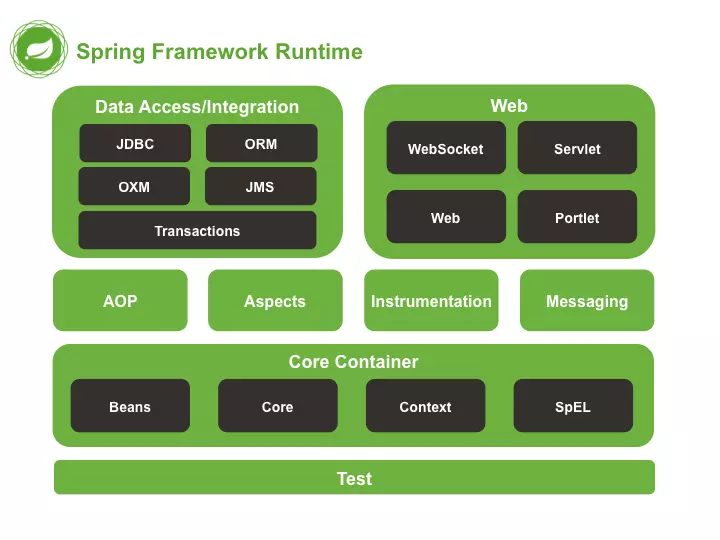
**BÁO CÁO VỀ SPRINGBOOT**

**I/ Springboot là gì?**

1. **Spring framework**

* Framework là 1 bộ công cụ và thư viện phát triển được thiết kế để giúp việc xây dựng ứng dụng trở nên dễ dàng hơn. Nó cung cấp 1 cấu trúc, các tiêu chuẩn và các công cụ giúp lập trình viên tập trung vào việc triển khai business logic thay vì phải viết lại các thành phần cốt lõi của ứng dụng.
* Spring là một framework phát triển các ứng dụng Java được sử dụng phổ biến. Nó giúp tạo các ứng dụng có hiệu năng cao, dễ kiểm thử, sử dụng lại code, …
* Spring được chia làm nhiều module khác nhau, tùy theo mục đích phát triển ứng dụng mà sử dụng.



