



在 `AppServiceProvider` 的 `register` 方法中，我们使用 `singleton` 方法注册服务，确保服务只被实例化一次。

在 `AppServiceProvider` 中：

- 在 `AppServiceProvider` 的 `register` 方法中，我们使用 `singleton` 方法注册服务，确保服务只被实例化一次。
- 在 `AppServiceProvider` 的 `register` 方法中，我们使用 `singleton` 方法注册服务，确保服务只被实例化一次。

```
namespace App\Providers;

use Illuminate\Support\ServiceProvider;
use App\Services\SomeService;

class AppServiceProvider extends ServiceProvider
{
    /**
     * Register any application services.
     *
     * @return void
     */
    public function register()
    {
        $this->app->singleton(SomeService::class, function ($app) {
            return new SomeService();
        });
    }
}
```

- 在 `SomeService` 类中，我们使用 `SomeService` 类。
- 在 `SomeService` 类中，我们使用 `SomeService` 类。

```
use App\Services\SomeService;

class SomeController extends Controller
{
    protected $someService;

    public function __construct(SomeService $someService)
```

```

{
    $this->someService = $someService;
}

public function someMethod()
{
    // $someService 객체 참조.
    $this->someService->performAction();
}
}

```

이 `SomeController` 클래스는 `SomeService` 클래스를 `SomeController` 클래스 내부에 참조하고 있습니다.

이 경우: 서비스 객체 참조를 통해 서비스 객체를 참조할 수 있습니다. 이 경우 이 객체는 서비스 객체 참조를 통해 참조됩니다.

이 경우:

1. `Illuminate\Support\ServiceProvider` 클래스를 상속합니다. 이 클래스는 서비스 객체를 참조할 수 있는 객체 참조를 참조합니다, 이 경우 이 객체는 서비스 객체 참조를 통해 참조됩니다.
2. `Illuminate\Http` 클래스를 상속합니다, 이 경우 이 객체는 서비스 객체를 참조할 수 있습니다. 이 경우 이 객체는 서비스 객체 참조를 통해 참조됩니다.
3. `Illuminate` 클래스를 상속합니다.

```

namespace App\Providers;

use Illuminate\Support\ServiceProvider;
use App\Services\LoggerService;

class AppServiceProvider extends ServiceProvider
{
    /**
     * Register any application services.
     *
     * @return void
     */
    public function register()
    {
        $this->app->singleton(LoggerService::class, function ($app) {
            return new LoggerService();
        });
    }
}

```

이 `LoggerService` 클래스는 서비스 객체를 참조합니다. 이 경우 이 객체는 서비스 객체 참조를 통해 참조됩니다.

```
namespace App\Http\Controllers;

use App\Services\LoggerService;

class SomeController extends Controller
{
    protected $logger;

    public function __construct(LoggerService $logger)
    {
        $this->logger = $logger;
    }

    public function logSomething()
    {
        $this->logger->log('This is a log message.');
```

[illegible]

□□ □□: □□□□ □□ □□□□ □□ □□ □□ □□□□, □□□□ □□□□ □□ □ □□□ □□□□ □□ □□□□ □□□□, □□, □□□ □□ □□□ □□



1. `new SomeClass()` `SomeClass` `.someMethod()`.
2. `@RequestMapping`
  - HTTP `POST` `/api/users`.
  - `SomeController` `handlePostRequest()`:

```
namespace App\Http\Controllers;

use App\Services\SomeService;

class SomeController extends Controller
{
    protected $someService;

    public function __construct(SomeService $someService)
    {
```

```

        $this->someService = $someService;
    }

    public function handleRequest()
    {
        // ...
    }
}

```

- `SomeController` 依赖 `SomeService`，所以 `SomeController` 需要 `SomeService`。

### 3. 配置服务提供者

- 在 `AppServiceProvider` 中注册 `SomeService`。
- `AppServiceProvider` 实现 `SomeService` 的提供者接口：

```

namespace App\Providers;

use Illuminate\Support\ServiceProvider;
use App\Services\SomeService;

class AppServiceProvider extends ServiceProvider
{
    public function register()
    {
        $this->app->singleton(SomeService::class, function ($app) {
            return new SomeService();
        });
    }
}

```

### 4. 1000 个服务提供者

- 配置服务提供者
  - `SomeController` 依赖 `SomeService`，所以 `SomeController` 需要 `SomeService`。
  - 1000 个 `SomeController`，1 个 `SomeService` 提供者。
- 配置服务提供者
  - `SomeService` 提供者。
  - 1000 个 `SomeService` 提供者。

1:

SomeController 依赖 SomeService

SomeController 依赖 SomeService

2:



```

        if (self::$instance === null) {
            self::$instance = new self();
        }

        return self::$instance;
    }
}

```

4. **Singleton pattern** is a design pattern that restricts the instantiation of a class to one object. This is useful when exactly one object is needed to coordinate actions across the system.
- **Singleton pattern** is a design pattern that restricts the instantiation of a class to one object.

```

interface SomeServiceInterface
{
    public function performAction();
}

class SomeService implements SomeServiceInterface
{
    public function performAction()
    {
        // ...
    }
}

class AppServiceProvider extends ServiceProvider
{
    public function register()
    {
        $this->app->singleton(SomeServiceInterface::class, SomeService::class);
    }
}

```

**Q1:** What is the Singleton pattern? Why is it used?

**Q2:** How does the Singleton pattern ensure only one instance exists?

**Q3:** What are the advantages and disadvantages of the Singleton pattern?

You wanna more detailed information?

Revision #3

Created 4 August 2024 00:05:04 by [\[Link\]](#)

Updated 4 August 2024 00:22:55 by [\[Link\]](#)