모듈 java.base
패키지 java.util.regex

클래스 패턴

- java.lang.Object
- - java.util.regex.Pattern
- 구현 된 모든 인터페이스 :
  Serializable

```
공개 최종 클래스 패턴
확장 개체 직렬화
구현
```

정규식의 컴파일 된 표현입니다.

문자열로 지정된 정규식은 먼저이 클래스의 인스턴스로 컴파일되어야합니다. 그런 다음 결과 패턴을 사용 하여 정규식에 대해 Matcher임의의 문자 시퀀스 를 일치시킬 수 있는 개체 를 만들 수 있습니다 . 일치를 수행하는 데 관련된 모든 상태가 일치 자에 있으므로 많은 일치자가 동일한 패턴을 공유 할 수 있습니다.

따라서 일반적인 호출 순서는 다음과 같습니다.

```
패턴 p = 패턴. compile( "a * b");
 매처 m = p. matcher( "aaaaab");
 부울 b = m. matches();
```

matches방법은 정규 표현식은 한 번만 사용될 때의 편의이 클래스에 의해 정의된다. 이 메소드는 표현식을 컴파일하고 단일 호출에서 입력 시퀀스를 일치시킵니다. 진술

```
부울 b = Pattern.matches ( "a * b", "aaaaab");
```

위의 세 문과 동일하지만 반복 된 일치의 경우 컴파일 된 패턴을 재사용 할 수 없기 때문에 효율성이 떨어집니다.

이 클래스의 인스턴스는 변경할 수 없으며 여러 동시 스레드에서 사용하기에 안전합니다. Matcher클래스의 인스턴스는 이러한 사용에 안전하지 않습니다.

정규식 구문 요약

구성	성냥
캐릭터	
엑스	문자 x
₩₩	백 슬래시 문자
₩0엔	8 진수 값 0n (0 <= n <= 7)을 가진 문자
₩0nn	진수 값을 가지는 문자 0NN (0 <= N <= 7)

구성	성냥
₩0mnn	진수 값을 가지는 문자 0MNN (0 <= m <= 3 0 <= N <= 7)
₩xhh	16 진수 값이 hh 인 문자 0x
₩u으	16 진수 값이있는 문자 0xhhhh
₩x{h … h}	16 진수 값 0xh … h (Character.MIN_CODE_POINT <= 0xh … h <= Character.MAX_CODE_POINT)가 있는 문자
₩N{이름}	유니 코드 문자 이름이 'name'인 문자
₩t	탭 문자 ('₩u0009')
₩n	개행 (줄 바꿈) 문자 ('₩u000A')
₩r	캐리지 리턴 문자 ('₩u000D')
₩f	용지 공급 문자 ('₩u000C')
₩a	경고 (종) 문자 ('₩u0007')
₩e	이스케이프 문자 ('₩u001B')
₩c엑스	x에 해당하는 제어 문자

캐릭터 클래스

[abc]	a,, b또는 c(단순 클래스)
[^abc]	모든 문자 제외 a, b또는 c(부정)
[a-zA-Z]	athrough z 또는 Athrough Z, 포괄적 (범위)
[a-d[m-p]]	a~ d~ ~ m~ p: [a-dm-p](연합)
[a-z&&[def]]	d, e또는 f(교차점)
[a-z&&[^bc]]	a~ z를 제외하고 b및 c:([ad-z]빼기)
[a-z&&[^m-p]]	a를 통해 z, 그리고를 m통해 p: [a-lq-z](빼기)

미리 정의 된 문자 클래스

.	모든 문자 (줄 종결 자 와 일치하거나 일치하지 않을 수 있음)
₩d	숫자 : [0-9]
₩D	숫자가 아닌 : [^0-9]
₩h	수평 공백 문자 : [₩t₩xA0₩u1680₩u180e₩u2000-₩u200a₩u202f₩u205f₩u3000]
₩H	수평이 아닌 공백 문자 : [^₩h]
₩s	공백 문자 : [₩t₩n₩x0B₩f₩r]
₩S	공백이 아닌 문자 : [^₩s]
₩v	세로 공백 문자 : [₩n₩x0B₩f₩r₩x85₩u2028₩u2029]
₩V	수직이 아닌 공백 문자 : [^₩v]
₩w	워드 캐릭터 : [a-zA-Z_0-9]

구성	성냥
₩W	비 단어 문자 : [^₩w]

POSIX 문자 클래스 (US-ASCII 만 해당)

₩p{Lower}	소문자 알파벳 문자 : [a-z]
₩p{Upper}	알파벳 대문자 :[A-Z]
₩p{ASCII}	모든 ASCII :[₩x00-₩x7F]
₩p{Alpha}	알파벳 문자 :[₩p{Lower}₩p{Upper}]
₩p{Digit}	10 진수 : [0-9]
₩p{Alnum}	영숫자 :[₩p{Alpha}₩p{Digit}]
₩p{Punct}	구두점 : 다음 중 하나 !"#$%&'()*+,-./:;<=>?@[₩]^_`{|}~
₩p{Graph}	보이는 캐릭터 : [₩p{Alnum}₩p{Punct}]
₩p{Print}	인쇄 가능한 문자 : [₩p{Graph}₩x20]
₩p{Blank}	공백 또는 탭 : [₩t]
₩p{Cntrl}	제어 문자 : [₩x00-₩x1F₩x7F]
₩p{XDigit}	16 진수 : [0-9a-fA-F]
₩p{Space}	공백 문자 : [₩t₩n₩x0B₩f₩r]

java.lang.Character 클래스 (단순 Java 문자 유형)