* Core Container
* spring-core và spring-beans: Cung cấp tính năng cơ bản IoC và DI.
* spring-context: Xây dựng trên 1 nền tảng vững chắc được cung cấp bởi các module Core và Beans, được sử dụng để truy cập vào các đối tượng trong framework-style tương tự như việc đăng ký một JNDI. spring-context-support hỗ trợ tích hợp libraries của bên thứ 3 (third-party) vào ứng dụng Spring như caching (EhCache, Guava, JCache), mailing (JavaMail), schedule (CommonJ, Quatz) và các template engine (FreeMarker, JasperReports, Velocity)).
* spring-expression (Spring Expression Language): Được mở rộng từ Expresion Language trong JSP. Nó cung cấp hỗ trợ việc setting/getting giá trị, các method cải tiến cho phép truy cập collections, index, các toán tử logic…
* Data Access/Integration
* spring-jdbc: Cung cấp 1 lớp JDBC-abstraction để loại bỏ những code tẻ nhạt cả JDBC và phân tích những mã lỗi cụ thể của database-vendor.
* spring-orm: Cung cấp lớp tích hợp với các orm API phổ biến như JPA, JDO và Hibernate.
* spring-oxm: Cung cấp lớp abstraction hỗ trợ triển khai Object/XML mapping như AXB, Castor, XMLBeans, JiBX and XStream.
* spring-jms: Chứa các tính năng tạo và sử dụng các message. Từ Spring Framework 4.1, nó được tích hợp với spring-messaging.
* spring-transaction: Hỗ trợ quản lý giao dịch theo chương trình và khai báo cho các lớp mà thực hiện các giao diện đặc biệt và cho tất cả các POJO của bạn.
* Web
* spring-web: Cung cấp tính năng tích hợp web theo định hướng cơ bản như chức năng tập tin tải lên nhiều phần dữ liệu và khởi tạo các container IoC sử dụng nghe servlet.
* spring-webmvc: Bao gồm việc triển khai Model-View-Controller (MVC) của Spring cho các ứng dụng web.
* spring-websocket: Cung cấp hỗ trợ cho WebSocket-based, giao tiếp hai chiều giữa client và server trong các ứng dụng web.
* springwebmvc-portlet cung cấp việc triển khai MVC được sử dụng trong môi trường portlet và ánh xạ chức năng của module Web-Servlet.
* Các module quan trọng khác
* AOP: Hỗ trợ cài đặt lập trình hướng khía cạnh (Aspect Oriented Programming)
* Aspects: Cung cấp tích hợp với AspectJ, là 1 khuôn khổ AOP mạnh mẽ.
* Instrumentation: Cung cấp thiết bị đo đạc lớp hỗ trợ và triển khai lớp bộ nạp được sử dụng trong các máy chủ ứng dụng nhất định.
* Messaging: Cung cấp hỗ trợ cho STOMP như WebSocket sub-protocol để sử dụng trong các ứng dụng. Nó cũng hỗ trợ một mô hình lập trình chú thích cho việc định tuyến và xử lý tin nhắn STOMP từ các máy khách WebSocket.
* Test: Cung cấp khả năng hỗ trợ kiểm thử với JUnit và TestNG.
* Dựa trên các nguyên tắc thiết kế cơ bản của spring core. Spring còn phát triển nhiều project con như:
* Spring MVC: Dành cho việc xây dựng các ứng dụng nền tảng web.
* Spring Security: Cung cấp các cơ chế authentication và authorization cho ứng dụng.
* Spring Boot: Giúp phát triển cũng như chạy ứng dụng một cách nhanh chóng.
* Spring Batch: Giúp dễ dàng tạo các lịch trình (scheduling) và tiến trình (processing) cho các công việc xử lý theo mẻ (batch job).
* Spring Social: Cho phép kết nối ứng dụng với các API bên thứ ba của Facebook, Twitter, Linkedin, ...
* ...
* Spring Framework được xây dựng dựa trên 2 nguyên tắc design chính: DI và AOP
* Dependency Injection (DI)

***Nhắc lại về nguyên tắc Dependency Inversion trong SOLID***

*“Các module cấp cao không nên phụ thuộc vào module cấp thấp, cả hai nên phụ thuộc vào interface.*

*Các class giao tiếp với nhau thông qua interfce, không phải thông qua implementation”*

IoC được tạo ra nhằm tuân thủ theo nguyên lý này

***IoC là gì?***

* IoC (Inversion of Control) là một nguyên tắc trong lập trình, trong đó quyền kiểm soát các đối tượng và luồng của chương trình được chuyển từ phần mềm chính tới một framework hoặc container. Thay vì phần mềm gọi các hàm hoặc khởi tạo các đối tượng, thì framework sẽ gọi các hàm và khởi tạo các đối tượng để tiếp quản và điều khiển luồng của chương trình. Điều này giúp tăng tính linh hoạt, dễ bảo trì và tái sử dụng mã nguồn
* IoC Container là thành phần thực hiện IoC. Trong Spring, Spring IoC (IoC Container) là một phần quan trọng trong Spring để quản lý vòng đời và dependencies của các đối tượng trong ứng dụng. Nó có vai trò quản lý đối tượng, có trách nhiệm tạo ra, cấu hình và quản lý các đối tượng bean của ứng dụng. Thay vì tái tạo đối tượng bằng cách gọi trực tiếp từ mã, ta chỉ cần định nghĩa các đối tượng trong cấu hình (tệp XML hoặc thông qua Java Config) và container sẽ tự động khởi tạo, khởi tạo dependencies, quản lý và giải phóng tài nguyên khi không cần thiết nữa. Nó cũng đảm bảo dependencies được đưa vào bean một cách tự động, dễ dàng thiết lập và thay thế.
* Container (IoC container sử dụng DI để quản lý) có 2 loại là: BeanFactory và ApplicationContext (extends từ interface BeanFactory, nhưng mở rộng hơn). BeanFactory tạo bean khi gọi getBean(), còn ApplicationContext tự động tạo sẵn
* Dependency được đặt tên là: Bean, có nhiều loại bean như: @Scope(“<loại bean>”)
* Singleton (deafault): 1 bean được tạo duy nhất một lần từ class
* Prototype: return 1 bean riêng biệt cho mỗi lần sử dụng
* Request: Tạo mỗi bean 1 request
* Session: Tạo mỗi bean 1 session
* Global session:
* Spring Framework sử dụng khái niệm "bean" để đại diện cho các đối tượng và được quản lý bởi Spring Container.
* Springboot có thể tự động quét tất cả các cấu hình và các bean nhờ vào các annotation.
* Các Bean đều được tạo một lần duy nhất, chỉ có 1 instance => Khái niệm singleton pattern => Các bean đều là singleton
* Singleton giải quyết 2 vấn đề:
* Một class chỉ có một instance duy nhất
* Cung cấp một biến truy cập toàn cục cho instance đó

