Hooks

Index:

summarize concept of hooks within Frappe framework :

- Hooks are powerful feature & way of **extending** or **modifying** core functionality/events of the framework without changing the source code.
- override the standard implementation or extend it ..
- Customization..
- without changing the source code

definition:

- Hooks are defined in a file **"hooks.py"** in each app..
- **hooks.py** : ⇒ used to define/register various hooks.. "which are essentially Python functions path that get executed at specific Points...

Hook Executions:

- Hooks: resolved using a "last writer wins" strategy
- ⇒ (last installed app on the site will have the highest priority over others)
- When you call frappe.get_hooks, it will return ⇒ all the values in a List[ ]..

  ⇒ if the hook is defined in multiple apps, the values will be collected from those apps..
- The hook value can be handled in different ways.

  ## **For example:**

  ⇒ for including JS assets/using app_include_js: >>> **All of the values are included**.. **Way work? ⇒ When specified event occurs, Frappe iterates through the list of registered hooks and executes them in the order they are defined.**

  ⇒ But for overriding DocType Class/Whitelisted method: >>> **the last value in the list is used**..

Flexibility & Modularsations:

- It follows principle of separation of concerns and allows for easier maintenance and upgrade of the application.
- provide flexible way to modify/extend the behavior of Frappe applications.. allow you to customize the appl without modifying the core code, making it easier to update and maintain your customizations as the framework evolves.
- The concept of hooks is based on the observer pattern, where you define specific hooks in your custom code and register them to be executed when certain events occur within the Frappe framework .
- Hooks in Frappe are implemented using Python decorators and are typically defined in Python modules .
- A decorator in Python is a design pattern that allows you to modify the behavior of a function or class without changing its source code directly .
- Decorators are defined using the "@" symbol followed by the decorator name, which is then placed above the function or class definition.

Features | Common Uses Case:

⇒ to inject custom code,

⇒ to inject assets,

⇒ to inject data, custom fields,

⇒ to add scripts,

⇒ to add templates, webhooks, or workflows to your app.

Resources:

https://frappeframework.com/docs/user/en/python-api/hooks