₩p{javaLowerCase}	java.lang.Character.isLowerCase ()와 동일
₩p{javaUpperCase}	java.lang.Character.isUpperCase ()와 동일
₩p{javaWhitespace}	java.lang.Character.isWhitespace ()와 동일
₩p{javaMirrored}	java.lang.Character.isMirrored ()와 동일

유니 코드 스크립트, 블록, 범주 및 이진 속성에 대한 클래스

₩p{IsLatin}	라틴 스크립트 문자 (script)
₩p{InGreek}	그리스어 블록의 문자 (block)
₩p{Lu}	대문자 (category)
₩p{IsAlphabetic}	알파벳 문자 (이진 속성)
₩p{Sc}	통화 기호
₩P{InGreek}	그리스어 블록에서 하나를 제외한 모든 문자 (부정)
[₩p{L}&&[^₩p{Lu}]]	대문자 (빼기)를 제외한 모든 문자

경계 매처

^	줄의 시작

구성	성냥
$	줄의 끝
₩b	단어 경계
₩b{g}	유니 코드 확장 자소 클러스터 경계
₩B	비 단어 경계
₩A	입력의 시작
₩G	이전 경기의 끝
₩Z	입력의 끝이지만 최종 종결 자 (있는 경우)
₩z	입력의 끝

줄 바꿈 매처

₩R	모든 유니 코드 줄 바꿈 시퀀스는 다음과 같습니다. ₩u000D₩u000A|[₩u000A₩u000B₩u000C₩u000D₩u0085₩u2028₩u2029]

유니 코드 확장 Grapheme 매처

₩X	모든 유니 코드 확장 자소 클러스터

욕심 많은 수량 자

엑스?	X , 한 번 또는 전혀
엑스*	X , 0 회 이상
엑스+	X , 1 회 이상
X의 {N}	X , 정확히 n 번
X의 {N, }	X , 최소 n 회
X {n ,m}	X , n 회 이상 m 회 이하

꺼리는 수량 자

엑스??	X , 한 번 또는 전혀
엑스*?	X , 0 회 이상
엑스+?	X , 1 회 이상
X의 {N}?	X , 정확히 n 번
X의 {N,}?	X , 최소 n 회
X {n ,m}?	X , n 회 이상 m 회 이하

소유 수량 자

엑스?+	X , 한 번 또는 전혀

구성	성냥
엑스*+	X , 0 회 이상
엑스++	X , 1 회 이상
X의 {N}+	X , 정확히 n 번
X의 {N,}+	X , 최소 n 회
X {n ,m}+	X , n 회 이상 m 회 이하

논리 연산자

XY	X 다음에 Y
X |Y	어느 X 또는 Y
(엑스)	X, 캡처 그룹으로

역 참조

₩엔	n 번째 캡처 그룹이 일치하는 항목이 무엇이든
₩k < 이름 >	명명 된 캡처 그룹 "이름"이 일치하는 항목이 무엇이든

인용

₩	아무것도 없지만 다음 문자를 인용합니다.
₩Q	아무것도 없지만 모든 문자를 인용합니다. ₩E
₩E	아무것도,하지만 시작된 인용 끝 ₩Q

특수 구조 (명명 된 캡처 및 비 캡처)

(?<name>엑스)	X , 명명 된 캡처 그룹
(?:엑스)	X , 비 캡처 그룹으로
(?idmsuxU-idmsuxU)	아무것도하지만, 회전이 플래그를 일치 내가 D m s의 u는 X U를 에 - 오프
(?idmsux-idmsux:엑스)	X , 주어진 플래그가 있는 비 캡처 그룹 i d m s u x on-off
(?=엑스)	X , 너비가 0 인 긍정적 예측을 통해
(?!엑스)	X , 너비가 0 인 부정적인 예측을 통해
(?<=엑스)	X , 너비가 0 인 긍정적 인 룩백을 통해
(?<!엑스)	X , 너비가 0 인 네거티브 lookbehind를 통해
(?>엑스)	X , 독립적 인 비 캡처 그룹

백 슬래시, 이스케이프 및 인용

백 슬래시 문자 ('₩')는 위의 표에 정의 된 이스케이프 구조를 도입하고 그렇지 않으면 이스케이프되지 않은 구조로 해석 될 문자를 인용하는 데 사용됩니다. 따라서 표현식 ₩₩은 단일 백 슬래시와 ₩{일치하고 왼쪽 중괄호 와 일치합니다.

이스케이프 된 구조를 나타내지 않는 알파벳 문자 앞에 백 슬래시를 사용하는 것은 오류입니다. 정규 표현식 언어에 대한 향후 확장을 위해 예약되어 있습니다. 문자가 이스케이프되지 않은 구조의 일부인지 여부에 관계없이 알파벳이 아닌 문자 앞에 백 슬래시를 사용할 수 있습니다.