***DI là gì?***

* DI (Dependency Injection) là một cách triển khai của IoC. Nó tập trung vào việc cung cấp các phụ thuộc (dependencies) cho một đối tượng từ bên ngoài. Thay vì đối tượng tự tạo hoặc tìm kiếm những đối tượng phụ thuộc của nó, các phụ thuộc được chuyển vào từ bên ngoài. Các phụ thuộc này có thể là các đối tượng, các giao diện, hoặc các giá trị cấu hình. DI giúp tách biệt các thành phần và làm cho chúng dễ dàng thay thế và kiểm soát.

|  |  |
| --- | --- |
| Thay vì dùng như này: | Nên dùng như này: |
| *public class UserRepository {*  *private DatabaseConnection dbConnection;*  *public UserRepository () {*  *this.dbConnection = new DatabaseConnection();*  *}*  *}*  *UserRepository trực tiếp tạo ra và nắm quyền điều khiển DatabaseConnection. Gây bất tiện khi muốn thay đổi* | *public class UserRepository {*  *private DatabaseConnection dbConnection;*  *public UserRepository (DatabaseConnection dbConnection) {*  *this.dbConnection = dbConnection;*  *}*  *}*  *UserRepository không tạo trực tiếp một đối tượng DatabaseConnection, nhưng nó được cung cấp từ bên ngoài qua constructor của UserRepository. Điều này làm linh hoạt trong việc thay đổi mà không ảnh hưởng đến UserRepository.* |

* Aspect Oriented Programming (AOP)

Aspect Oriented Programming (AOP) Spring Framwork là 1 kỹ thuật lập trình dùng để tách logic chương trình thành các phần riêng biệt…

Trong Spring AOP, có 4 loại advice được hỗ trợ:

* Before advice: chạy trước khi method được thực thi
* After returning advice: Chạy sau khi method trả về một kết quả
* After throwing adivce: Chạy khi method ném ra một exception
* Around advice: Chạy khi method được thực thi (Bao gồm cả 3 loại advice trên)

Khi hoạt động, chương trình sẽ kết hợp các module lại để thực hiện các chức năng nhưng khi sửa đổi 1 chức năng thì chỉ cần sửa 1 module.

AOP không phải dùng để thay thế OOP mà để bổ sung cho OOP.

1. **Springboot**

***Tại sao phải sử dụng springboot?***

Vấn đề đặt ra như sau: Khi 1 project có quá nhiều dependency và ta phải inject chúng, điều này gây ra sự khó khăn, vất vả. Việc xuất hiện của springboot như 1 kho chứa các dependencies và sẽ tự động inject (thông qua @Autowired), giúp rút ngắn thời gian lập trình, giảm sự rườm rà.

