PHP: esempi di uso delle espressioni regolari

Le espressioni regolari (regex) sono uno strumento potente per la manipolazione e la ricerca di testo in PHP. Sono usate per trovare e sostituire modelli specifici all'interno di stringhe e per validare l'input. In PHP, le regex sono gestite principalmente con la funzione `preg_`, come `preg_match`, `preg_replace` e `preg_split`. In questo articolo, esploreremo esempi avanzati di espressioni regolari in PHP e vedremo come utilizzarle in scenari complessi.

In PHP, esistono due modalità principali per gestire le regex:

- **PCRE (Perl Compatible Regular Expressions)**: Gestite con funzioni come `preg_match`, `preg_replace`, `preg_split`.
- **POSIX (Extended Regular Expressions)**: Obsoleta e non più consigliata.

Tra i flag più comuni ci sono:

- `i` per ignorare la distinzione tra maiuscole e minuscole,
- `m` per la modalità multilinea,
- `u` per il supporto a UTF-8.

Un esempio base è la ricerca di una parola all'interno di una stringa. Ad esempio, per cercare la parola "gatto" all'interno di una stringa possiamo usare `preg_match`:

```
$string = "Il gatto è sul tetto.";
if (preg_match("/gatto/", $string)) {
    echo "Trovato!";
}
```

In questo esempio, se la stringa contiene "gatto", viene stampato "Trovato!".

Ora entriamo nel vivo degli esempi avanzati, dove utilizzeremo vari pattern per risolvere casi più complessi.

Validazione di un indirizzo email complesso

Per la validazione di un'email, è importante rispettare i caratteri validi secondo le specifiche RFC. Un pattern avanzato per validare un'email potrebbe essere:

```
$email = "esempio.email@dominio.com";
$pattern = "/^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/";
if (preg_match($pattern, $email)) {
    echo "Email valida!";
} else {
    echo "Email non valida!";
}
```

Questa regex verifica che l'indirizzo inizi con lettere, numeri o simboli specifici, seguiti da un `@`, un dominio e un TLD di almeno 2 caratteri (es. `.com`).

Estrarre tutte le occorrenze di una parola

In PHP, per estrarre tutte le occorrenze di una parola o di un pattern, usiamo `preg_match_all`. Supponiamo di voler estrarre tutti i numeri di telefono in un testo:

```
$text = "Contattaci al numero 123-456-7890 o al numero 098-765-4321.";
$pattern = "/\d{3}-\d{3}-\d{4}/";
preg_match_all($pattern, $text, $matches);
print_r($matches[0]);
```

Qui il pattern `\d{3}-\d{3}-\d{4}` cerca numeri di telefono con formato `123-456-7890`. La funzione `preg_match_all` restituisce tutte le corrispondenze trovate.

Sostituzione avanzata con preg_replace_callback

Usiamo `preg_replace_callback` per sostituire modelli complessi in base a condizioni logiche. Supponiamo di voler mascherare tutte le cifre di una carta di credito tranne le ultime quattro:

```
$card = "1234 5678 9123 4567";
$maskedCard = preg_replace_callback(
    "/\b\d{4}\b/",
    function ($match) {
        return str_repeat("*", strlen($match[0]));
    },
    $card
);
echo $maskedCard;
```

In questo esempio, `preg_replace_callback` sostituisce ogni gruppo di quattro cifre con degli asterischi, tranne l'ultimo gruppo, che può essere mantenuto intatto.

Divisione di una stringa con delimitatori multipli

Per dividere una stringa in base a più delimitatori, utilizziamo `preg_split` con un pattern. Supponiamo di voler dividere una stringa ogni volta che incontriamo uno spazio, una virgola o un punto e virgola:

```
$string = "Questo è un testo, con più delimitatori; di esempio.";
$tokens = preg_split("/[\s,;]+/", $string);
print_r($tokens);
```

Il pattern `[\s,;]+` divide la stringa su spazi (`\s`), virgole e punti e virgola, restituendo un array con ogni singola parola.