Java 소스 코드의 문자열 리터럴 내의 백 슬래시 는 Java ™ 언어 사양에서 요구하는 대로 유니 코드 이스케이프 (섹션 3.3) 또는 기타 문자 이스케이프 (섹션 3.10.6)로 해석됩니다. 따라서 정규식을 나타내는 문자열 리터럴에서 이중 백 슬래시가 필요합니다. Java 바이트 코드 컴파일러에 의한 해석으로부터 보호합니다. "₩b"예를 들어, 문자열 리터럴 은 정규 표현식으로 해석 될 때 단일 백 스페이스 문자 "₩₩b"와 일치하는 반면 단어 경계와 일치합니다. 문자열 리터럴 "₩(hello₩)"이 잘못되어 컴파일 타임 오류가 발생합니다. 문자열을 일치 시키 (hello)려면 문자열 리터럴을 "₩₩(hello₩₩)" 사용해야합니다.

캐릭터 클래스

문자 클래스는 다른 문자 클래스 내에 나타날 수 있으며 공용체 연산자 (암시 적) 및 교차 연산자 (&&) 로 구성 될 수 있습니다 . 통합 연산자는 피연산자 클래스 중 하나 이상에있는 모든 문자를 포함하는 클래스를 나타냅니다. 교차 연산자는 두 피연산자 클래스 모두에있는 모든 문자를 포함하는 클래스를 나타냅니다.

문자 클래스 연산자의 우선 순위는 가장 높은 것에서 가장 낮은 것까지 다음과 같습니다.

1	리터럴 이스케이프	₩x
2	그룹화	[...]
삼	범위	a-z
4	노동 조합	[a-e][i-u]
5	교차로	[a-z&&[aeiou]]

다른 메타 문자 집합은 문자 클래스 외부가 아닌 문자 클래스 내부에 적용됩니다. 예를 들어, 정규 표현식 .은 문자 클래스 내에서 특별한 의미를 잃고 표현식 -은 메타 문자를 형성하는 범위가됩니다.

줄 종결 자

광고 터미네이터 는 단색 또는 두 문자 시퀀스 인 마크 즉 입력 문자 시퀀스의 줄 끝. 다음은 줄 종결 자로 인식됩니다.

- 줄 바꿈 (줄 바꿈) 문자 ('₩n'),
- 캐리지 리턴 문자 바로 뒤에 개행 문자 ("₩r₩n")가 옵니다 .
- 독립형 캐리지 리턴 문자 ('₩r'),
- 다음 줄 문자 ('₩u0085'),
- 줄 구분 문자 ('₩u2028') 또는
- 단락 구분 문자 ('₩u2029').

UNIX_LINES모드가 활성화 된 경우 인식되는 유일한 줄 종결자는 개행 문자입니다.

정규식 .은 DOTALL플래그가 지정 되지 않은 경우 줄 종결자를 제외한 모든 문자와 일치 합니다.

기본적으로 정규 표현식 ^및 $행 종결을 무시하고 단지 입력 순서 전체의 각각 시작과 끝 부분에 일치합니다. MULTILINE모드가 활성화 되면 ^입력 시작 부분과 입력 끝 부분을 제외한 모든 라인 종결 자 뒤에서 일치합니다. MULTILINE모드 $ 에서 라인 종결 자 바로 앞 또는 입력 시퀀스의 끝과 일치 할 때 .

그룹 및 캡처

그룹 번호

캡처 그룹은 여는 괄호를 왼쪽에서 오른쪽으로 세어 번호가 매겨집니다. ((A)(B(C)))예를 들어 표현식 에는 다음과 같은 4 개의 그룹이 있습니다.

1	((A)(B(C)))
2	(A)
삼	(B(C))
4	(C)

그룹 0은 항상 전체 표현식을 나타냅니다.

캡처 그룹은 일치하는 동안 해당 그룹과 일치하는 입력 시퀀스의 각 하위 시퀀스가 저장되기 때문에 이름이 지정됩니다. 캡처 된 하위 시퀀스는 나중에 역 참조를 통해 표현식에서 사용할 수 있으며 일치 작업이 완료되면 일치 자에서 검색 할 수도 있습니다.

그룹 이름

캡처 그룹에 "이름", a를 할당 한 named-capturing group다음 나중에 "이름"으로 역 참조 할 수도 있습니다 . 그룹 이름은 다음 문자로 구성됩니다. 첫 번째 문자는 letter.

- (~) 'A'까지 의 대문자 , 'Z''₩u0041''₩u005a'
- (~) 'a'까지 의 소문자 , 'z''₩u0061''₩u007a'
- (~) '0'까지 의 숫자 , '9''₩u0030''₩u0039'

A named-capturing group는 그룹 번호에 설명 된대로 여전히 번호가 지정됩니다 .

그룹과 연관된 캡처 된 입력은 항상 그룹이 가장 최근에 일치 한 하위 시퀀스입니다. 정량화로 인해 그룹이 두 번째로 평가되는 경우 두 번째 평가가 실패하면 이전에 캡처 된 값이 유지됩니다. 예를 들어, "aba"표현식 (a(b)?)+과 문자열 을 일치 시키면 그룹 2가로 설정됩니다 "b". 캡처 된 모든 입력은 각 일치가 시작될 때 삭제됩니다.

