정규 표현식

정규표현식 (Regular expressions)

- 텍스트 검색, 치환에 사용
- 수십 라인의 프로그래밍 없이 정규식 1~2줄로 대부분의 문자열 작업 가능
- 선배들의 노하우가 담긴 파워풀한 텍스트 관련 도구
- 익숙해진다면 손 빠른 개발에 큰 도움을 줄 수 있음
- 온전한 프로그래밍 언어는 아님
- 다른 프로그래밍 언어나 제품에 포함된 '작은 언어'의 느낌
- 제품마다 조금씩 문법이 다름

⇒ ex

→ email, 주민 번호, 생년월일 등의 형식 검증

→ 텍스트를 취급하는 개발 코드 작성

→ (텍스트) 데이터의 전 처리 작업

→ 프로젝트 리팩토링 작업

→ Database 검색, 치환 작업

- SW 엔지니어에게 '외국어' 같은 존개
- 다양한 문제 해결법
- 친숙해진다면 파워풀한 도구

정규 표현식 온라인 테스트 도구

- http://regexr.com

문자 하나 찾기

- 일반적인 문자 그대로 기재
- `.` 과 같은 메타 문자를 검색하려 `\` 로 이스케이프

문자 집합으로 찾기

- 대괄호 `[]`를 사용하여 문자 집합 표현
- []집합에 속한 문자 가운데 하나가 일치
- []내에 `-` 은 연속 요소를 표현 ⇒ ex) [1-5] → [12345]
- 캐럿 `^` 문자는 잡합 안에 있는 문자나 범위를 모두 제외

반복 찾기

- 파워풀한 정규 표현 패턴의 능력
- `+` : 하나 이상 일치
- `*` : 없거나 하나 이상 일치
- `?` : 없거나 하나 일치
- 중괄호 `{}` 내에 반복 횟수 기재 ⇒ ex) {3} → 3번
- 게으른 수량자로 문자를 최소로 일치

위치 찾기

- 텍스트 영역 내 특정 위치에서 검색 희망
- `\b` : 단어 경계
- `^` : 문자열 경계의 시작
- `$` : 문자열 경계의 끝

하위 표현식

- 큰 표현식 안에 속한 일부 표현식을 한 항목으로 다루도록 묶음
- `()` : 괄호로 묶음 기능 ⇒ ex) >{2,} → (>){2,}
- 파워풀한 중첩된 하위 표현식

역참조

- 하위 표현식으로 매칭된 타겟을 참조
- 일치한 부분을 반복해 찾거나 치환에 사용
- 텍스트를 검색하고 치환하는데 매우 유용
- 실수로 중복된 전치사 수정

전방 탐색

- 일치 영역을 발견해도 그 값을 반환하지 않는 패턴
- 실제로는 하위 표현식이며 같은 형식으로 작성
- (?=일치할 텍스트)

후방 탐색

- 전방탐색과 동일한 개념으로 방향만 역방향
- (?<=일치할 텍스트)

```
/** 정규표현식 예시 ex) 자바 */
import java.util.regex.Pattern;

/** 문자열의 형식을 검사하는 기능을 갖는 클래스 */
public class PatternChecker {

    /** 숫자 모양에 대한 형식 검사 */
    public static boolean isNum(String str) {
        return Pattern.matches("^[0-9]*$", str);
    }

    /** 영문으로만 구성되었는지에 대한 형식 검사 */
    public static boolean isEng(String str) {
        return Pattern.matches("^[a-zA-Z]*$", str);
    }

    /** 한글로만 구성되었는지에 대한 형식 검사 */
    public static boolean isKor(String str) {
        return Pattern.matches("^[ㄱ-ㅎ가-힣]*$", str);
    }

    /** 영문과 숫자로만 구성되었는지에 대한 형식 검사 */
    public static boolean isEngNum(String str) {
        return Pattern.matches("^[a-zA-Z0-9]*$", str);
    }

    /** 한글과 숫자로만 구성되었는지에 대한 형식 검사 */
    public static boolean isKorNum(String str) {
        return Pattern.matches("^[ㄱ-ㅎ가-힣0-9]*$", str);
    }

    /** 이메일 형식인지에 대한 검사 --> "아이디@도메인"의 형식을 충족해야 한다. */
    public static boolean isEmail(String str) {
        return Pattern.matches("[0-9a-zA-Z]+(.[_a-z0-9-]+)*@(?:\\w+\\.)+\\w+$", str);
    }

    /** 핸드폰번호인지에 대한 형식검사.
        반드시 앞자리가 010,010,016~9사이를 충족해야 하며,
        각 부분에 대한 자리수도 충족시켜야 한다.
        "-"는 허용하지 않는다. */
    public static boolean isCellPhone(String str) {
        return Pattern.matches("^01(?:0|1|[6-9])(?:\\d{3}|\\d{4})\\d{4}$", str);
    }

    /** 전화번호인지에 대한 형식검사. 각 연결부는 "-"로 구분되어야 한다.
        각 부분에 대한 자리수도 충족시켜야 한다.
```

```
        "-"는 허용하지 않는다. */
    public static boolean isTel(String str) {
        return Pattern.matches("^\\d{2,3}-\\d{3,4}-\\d{4}$", str);
    }

    /** 주민번호에 대한 글자수 및 뒷자리 첫글자가 1~4의 범위에 있는지에 대한 검사.
        "-"는 허용하지 않는다. */
    public static boolean isJumin(String str) {
        return Pattern.matches("^\\d{6}[1-4]\\d{6}", str);
    }

    /** 아이피주소 형식에 대한 검사 */
    public static boolean isIP(String str) {
        return Pattern.matches("([0-9]{1,3})\\.([0-9]{1,3})\\.([0-9]{1,3})\\.([0-9]{1,3})", str);
    }
}
```