Lookahead e lookbehind

I **lookahead** e **lookbehind** permettono di trovare un pattern senza includerlo nella selezione. Vediamo un esempio per trovare tutte le parole seguite da un punto esclamativo, ma escludendo il punto esclamativo nella selezione.

```
$text = "Salve! Come stai? Bene! Che bello!";
$pattern = "/\b\w+(?=!)/";
preg_match_all($pattern, $text, $matches);
print_r($matches[0]);
```

Qui `(?=!)` è un **positive lookahead** che cerca il punto esclamativo ma non lo include nel risultato. Questo pattern trova "Salve" e "Bene", lasciando fuori i punti esclamativi.

Ottimizzazione delle espressioni regolari

Quando si lavora con pattern complessi, è facile creare regex pesanti che rallentano il sistema. Per ottimizzare:

1. **Evitare quantificatori pesanti** come `.*` all'inizio di una regex, poiché catturano tutto e possono creare ambiguità.

2. **Utilizzare lookahead e lookbehind** per verificare condizioni senza aumentare l'output.
3. **Specificare pattern precisi** (es. `\d{4}` invece di `\d+`) per aumentare l'efficienza.

Parsing di dati strutturati

Consideriamo un caso in cui dobbiamo parsare dati strutturati, come un CSV. Supponiamo che ogni riga abbia il formato `"Nome, Cognome, Email, Telefono"`. Possiamo usare `preg_match_all` per estrarre ogni dato in base a gruppi di cattura.

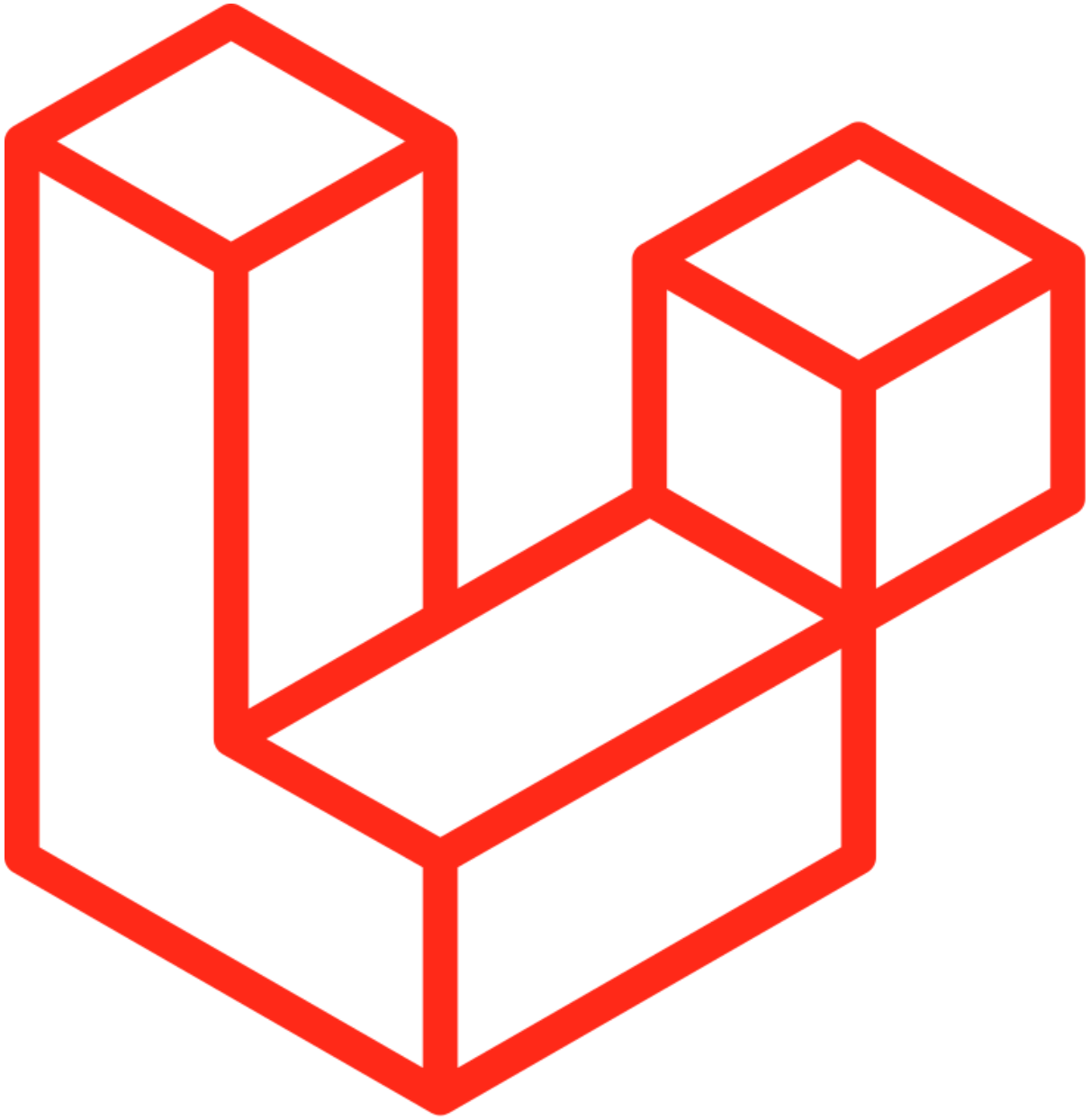
```
$data = "Mario, Rossi, mario.rossi@example.com, 123-456-7890\nAnna, Bianchi,
anna.bianchi@example.com, 098-765-4321";
$pattern = "/(\w+), (\w+), ([\w.%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}), (\d{3}-
\d{3}-\d{4})/";
preg_match_all($pattern, $data, $matches, PREG_SET_ORDER);

foreach ($matches as $match) {
    echo "Nome: {$match[1]}, Cognome: {$match[2]}, Email: {$match[3]},
Telefono: {$match[4]}\n";
}
```

