Spring Bean Scope Explained

give me the deep and detailed explaination of Spring Bean Scope in spring boot so that i can able to understand it well and do not have any type of doubt in this.

so give me the deep and detailed explaination

Piyush, since you are deeply learning **Spring Boot internals** and backend concepts, understanding **Spring Bean Scope** clearly is extremely important. I'll explain this **from zero to advanced**, step-by-step, with **real-life analogies, lifecycle behavior, memory impact, threading**, and **when to use what** — so that **no doubt remains**.

🌱 Spring Bean Scope in Spring Boot — Deep & Complete Explanation

---

1️⃣ What is a Spring Bean?

Before scope, let's lock this concept clearly.

A **Spring Bean** is:

> An **object** that is **created, managed, and destroyed** by the **Spring IoC container**.

Spring creates these objects using:

- `@Component`
- `@Service`
- `@Repository`
- `@Controller`
- `@Bean`

These beans live inside the **Spring container**.

📦 The container in Spring Boot is:

- `ApplicationContext`

👉 `ApplicationContext` is the **brain** that:

- Creates beans
- Injects dependencies
- Manages lifecycle
- Applies scopes

2️⃣ What is Bean Scope?

Bean Scope defines:

> **How many instances** of a bean Spring creates
> **When they are created**
> **How long they live**
> **Who shares them**

📌 In short:

> **Scope controls the lifecycle and visibility of a bean**

3️⃣ Why Do We Need Bean Scope?

Imagine this situation 👇

- You have a **UserService**
- 1000 users hit your API at the same time

❓ Should Spring:

- Create **1 UserService object** and share it?
- OR create **1000 UserService objects**?

👉 The answer depends on **scope**

4️⃣ Default Scope in Spring Boot

🔹 Default Scope = `singleton`

If you do **nothing**, Spring uses **singleton scope**.

```java
@Service
public class UserService {
}
```

- ✔ Only **ONE object**
- ✔ Shared across **entire application**
- ✔ Created at **application startup**

5️⃣ Types of Bean Scopes in Spring Boot

Spring supports **two categories**:

🟢 Core Scopes (Always Available)

1. `singleton`
2. `prototype`

🌐 Web-Aware Scopes (Only in Web Applications)

3. `request`
4. `session`
5. `application`
6. `websocket`

🧠 Let's Understand Each Scope Deeply

6️⃣ Singleton Scope (DEFAULT)

📌 Definition

> **One bean instance per Spring container**

🧪 Example

```java
```

```
@Component
@Scope("singleton")
public class PaymentService {
}
```

🔍 Behavior

- Only **1 object** is created
- Shared across:
  - All users
  - All requests
  - All threads

📦 Lifecycle

1. Application starts
2. Bean is created
3. Bean stays alive till app shuts down
4. Bean destroyed on shutdown

🧠 Memory Impact

✔ Very memory efficient
✔ Faster performance

⚠️ Thread Safety (VERY IMPORTANT)

Because multiple threads use the **same instance**:

- Avoid mutable instance variables
- Prefer stateless design

❌ BAD

java

```
@Service
public class CounterService {
    int count = 0;   // NOT thread-safe
}
```

✔ GOOD

java

```
@Service
public class CounterService {
    public int calculate(int a, int b) {
        return a + b;
    }
}
```

✅ When to Use Singleton

- Services
- Repositories
- Controllers
- Utility classes
- Stateless logic

👉 **90% of Spring beans are singleton**

7️⃣ Prototype Scope

📌 Definition

> **New bean instance every time it is requested**

💉 Example

java

```
@Component
@Scope("prototype")
public class ReportGenerator {
}
```

🔍 Behavior

- Every `getBean()` → new object
- NOT shared
- Spring **does not manage destruction**

📦 Lifecycle

1. Bean requested

2. New instance created
3. Injected
4. Spring **forgets it**

❗ Spring does **NOT** call `@PreDestroy`

🧠 Memory Impact

❌ More memory usage
❌ Slower than singleton

⚠️ Important Limitation

If prototype is injected into singleton:

```java
@Service
public class InvoiceService {
    @Autowired
    private ReportGenerator reportGenerator;
}
```

➡️ Only **ONE instance** is injected
❌ Prototype behavior is LOST

✅ Solution

Use:

- `ObjectFactory`
- `Provider`
- `ApplicationContext`

```java
@Autowired
private ApplicationContext context;

public void generate() {
    ReportGenerator rg = context.getBean(ReportGenerator.class);
}
```

✅ When to Use Prototype

- Stateful objects
- Temporary objects
- Objects with user-specific data

8️⃣ Request Scope (Web Only)

📌 Definition

> **One bean per HTTP request**

🧪 Example

```java
@Component
@Scope(value = "request", proxyMode = ScopedProxyMode.TARGET_CLASS)
public class RequestTracker {
}
```

🔍 Behavior

- New bean for each HTTP request
- Destroyed after request completes

📦 Lifecycle

```nginx
HTTP Request → Bean Created
Request Ends → Bean Destroyed
```

🧠 Use Case

- Request-specific data
- Request logging
- Correlation IDs

⚠️ Why proxyMode?

Because request beans are injected into singleton controllers

---

9️⃣ Session Scope (Web Only)

📌 Definition

> One bean per HTTP session

🧪 Example

```java
@Component
@Scope(value = "session", proxyMode = ScopedProxyMode.TARGET_CLASS)
public class UserSession {
}
```

🔍 Behavior

- Created when session starts
- Destroyed when session expires

🧠 Use Case

- Logged-in user info
- Shopping cart
- User preferences

🔟 Application Scope (Web Only)

📌 Definition

> One bean per ServletContext

🧪 Example

```java
@Component
@Scope(value = "application")
public class AppConfigCache {
}
```

🔍 Behavior

- One instance per web application
- Similar to singleton but **web-specific**

🧠 Use Case

- Shared configuration
- App-level caching

1️⃣1️⃣ WebSocket Scope

📌 Definition

> **One bean per WebSocket session**

Used in:

- Real-time apps
- Chat applications

1️⃣2️⃣ Summary Table (Very Important)

Scope	Instances	Lifetime	Use Case
singleton	1	Entire app	Services, Controllers
prototype	Many	Caller-managed	Temporary objects
request	1 per request	HTTP request	Request data
session	1 per session	HTTP session	User data
application	1 per app	App lifetime	Global data
websocket	1 per socket	WebSocket session	Real-time apps

1️⃣3️⃣ Interview-Level Understanding

🔥 Key Interview Points

- Default scope is `singleton`

- Singleton beans must be thread-safe
- Prototype beans are NOT fully managed
- Request/session need proxyMode
- Prototype inside singleton is dangerous
- Choose scope based on lifecycle needs

1️⃣4️⃣ Best Practices (VERY IMPORTANT)

✔ Prefer **singleton**
✔ Keep beans **stateless**
✔ Use **prototype carefully**
✔ Use **request/session only when needed**
✔ Avoid heavy objects in session scope

1️⃣5️⃣ Final Mental Model 🧠

Think like this:

> **Bean Scope = Who owns the object and how long?**

- App owns → singleton
- Caller owns → prototype
- Request owns → request
- Session owns → session

If you want, next I can explain:

- 🔥 **Bean Lifecycle vs Scope**
- 🔥 **Scoped Proxy Internals**
- 🔥 **Thread safety in Spring Beans**
- 🔥 **Real-world project design using scopes**

Just tell me 👍