로 시작하는 그룹은 텍스트를 캡처하지 않고 그룹 합계에 포함되지 않는 (?순수 비 캡처 그룹 또는 명명 된 캡처 그룹입니다.

유니 코드 지원

이 클래스는 Unicode Technical Standard # 18 : Unicode Regular Expression 의 레벨 1 과 RL2.1 Canonical Equivalents를 준수합니다.

\u2014Java 소스 코드에서 와 같은 **유니 코드 이스케이프 시퀀스** 는 Java ™ 언어 사양의 섹션 3.3에 설명 된대로 처리됩니다 . 이러한 이스케이프 시퀀스는 정규식 구문 분석기에 의해 직접 구현되므로 파일 또는 키보드에서 읽은 식에서 유니 코드 이스케이프를 사용할 수 있습니다. 따라서 문자열 "\u2014"및 "\\u2014"은 같지 않지만 동일한 패턴으로 컴파일되며 16 진수 값이있는 문자와 일치합니다 0x2014.

유니 코드 문자는 구문에 설명 된대로 16 **진수 표기법** (16 진수 코드 포인트 값) 을 사용하여 직접 표현할 수도 있습니다. \x{...}예를 들어 보조 문자 U + 2011F는 \x{2011F}서로 게이트 쌍의 두 개의 연속 된 유니 코드 이스케이프 시퀀스 대신 로 지정 될 수 있습니다 \uD840\uDD1F.

유니 코드 문자 이름 은 명명 된 문자 구조 \N{... }에서 지원됩니다 . 예를 들어, \N{WHITE SMILING FACE} character를 지정합니다 \u263A. 이 클래스에서 지원하는 문자 이름은와 일치하는 유효한 유니 코드 문자 이름 Character.codePointOf(name)입니다.

유니 코드 확장 자소 클러스터 는 자소 클러스터 매처\X와 해당 경계 매처에의해 지원됩니다 \b{g}.

유니 코드 스크립트, 블록, 범주 및 이진 속성은 Perl에서 와 같이 \p및 \P구문으로 작성됩니다 . \p{소품} 입력이 속성이있는 경우 일치하는 소품을 하면서, \P{소품은} 입력이 해당 속성이 있으면 일치하지 않습니다.

스크립트, 블록, 범주 및 이진 속성은 문자 클래스 내부와 외부에서 모두 사용할 수 있습니다.

스크립트는 접두사 중 하나를 지정Is에서와 같이, IsHiragana또는 사용하여script키워드 (또는 짧은 형식을sc같이)script=Hiragana나sc=Hiragana.

에서 지원하는 스크립트 이름 Pattern은에서 허용하고 정의한 유효한 스크립트 이름 UnicodeScript.forName입니다.

블록이 접두사로 지정한In마찬가지로, InMongolian혹은 키워드를 이용하여block(또는 짧은 형태blk에서와 같이)block=Mongolian또는blk=Mongolian.

에서 지원하는 블록 이름 Pattern은에서 허용하고 정의한 유효한 블록 이름 UnicodeBlock.forName입니다.

카테고리 옵션 접두어로 지정 될 수있다Is: 모두\p{L}와\p{IsL}유니 코드 문자의 범주를 나타낸다. 스크립트 및 블록과 마찬가지로 카테고리는general_category또는gc에서 general_category=Lu와같이키워드(또는 짧은 형식)를사용하여 지정할 수도 있습니다gc=Lu.

지원되는 범주는 클래스에서 지정한 버전 의 유니 코드 표준 입니다 . 카테고리 이름은 표준에 정의 된 것으로 규범 적 및 정보 적 이름입니다. Character

이진 속성 은에서와 같이 접두사로 지정Is됩니다 IsAlphabetic. 지원되는 바이너리 속성은 다음Pattern 과 같습니다.

- 알파벳
- 표의 문자
- 편지
- 소문자
- 대문자
- 타이틀 케이스

- 구두
- 제어
- White_Space
- 숫자
- Hex_Digit
- Join_Control
- Noncharacter_Code_Point
- 할당 됨

다음 **사전 정의 된 문자 클래스** 및 **POSIX 문자 클래스** 는 플래그가 지정된 경우 부록 C : 유니 코드 정규식 의 호환성 속성 의 권장 사항을 준수 UNICODE_CHARACTER_CLASS합니다.