Questa regex cattura nome, cognome, email e telefono come singoli gruppi, permettendo di parsare facilmente il CSV.

Conclusione

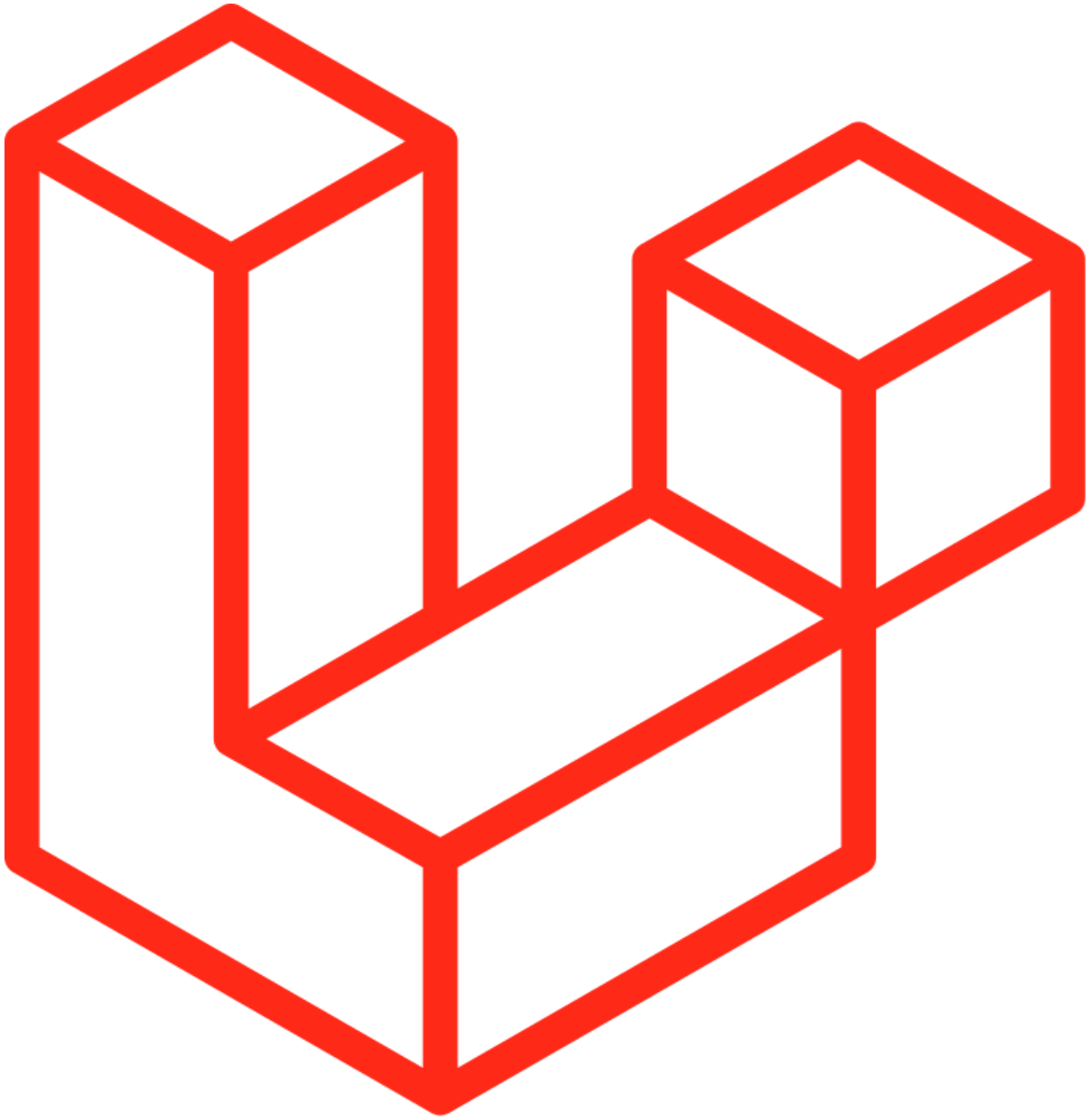
Le espressioni regolari in PHP offrono una vasta gamma di funzionalità per la manipolazione e la validazione del testo. Con un po' di pratica, è possibile utilizzarle per risolvere problemi complessi, dal parsing dei dati alla validazione di input strutturati. Sperimentare con pattern avanzati, lookahead/lookbehind e callback può aprire nuove possibilità per la gestione dei dati in PHP, migliorando le performance e l'affidabilità del codice.