Spring Boot là một module của Spring Framework giúp các lập trình viên loại bỏ các bước cấu hình phức tạp mà Spring bắt buộc. Spring Boot là dự án phát triển bởi ngôn ngữ Java (JAV) trong hệ sinh thái Spring framework.

Một số tính năng nổi bật của Spring Boot đó là:

* Tạo ra các ứng dụng Spring mang tính độc lập.
* Nhúng trực tiếp Tomcat, Jetty hoặc Undertow mà không cần phải triển khai ra file WAR.
* Starter dependency giúp cho việc chạy cấu hình Maven trở nên đơn giản hơn.
* Tự động chạy cấu hình Spring khi cần thiết.
* Không sinh code cấu hình, đồng thời không yêu cầu phải cấu hình bằng XML.

Spring có kích thước rất nhỏ, trong suốt và nhẹ trong quá trình chạy. Chính vì kích thước nhỏ, nên đây là một giải pháp khá gọn, nhẹ với khả năng hỗ trợ để tạo ra và phát triển các ứng dụng web có mã hiệu suất cao. Hơn hết, có thể dễ dàng kiểm tra, thử nghiệm hoặc sử dụng lại code.

Những tính năng cốt lõi của Spring framework có thể được sử dụng trong việc phát triển java desktop, java web, … Mục tiêu chính là dễ dàng phát triển các ứng dụng J2EE dựa trên mô hình sử dụng POJO

**II/ Caching dữ liệu**

1. **Giới thiệu**

Caching là một kỹ thuật trong lập trình máy tính được sử dụng để lưu trữ tạm thời các dữ liệu, kết quả của các phép toán, hoặc các tài nguyên có thể được truy cập một cách nhanh chóng. Mục tiêu của caching là giảm thời gian truy cập dữ liệu, tăng cường hiệu suất và giảm gánh nặng cho hệ thống.

Khi một ứng dụng yêu cầu dữ liệu, nó sẽ kiểm tra xem dữ liệu đã được lưu trong bộ nhớ cache hay chưa. Nếu dữ liệu đã tồn tại trong cache, ứng dụng sẽ lấy từ đó thay vì phải truy cập vào nơi lữu trữ gốc (ví dụ: cơ sở dữ liệu, API, hoặc tệp tin). Nếu dữ liệu không có trong cache, ứng dụng sẽ lấy từ nơi lưu trữ gốc và sau đó lưu vào cache để sử dụng cho các lần truy cập sau.

1. Các phương pháp caching

* **Spring Cache Abstraction**

Sử dụng các annotations (@Cacheable, @CacheEvict, @CachePut, @Caching) để quản lý cache một cách trừu tượng mà không cần quan tâm đến việc thực hiện cụ thể.

Nó không phải là một trình quản lý cache thực tế, mà là một API trừu tượng giúp bạn dễ dàng tích hợp với các trình quản lý cache khác như EhCache, Caffeine, Hazelcast, Guava, Redis,..

**@Cacheable:** Dùng để đánh dấu một phương thức rằng kết quả của nó có thể được cache. Khi phương thức được gọi, Spring sẽ kiểm tra xem kết quả của phương thức đó đã được cache chưa. Nếu có, kết quả từ cache sẽ được trả về thay vì thực hiện phương thức

**@CacheEvict:** Dùng để loại bỏ dữ liệu khỏi cache. Bạn có thể cấu hình nó để xóa một mục cụ thể khỏi cache hoặc để xóa tất cả các mục trong một cache cụ thể.

**@CachePut:** Luôn thực hiện phương thức và kết quả của nó sẽ được đưa vào cache theo khóa chỉ định. Điều này hữu ích khi bạn muốn cập nhật cache với một kết quả mới.

**@Caching:** Được sử dụng để nhóm nhiều chú thích caching trên một method duy nhất. Nó cho phép bạn áp dụng nhiều hành vi caching khác nhau cho một method, kết hợp hiệu ứng của @Cacheable, @CachePut, và @CacheEvict.

