`@prepend` and `@prependonce` in Laravel Blade

Both `@prepend` and `@prependonce` are directives used within Laravel Blade templates to manage the order of content added to sections. Here's a breakdown of their functionalities and key differences:

`@prepend`

- Inserts content **before** any existing content stacked using `@push` within a named section.
- **Multiple `@prepend` directives** can be used within the same section, and they'll be executed in the reverse order they appear in the template. This means the content from the last `@prepend` directive will be displayed first.
- Ideal for ensuring specific content appears at the top of a section, regardless of the order of `@push` directives used elsewhere.

Example:

HTML
```
@section('sidebar')
  @prepend('important')
    <h2>Important Links</h2>
    <ul>
      <li>Priority Link 1</li>
    </ul>
  @prepend

  @push('sidebar')
    <ul>
      <li>Link from View A</li>
      <li>Link from View B</li>
    </ul>
  @endpush
@endsection
```

Use code with caution.
content_copy

In this example, the "Important Links" section will be displayed first in the sidebar, followed by the links added using `@push`.

`@prependonce`

- Similar to `@prepend`, but it ensures the content is added **only once**, even if multiple `@prependonce` directives exist for the same section.
- Useful for preventing duplicate content within a section, especially when you have multiple Blade components contributing to the same section.

Example:

HTML
```
@section('messages')
  @prependonce('welcome')
    <h2>Welcome User!</h2>
    <p>This message will only be displayed once.</p>
  @prependonce
@endsection
```

Use code with caution.
content_copy

In this scenario, even if multiple views use `@prependonce` for the "welcome" message, it will only appear once in the final rendered section.

Choosing Between `@prepend` and `@prependonce`:

- Use `@prepend` when you want to add content before existing content, and the order of the content within the `@prepend` directives might change.
- Use `@prependonce` when you need to ensure content is added only once, even if multiple views use the directive for the same section.

Additional Considerations:

- These directives work in conjunction with `@section` and `@yield` for defining and rendering named sections in Blade templates.
- They can be nested within each other and with `@push` directives to achieve more complex content organization.

`@include` and `@extends` in Laravel Blade

Both `@include` and `@extends` are directives used within Laravel Blade templates to manage how Blade files interact with each other. Here's a breakdown of their functionalities and key differences:

`@include`

- **Purpose:** Inserts the content from a separate Blade template file into the current view.
- **Functionality:**
  - Takes the name of the Blade file to include as an argument.
  - The content from the included file is inserted at the location where the `@include` directive appears in the current view.
- **Benefits:**
  - Promotes code reuse by allowing you to create modular components that can be included in multiple views.
  - Helps organise your views into smaller, more manageable files.
  - **Example:**
  - HTML
  - <header>
  -     @include('header')
  - </header>
  -
  - <main>
  -     </main>
  -
  - <footer>
  -     @include('footer')
  - </footer>
  -
  - <h1>My Application</h1>
  - <nav>
  -     </nav>
  -
  - Use code with caution.
  - content_copy
  - **Example**

```
<header>

   @include('header')

</header>



<main>
```

```
    </main>


<footer>

    @include('footer')

</footer>


<h1>My Application</h1>

<nav>

    </nav>
```

In this example, `main_view.blade.php` includes the content of `header.blade.php` and `footer.blade.php` at the designated locations.

`@extends`

- **Purpose:** Inherits the layout from another Blade template file.
- **Functionality:**
    - Takes the name of the layout Blade file as an argument.
    - The current view acts as a child view, extending and overriding specific sections within the parent layout.
- **Benefits:**
    - Enables creation of consistent layouts for your application by defining a base structure in the parent layout.
    - Allows child views to customise specific content sections within the inherited layout.
    -

Choosing Between `@include` and `@extends`:

- Use `@include` when you want to incorporate a small, reusable component into a view.
- Use `@extends` when you want to build a view upon a base layout template with customizable sections.

@push

Purpose:

- The `@push` directive allows you to accumulate content for a specific section defined in your Blade templates.
- This enables you to dynamically build view components from multiple sources and control their order within a section.

Functionality:

1. **Section Definition:**
    - You define a section using the `@section('name')` directive within your main Blade layout or parent view. This section serves as a placeholder where content can be stacked from other views.
2. **Content Stacking with `@push`:**
    - Other Blade components or views can use `@push('section_name')` to add content to the defined section. Each `@push` directive adds content to a stack. The order in which `@push` directives appear determines the final order of the content in the section.

`@section`:

- **Purpose:** Defines a named placeholder within a Blade template. This section serves as a designated area where content can be dynamically added or replaced from other views.
- **Syntax:** `@section('name')` ... content ... `@endsection`
    - Replace `'name'` with a unique identifier for the section.
    - The content placed between `@section` and `@endsection` will be the placeholder for dynamically added content.

`@stack`:

- **Purpose:** Used within a defined section to access and render content that has been previously added using `@push` directives from other views.
- **Syntax:** `@stack('name')`
    - Replace `'name'` with the same name used in the corresponding `@section` directive.
    - HTML
    - @section('sidebar')   @stack('sidebar') @endsection

`@yield` is a directive that serves as a placeholder within a parent layout for content defined in a child view using `@section`. It's a crucial element in creating reusable layouts with dynamic content sections.

The `yield` is a powerful tool for creating reusable layouts. By defining sections in a base layout with `yield` and filling those sections in child views with `@section`, you can efficiently manage and organise your view content, ensuring consistency across your application's pages.