클래스	성냥
`Wp{Lower}`	소문자 : `Wp{IsLowercase}`
`Wp{Upper}`	대문자 : `Wp{IsUppercase}`
`Wp{ASCII}`	모든 ASCII : `[Wx00-Wx7F]`
`Wp{Alpha}`	알파벳 문자 : `Wp{IsAlphabetic}`
`Wp{Digit}`	10 진수 문자 : `p{IsDigit}`
`Wp{Alnum}`	영숫자 : `[Wp{IsAlphabetic}Wp{IsDigit}]`
`Wp{Punct}`	구두점 문자 : `p{IsPunctuation}`
`Wp{Graph}`	보이는 캐릭터 : `[^Wp{IsWhite_Space}Wp{gc=Cc}Wp{gc=Cs}Wp{gc=Cn}]`
`Wp{Print}`	인쇄 가능한 문자 : `[Wp{Graph}Wp{Blank}&&[^Wp{Cntrl}]]`
`Wp{Blank}`	공백 또는 탭 : `[Wp{IsWhite_Space}&&[^Wp{gc=Zl}Wp{gc=Zp}Wx0aWx0bWx0cWx0dWx85]]`
`Wp{Cntrl}`	제어 문자 : `Wp{gc=Cc}`
`Wp{XDigit}`	16 진수 : `[Wp{gc=Nd}Wp{IsHex_Digit}]`
`Wp{Space}`	공백 문자 : `Wp{IsWhite_Space}`
`Wd`	숫자 : `Wp{IsDigit}`
`WD`	숫자가 아닌 : `[^Wd]`
`Ws`	공백 문자 : `Wp{IsWhite_Space}`
`WS`	공백이 아닌 문자 : `[^Ws]`
`Ww`	워드 캐릭터 : `[Wp{Alpha}Wp{gc=Mn}Wp{gc=Me}Wp{gc=Mc}Wp{Digit}Wp{gc=Pc}Wp{IsJoin_Control}]`
`WW`	비 단어 문자 : `[^Ww]`

java.lang.Character boolean is methodname 메소드 (사용되지 않는 메소드 제외) 와 같이 작동하는 카테고리 는 지정된 속성에 name이있는 동일한 `Wp{prop}` 구문을 통해 사용할 수 있습니다`javamethodname` .

Perl 5와 비교

`Pattern`펄 5 발생할 때 엔진은 명령 교대 전통적인 NFA 기반의 정합을 수행한다.

이 클래스에서 지원하지 않는 Perl 구조 :

- 역 참조 구조, \g{N} 대한 N번째 그룹 캡처 및 \g{이름을} 위한 그룹 이름 포착 .
- 조건부는 (?(조건)X) 및 (?(조건)X |Y를 구성합니다) .
- 포함 된 코드는 (?{코드}) 와 코드를 구성 (??{합니다}) .
- 포함 된 주석 구문 (?#comment)및
- 전처리 작업 \l \u, \L및 \U.

이 클래스에서 지원하지만 Perl에서는 지원하지 않는 구성 :

- 위에서 설명한대로 문자 클래스 공용체 및 교차 .

Perl과의 주목할만한 차이점 :

- Perl에서 \1through \9는 항상 역 참조로 해석됩니다. 보다 큰 백 슬래시 이스케이프 된 숫자 9는 최소한 많은 하위 표현식이 존재하는 경우 역 참조로 처리됩니다. 그렇지 않으면 가능한 경우 8 진 이스케이프로 해석됩니다. 이 클래스에서 8 진 이스케이프는 항상 0으로 시작해야합니다. 이 클래스에서 \1through \9는 항상 역 참조로 해석되며 정규식의 해당 지점에 최소한 많은 하위식이 존재하면 더 큰 숫자가 역 참조로 허용됩니다. 그렇지 않으면 파서가 숫자가 더 작거나 같을 때까지 숫자를 삭제합니다. 기존 그룹 수 또는 한 자리입니다.
- Perl은 g플래그를 사용하여 마지막 일치가 중단 된 지점에서 다시 시작되는 일치를 요청합니다. 이 기능은 Matcher클래스에 의해 암시 적으로 제공됩니다 find. 매 처가 재설정되지 않는 한 메서드 의 반복 된 호출은 마지막 일치가 중단 된 지점에서 다시 시작됩니다.
- Perl에서 식의 최상위 수준에 포함 된 플래그는 전체 식에 영향을줍니다. 이 클래스에서 포함 된 플래그는 최상위 레벨에 있든 그룹 내에 있든 상관없이 항상 나타나는 지점에서 적용됩니다. 후자의 경우 플래그는 Perl에서와 같이 그룹의 끝에 복원됩니다.

정규식 구조의 동작에 대한보다 정확한 설명은 Mastering Regular Expressions, 3nd Edition , Jeffrey EF Friedl, O'Reilly and Associates, 2006을 참조하십시오.

이후:
1.4
또한보십시오:
String.split(String, int), String.split(String), 직렬화 된 형식

- ## 필드 요약

필드

수정 자 및 유형	들	기술
static int	CANON_EQ	표준 동등성을 활성화합니다.
static int	CASE_INSENSITIVE	대소 문자를 구분하지 않는 일치를 사용합니다.
static int	COMMENTS	패턴에 공백과 주석을 허용합니다.

static int DOTALL	dotall 모드를 활성화합니다.
static int LITERAL	패턴의 리터럴 구문 분석을 사용합니다.
static int MULTILINE	여러 줄 모드를 활성화합니다.
static int UNICODE_CASE	유니 코드 인식 대소 문자 접기를 사용합니다.
static int UNICODE_CHARACTER_CLASS	미리 정의 된 문자 클래스 및 POSIX 문자 클래스의 유니 코드 버전을 활성화합니다 .
static int UNIX_LINES	Unix 라인 모드를 활성화합니다.