**@CacheConfig:** Dùng để chia sẻ một số thuộc tính cache giữa các method trong một lớp

Spring Cache rất linh hoạt. Không chỉ sử dụng các trình quản lý cache in-memory như EhCache, Caffeine mà còn có thể tích hợp với các hệ thống cache phân tán như Redis, Hazelc

Mặc dù nó được thiết kế để làm việc với bất kỳ trình quản lý cache nào nhưng không yêu cầu phải tích hợp với một trình quản lý cụ thể. Khi không cung cấp một cấu hình cache cụ thể, Spring sẽ tạo ra một SimpleCacheMemory với một ConcurrentMapChe mặc định, là một cách đơn giản để thực hiện caching in-memory mà không cần bất kỳ sự tích hợp đặc biệt nào

ConcurrentMapCache là một cách tiện lợi để bắt đầu với caching trong Spring nếu bạn không có nhu cầu phức tạp và không cần một giải pháp caching mạnh mẽ như Caffeine, Redis, EhCache, ... Đây là một cách tốt để thêm caching vào ứng dụng của bạn nếu bạn chỉ cần cải thiện hiệu suất cho các trường hợp sử dụng đơn giản và không yêu cầu tính năng đầy đủ của một trình quản lý cache chuyên nghiệp.

* In-Memory Cache

In-Memory Cache là một hình thức caching trong đó dữ liệu được lưu trữ trực tiếp trong bộ nhớ chính (RAM) của máy chủ. Giúp việc truy xuất dữ liệu nhanh hơn nhiều so với việc phải đọc từ ổ đĩa cứng hoặc một nguồn dữ liệu từ xa như cơ sở dữ liệu hoặc dịch vụ web.

Tuy nhiên, RAM có kích thước hạn chế và dữ liệu trong cache thường bị mất khi ứng dụng hoặc máy chủ bị tắt, cần phải có chiến lược để quản lý việc đặt vào và loại bỏ khỏi cache (như eviction policies), cũng như để đồng bộ hóa cache trong một hệ thống phân tán.

Công cụ phổ biến để thực hiện In-Memory Cache

* HashMap hoặc ConcurrentHashMap: Sử dụng một bản đồ đơn giản trong bộ nhớ để lưu trữ dữ liệu. Đây là cách đơn giản nhất nhưng không cung cấp các tính năng quản lý cache tự động.
* Caffeine: Một thư viện caching nhanh chóng, mạnh mẽ với các cơ chế quản lý cache như evictions, loading, và computation.
* Guava: Trước khi có Caffeine, Guava là lựa chọn phổ biến cho in-memory cache với các tính năng tương tự nhưng không hiệu suất cao như Caffeine.
* Distributed Cache

Distributed Cache là một hệ thống cache được thiết kế để hoạt động trên nhiều máy chủ hoặc nút trong một mạng. Mục đích chính của nó là để tăng tốc độ truy cập dữ liệu bằng cách lưu trữ bản sao của dữ liệu thường xuyên được truy cập hoặc tính toán tốn kém trên nhiều địa điểm trong mạng.

Các Đặc Điểm Chính:

* Phân Tán và Đồng Bộ: Dữ liệu được phân tán giữa nhiều nút hoặc máy chủ và thường được đồng bộ hóa hoặc sao chép giữa các nút để đảm bảo tính nhất quán của dữ liệu.
* Tăng Tốc Độ và Giảm Độ Trễ: Cung cấp dữ liệu từ cache gần với vị trí của người dùng giúp giảm độ trễ và tăng tốc độ truy cập dữ liệu.
* Tăng Khả Năng Mở Rộng: Hỗ trợ khả năng mở rộng ngang, cho phép hệ thống mở rộng bằng cách thêm nhiều nút cache hơn mà không ảnh hưởng đến hiệu suất.
* Tăng Khả Năng Chịu Lỗi: Một số hệ thống cache phân tán cung cấp khả năng chịu lỗi, nơi mất mát một nút không làm mất mát dữ liệu hoặc làm gián đoạn dịch vụ do dữ liệu có thể được truy cập từ các nút khác.

