵션 파싱이다.

옵션에 가격이 있는 경우가 있으므로 맨 마지막에 괄호로 끝나는 경우 가격을 파싱해 낸다.

* 메뉴 파싱

그 외 경우는 메뉴 파싱이다.

* 할인 금액

할인 금액은 파싱 개수가 2개 이므로 2개인 경우 파싱에서 제외한다.