◦ 방법 요약

모든 방법 정적 방법 인스턴스 방법 구체적인 방법

수정 자 및 유형	방법	기술
Predicate<String>	asPredicate()	문자열을 일치시키는 데 사용할 수있는 술어를 작성합니다.
static Pattern	compile(String regex)	주어진 정규식을 패턴으로 컴파일합니다.
static Pattern	compile(String regex, int flags)	지정된 플래그를 사용하여 지정된 정규식을 패턴으로 컴파일합니다.
int	flags()	이 패턴의 일치 플래그를 반환합니다.
Matcher	matcher (CharSequence input)	이 패턴에 대해 주어진 입력을 일치시킬 매처를 만듭니다.
static boolean	matches(String regex, CharSequence input)	주어진 정규식을 컴파일하고 이에 대해 주어진 입력을 일치 시키려고합니다.
String	pattern()	이 패턴이 컴파일 된 정규식을 반환합니다.
static String	quote(String s)	지정된에 대한 리터럴 패턴 String을 반환합니다 String.
String[]	split (CharSequence input)	이 패턴의 일치 항목을 중심으로 지정된 입력 시퀀스를 분할합니다.
String[]	split (CharSequence input, int limit)	이 패턴의 일치 항목을 중심으로 지정된 입력 시퀀스를 분할합니다.
Stream<String>	splitAsStream (CharSequence input)	이 패턴의 일치에 대해 지정된 입력 시퀀스에서 스트림을 만듭니다.
String	toString()	이 패턴의 문자열 표현을 리턴합니다.

▪ 클래스 java.lang. 목적

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

- ◦ 필드 세부 정보

▪ UNIX_LINES

```
public static final int UNIX_LINES
```

Unix 라인 모드를 활성화합니다.

이 모드에서 만 '₩n'라인 터미네이터의 행동으로 인식하고 ., ^하고 $.

Unix 라인 모드는 임베디드 플래그 표현식을 통해 활성화 할 수도 있습니다 (?d).

또한보십시오:
상수 필드 값

- **CASE_INSENSITIVE**

  public static final int CASE_INSENSITIVE

  대소 문자를 구분하지 않는 일치를 사용합니다.

  기본적으로 대 / 소문자를 구분하지 않는 일치는 US-ASCII 문자 집합의 문자 만 일치한다고 가정합니다. UNICODE_CASE이 플래그와 함께 플래그를 지정하면 유니 코드를 인식하는 대소 문자를 구분하지 않는 일치를 사용할 수 있습니다 .

  포함 된 플래그 표현식을 통해 대소 문자를 구분하지 않는 일치를 활성화 할 수도 있습니다 (?i).

  이 플래그를 지정하면 약간의 성능 저하가 발생할 수 있습니다.

  또한보십시오:
  상수 필드 값

- **코멘트**

  public static final int COMMENTS

  패턴에 공백과 주석을 허용합니다.

  이 모드에서는 공백이 무시되고로 시작하는 포함 된 주석은 #줄 끝까지 무시됩니다.

  주석 모드는 포함 된 플래그 표현식을 통해 활성화 할 수도 있습니다 (?x).

  또한보십시오:
  상수 필드 값

- **여러 줄**

  public static final int MULTILINE

  여러 줄 모드를 활성화합니다.

  여러 줄 모드에서 식과 일치는 각각 줄 종결 자 또는 입력 시퀀스의 끝 바로 뒤 또는 바로 앞 ^과 $일치합니다. 기본적으로 이러한 표현식은 전체 입력 시퀀스의 시작과 끝에서만 일치합니다.

  여러 줄 모드는 포함 된 플래그 표현식을 통해 활성화 할 수도 있습니다 (?m).

  또한보십시오:
  상수 필드 값

- **정확한**

  `public static final int LITERAL`

  패턴의 리터럴 구문 분석을 사용합니다.

  이 플래그가 지정되면 패턴을 지정하는 입력 문자열이 리터럴 문자 시퀀스로 처리됩니다. 입력 시퀀스의 메타 문자 또는 이스케이프 시퀀스에는 특별한 의미가 없습니다.

  CASE_INSENSITIVE 및 UNICODE_CASE 플래그는이 플래그와 함께 사용될 때 일치에 대한 영향을 유지합니다. 다른 플래그는 불필요 해집니다.

  리터럴 구문 분석을 활성화하기위한 포함 된 플래그 문자가 없습니다.

  이후:
  1.5
  또한보십시오:
  상수 필드 값

- **도트**

  `public static final int DOTALL`

  dotall 모드를 활성화합니다.

  dotall 모드에서 표현식 .은 줄 종결자를 포함한 모든 문자와 일치합니다. 기본적으로이 표현식은 줄 종결 자와 일치하지 않습니다.

  Dotall 모드는 포함 된 플래그 표현식을 통해 활성화 할 수도 있습니다 `(?s)`. (s"single-line"모드의 니모닉으로 Perl에서 호출됩니다.)

  또한보십시오:
  상수 필드 값

- **UNICODE_CASE**

  `공용 정적 최종 정수 UNICODE_CASE`

  유니 코드 인식 대소 문자 접기를 사용합니다.

  이 플래그를 지정하면 CASE_INSENSITIVE플래그에 의해 활성화 될 때 대소 문자를 구분하지 않는 일치 가 유니 코드 표준과 일치하는 방식으로 수행됩니다. 기본적으로 대 / 소문자를 구분하지 않는 일치는 US-ASCII 문자 집합의 문자 만 일치한다고 가정합니다.