Các Công Cụ Distributed Cache Phổ Biến:

* Redis: Một cơ sở dữ liệu key-value nhanh chóng, hỗ trợ các cấu hình như cache phân tán, với khả năng lưu trữ dữ liệu trong bộ nhớ và cung cấp các cấu trúc dữ liệu phức tạp.
* Memcached: Một hệ thống caching phân tán nhẹ, dễ dàng cấu hình, tập trung vào việc lưu trữ dữ liệu key-value đơn giản trong bộ nhớ.
* Hazelcast: Một in-memory data grid, cung cấp các cấu trúc dữ liệu phân tán và hỗ trợ transaction, được sử dụng để tạo ra các hệ thống có khả năng chịu lỗi và mở rộng cao.

Về bản chất các công cụ này cũng hoàn toàn có thể được sử dụng cho mục đích In-memory cache tuy nhiên khi nhắc đến việc caching trên các hệ thống phân tán, chúng sẽ là những lựa chọn hàng đầu. Khi mà các công cụ In-memory cache không đáp ứng được hoặc sẽ khó để triển khai trên các ứng dụng hệ thống phân tán.

**Ưu điểm:**

* Tăng hiệu suất: giảm thời gian truy cập dữ liệu từ nơi lưu trữ gốc, giúp ứng dụng hoạt động nhanh chóng hơn, từ đó giúp cho ứng dụng của bạn tăng hiệu suất làm việc.
* Giảm gánh nặng cho cơ sở dữ liệu: giảm số lượng truy vấn trực tiếp đến cơ sở dữ liệu, giảm áp lực lên cơ sở dữ liệu và cải thiện hiệu suất hệ thống. Với những ứng dụng lớn, với hàng trăm, hàng ngàn query vào cơ sở dữ liệu mỗi giây, thì caching là một giải pháp cứu cánh.
* Cải thiện trải nghiệm người dùng: Cung cấp trải nghiệm người dùng mượt mà hơn và giảm độ trễ.
* Cải thiện độ ổn định hệ thống: Giúp cải thiện độ ổn định của hệ thống bằng cách giảm độ trễ và gánh nặng, giúp hệ thống chạy mượt mà hơn.

**Lưu ý khi sử dụng:**

* Rủi ro dữ liệu lỗi thời (Stale data): Nếu dữ liệu trong cache không được cập nhật thường xuyên, có thể dẫn đến sử dụng dữ liệu lỗi thời (stale) => cần phải kiểm tra và cập nhật dữ liệu trong cache thường xuyên để tránh cho nó rơi vào tình trạng này
* Quản lý thời gian sống (Time-to-live) cẩn thận: Thiết lập thời gian sống (time-to-live) sao cho phù hợp với tính chất của dữ liệu và mức độ thay đổi. Nhưng cũng phải cẩn thận và tính toán kỹ lưỡng cho khoảng thời gian này để tránh dữ liệu lỗi thời hoặc re-fresh quá nhiều khiến việc cache trở nên vô nghĩa.
* Quản lý bộ nhớ hiệu quả: Caching có thể làm tăng tiêu tốn bộ nhớ, đặc biệt là khi lưu trữ lớn lượng dữ liệu. Chính vì thế, kiểm soát lượng bộ nhớ sử dụng cho caching để tránh tình trạng chiếm dụng bộ nhớ quá mức.
* Xác định rõ mục tiêu và áp dụng chiến lược hợp lý: Đặt rõ mục tiêu bạn muốn đạt được với caching, liệu bạn đang tập trung vào tăng tốc độ, giảm áp lực cho cơ sở dữ liệu hay cải thiện trải nghiệm người dùng từ đó lựa chọn chiến lược caching phù hợp với yêu cầu và tính chất của ứng dụng.