Applicazioni Correlate



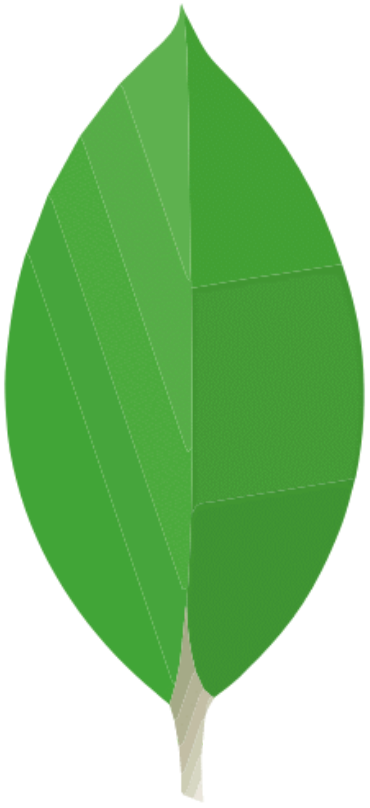
Laravel Placeholder Image

Applicazione in Laravel per creare immagini segnaposto.
DockerDocker ComposeComposerPHPLaravelJavaScript



Laravel Shopping Cart

Applicazione che fa parte del progetto Laravel E-commerce e implementa la gestione del carrello in Laravel.
DockerDocker ComposeComposerPHPLaravel



PHP MongoDB App

Applicazione basata su MongoDB con il driver PHP ufficiale.
DockerDocker ComposeComposerPHPMongoDB