  유니 코드 인식 대소 문자 접기는 포함 된 플래그 표현식을 통해 활성화 할 수도 있습니다 `(?u)`.

  이 플래그를 지정하면 성능이 저하 될 수 있습니다.

  또한보십시오:
  상수 필드 값

- **CANON_EQ**

  `public static final int CANON_EQ`

  표준 동등성을 활성화합니다.

  이 플래그가 지정되면 두 문자는 전체 표준 분해가 일치하는 경우에만 일치하는 것으로 간주됩니다. `"a₩u030A"`예를 들어 표현식 `"₩u00E5"`은이 플래그가 지정 될 때 문자열과 일치 합니다. 기본적으로 일치는 표준 동등성을 고려하지 않습니다.

  표준 동등성을 활성화하기위한 포함 된 플래그 문자가 없습니다.

  이 플래그를 지정하면 성능이 저하 될 수 있습니다.

  또한보십시오:
  상수 필드 값

- **UNICODE_CHARACTER_CLASS**

  `공개 정적 최종 정수 UNICODE_CHARACTER_CLASS`

  미리 정의 된 문자 클래스 및 POSIX 문자 클래스 의 유니 코드 버전을 활성화합니다 .

  이 플래그가 지정되면 (US-ASCII 전용) 사전 정의 된 문자 클래스 및 POSIX 문자 클래스 는 유니 코드 기술 표준 # 18 : 유니 코드 정규식 부록 C : 호환성 속성 을 준수 합니다.

  UNICODE_CHARACTER_CLASS 모드는 포함 된 플래그 표현식을 통해 활성화 할 수도 있습니다 `(?U)`.

  플래그는 UNICODE_CASE를 의미합니다. 즉, 유니 코드 인식 대소 문자 접기를 활성화합니다.

  이 플래그를 지정하면 성능이 저하 될 수 있습니다.

  이후:
  1.7
  또한보십시오:
  상수 필드 값

○ 방법 세부 정보

- **엮다**

  `공개 정적` 패턴 `컴파일 (` 문자열 `정규식)`

  주어진 정규식을 패턴으로 컴파일합니다.

  매개 변수 :
  `regex` -컴파일 할 표현식
  보고:
  패턴으로 컴파일 된 주어진 정규식
  던졌습니다 :

PatternSyntaxException -표현식의 구문이 유효하지 않은 경우

- **엮다**

  공개 정적 패턴 컴파일 (문자열 정규식,
  int 플래그)

  지정된 플래그를 사용하여 지정된 정규식을 패턴으로 컴파일합니다.

  매개 변수 :
  regex -컴파일 할 표현식
  flags- 경기 플래그, 비트 마스크가 포함 CASE_INSENSITIVE, MULTILINE, DOTALL, UNICODE_CASE, CANON_EQ, UNIX_LINES, LITERAL, UNICODE_CHARACTER_CLASS 및COMMENTS
  보고:
  주어진 플래그와 함께 패턴으로 컴파일 된 주어진 정규 표현식
  던졌습니다 :
  IllegalArgumentException -정의 된 일치 플래그에 해당하지 않는 비트 값이 flags
  PatternSyntaxException -표현식의 구문이 유효하지 않은 경우

- **무늬**

  공개 문자열 패턴 ()

  이 패턴이 컴파일 된 정규식을 반환합니다.

  보고:
  이 패턴의 근원

- 공용 문자열 toString ()

  이 패턴의 문자열 표현을 리턴합니다. 이 패턴이 컴파일 된 정규식입니다.

  재정의 :
  toString 클래스 Object
  보고:
  이 패턴의 문자열 표현
  이후:
  1.5

- **매처**

  public Matcher matcher (CharSequence 입력)

  이 패턴에 대해 주어진 입력을 일치시킬 매처를 만듭니다.

  매개 변수 :
  input -일치시킬 문자 시퀀스
  보고:
  이 패턴에 대한 새로운 매처

- **깃발**

공개 정수 플래그 ()

이 패턴의 일치 플래그를 반환합니다.

보고:
이 패턴이 컴파일 될 때 지정된 일치 플래그

- **성냥**

공개 정적 부울 일치 (문자열 정규식,
CharSequence 입력)

주어진 정규식을 컴파일하고 이에 대해 주어진 입력을 일치 시키려고합니다.

형식의이 편의 메서드 호출

```
Pattern.matches (정규식, 입력);
```

표현과 똑같은 방식으로 작동합니다.

```
Pattern.compile (regex) .matcher (input) .matches ()
```

패턴이 여러 번 사용되는 경우 한 번 컴파일하고 다시 사용하는 것이 매번이 메서드를 호출하는 것보다 더 효율적입니다.

매개 변수 :
regex -컴파일 할 표현식
input -일치시킬 문자 시퀀스
보고:
정규식이 입력과 일치하는지 여부
던졌습니다 :
PatternSyntaxException -표현식의 구문이 유효하지 않은 경우

- **스플릿**

public String [] split (CharSequence 입력,
int 제한)

이 패턴의 일치 항목을 중심으로 지정된 입력 시퀀스를 분할합니다.

이 메서드에서 반환 된 배열에는이 패턴과 일치하는 다른 하위 시퀀스로 종료되거나 입력 시퀀스의 끝으로 종료되는 입력 시퀀스의 각 하위 문자열이 포함됩니다. 배열의 하위 문자열은 입력에서 발생하는 순서를 따릅니다. 이 패턴이 입력의 하위 시퀀스와 일치하지 않는 경우 결과 배열에는 하나의 요소, 즉 문자열 형식의 입력 시퀀스 만 있습니다.

입력 시퀀스의 시작 부분에 양의 너비가 일치하는 경우 결과 배열의 시작 부분에 빈 선행 부분 문자열이 포함됩니다. 그러나 처음에 너비가 0 인 일치는 이러한 빈 선행 하위 문자열을 생성하지 않습니다.

limit파라미터는 패턴이 적용되는 횟수를 제어하기 때문에, 결과 어레이의 길이에 영향을 미친다. 제한 n 이 0보다 크면 패턴은 최대 n -1 회 적용되고 배열의 길이는 n 보다 크지 않으며 배열의 마지막 항목에는 마지막 일치 구분 기호를 초과하는 모든 입력이 포함됩니다. n 이 양수가 아닌 경우 패턴이 가능한 한 많이 적용

되고 배열의 길이는 제한되지 않습니다. 경우 n은 다음 패턴은 가능한 한 많이 적용되어 제로가되면, 배열의 길이는 공 문자열은 파기한다.

`"boo:and:foo"`예를 들어 입력 은 이러한 매개 변수를 사용하여 다음과 같은 결과를 생성합니다.

정규식, 제한 및 결과를 보여주는 분할 예제

정규식	한도	결과
:	2	`{ "boo", "and:foo" }`
:	5	`{ "boo", "and", "foo" }`
:	-2	`{ "boo", "and", "foo" }`
영형	5	`{ "b", "", ":and:f", "", "" }`
영형	-2	`{ "b", "", ":and:f", "", "" }`
영형	0	`{ "b", "", ":and:f" }`

매개 변수 :
`input` -분할 할 문자 시퀀스
`limit` -위에 설명 된 결과 임계 값
보고:
이 패턴의 일치 항목을 기준으로 입력을 분할하여 계산 된 문자열 배열

- **스플릿**

`public String [] split ( CharSequence 입력)`

이 패턴의 일치 항목을 중심으로 지정된 입력 시퀀스를 분할합니다.

이 메서드 `split`는 주어진 입력 시퀀스와 제한 인수가 0 인 두 인수 메서드를 호출하는 것처럼 작동 합니다. 따라서 후행 빈 문자열은 결과 배열에 포함되지 않습니다.

`"boo:and:foo"`예를 들어 입력 은 이러한 표현식으로 다음 결과를 생성합니다.

정규식	결과
:	`{ "boo", "and", "foo" }`
영형	`{ "b", "", ":and:f" }`

매개 변수 :
`input` -분할 할 문자 시퀀스
보고:
이 패턴의 일치 항목을 기준으로 입력을 분할하여 계산 된 문자열 배열

- **인용문**

`public static String quote ( String s)`

지정된에 대한 리터럴 패턴 `String`을 반환 합니다 `String`.

이 메서드는 리터럴 패턴 인 것처럼 문자열 과 일치하는 String을 만드는 데 사용할 수 있는를 생성합니다 .Patterns

입력 시퀀스의 메타 문자 또는 이스케이프 시퀀스에는 특별한 의미가 없습니다.

매개 변수 :
s -리터럴화할 문자열
보고:
리터럴 문자열 대체
이후:
1.5

■ asPredicate

public Predicate < String > asPredicate ()

문자열을 일치시키는 데 사용할 수있는 술어를 작성합니다.

보고:
문자열 일치에 사용할 수있는 술어
이후:
1.8

■ splitAsStream

public Stream < String > splitAsStream (CharSequence 입력)

이 패턴의 일치에 대해 지정된 입력 시퀀스에서 스트림을 만듭니다.

이 메서드가 반환하는 스트림에는이 패턴과 일치하는 다른 하위 시퀀스에 의해 종료되거나 입력 시퀀스의 끝으로 종료되는 입력 시퀀스의 각 하위 문자열이 포함됩니다. 스트림의 하위 문자열은 입력에서 발생하는 순서를 따릅니다. 후행 빈 문자열은 삭제되고 스트림에서 발견되지 않습니다.

이 패턴이 입력의 하위 시퀀스와 일치하지 않는 경우 결과 스트림에는 하나의 요소, 즉 문자열 형식의 입력 시퀀스 만 있습니다.

입력 시퀀스의 시작 부분에 양의 너비가 일치하면 빈 선행 부분 문자열이 스트림의 시작 부분에 포함됩니다. 그러나 처음에 너비가 0 인 일치는 이러한 빈 선행 하위 문자열을 생성하지 않습니다.

입력 시퀀스가 변경 가능한 경우 터미널 스트림 작업을 실행하는 동안 일정하게 유지되어야합니다. 그렇지 않으면 터미널 스트림 작업의 결과가 정의되지 않습니다.

매개 변수 :
input -분할 할 문자 시퀀스
보고:
이 패턴의 일치 항목을 기준으로 입력을 분할하여 계산 된 문자열 스트림
이후:
1.8
split(CharSequence)