Triển khai mô hình Microservices với framework Spring

1. Microservices là gì:

1.1 Kiến trúc 1 khối (Monolithic):

- Là kiến trúc phổ biến nhất hiện nay, các module sẽ được đóng gói và cài đặt lại thành 1 khối.

Chẳng hạn 1 chương trình java sẽ đóng gói source thành file war để chạy trên tomcat, jetty, glassfish...nếu ứng dụng là web, hoặc file jar thuần nếu chạy là app thường.

-> Ưu điểm: Dễ viết, dễ triển khai, dễ test.

-> Nhược điểm: Cho dù thiết kế degsin tốt đến mấy, khi ứng dụng phình to sẽ khó quản lý vì số lượng dòng code tăng lên.

1.2 Kiến trúc "nhiều" khối (microservices):

- Để giải quyết vấn đề này thì ý tưởng của microservices là chia nhỏ thành các service, mỗi service làm 1 nhiệm vụ riêng biệt.
- Khi có request sẽ gọi tới từng service phù hợp để lấy về kết quả cần thiết.

Triển khai mô hình Microservices với framework Spring

1. Microservices là gì:

1.2 Kiến trúc "nhiều" khối (microservices):

-> Đặc điểm:

- Kiến trúc microservices có những đặc điểm sau:

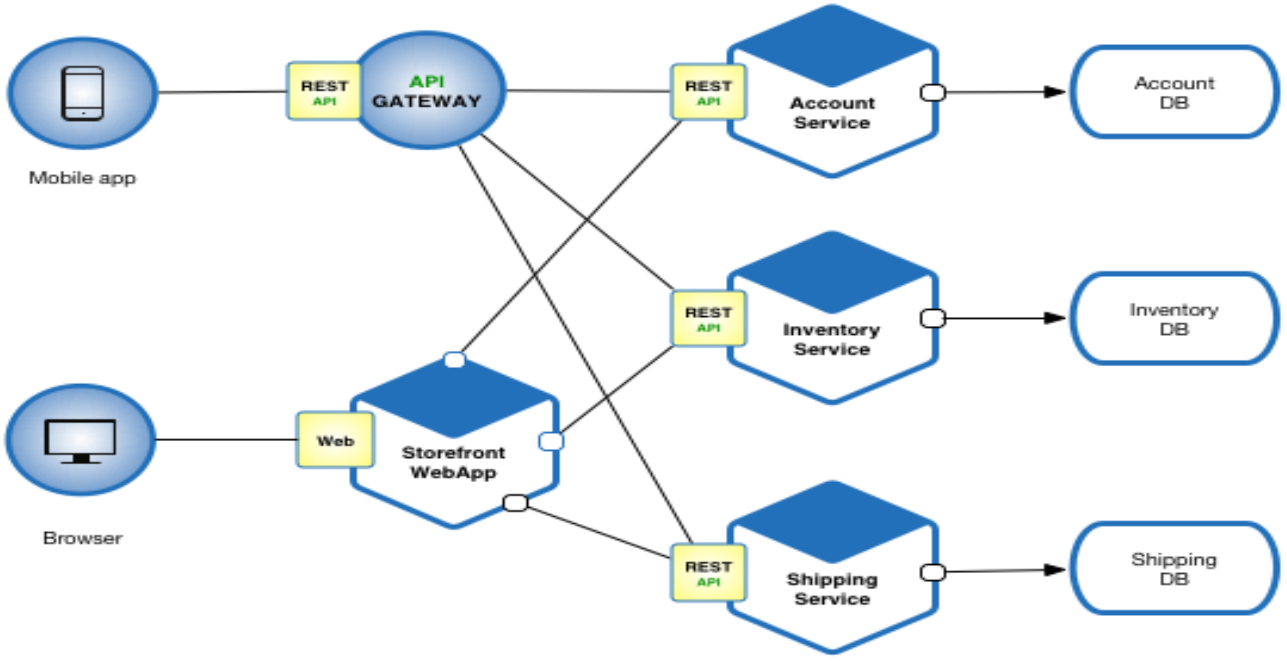
1. **Decoupling**: Các service được tách rời nhau.
2. **Componentization**: Các service độc lập nhau.
3. **Business Capabilities:** Mỗi thành phần trong service tập trung vào 1 logic duy nhất.
4. **Autonomy**: Cho phép các lập trình viên làm việc độc lập với nhau (mỗi team code 1 service).
5. **Continous Delivery**: Cập nhật thay đổi từng service mà không sợ bị ảnh hưởng tới các service khác.
6. **Decentralized Governance**: Có nhiều sự lựa chọn về công nghệ framework để phát triển.
7. **Agility**: Hỗ trợ phát triển theo mô hình Agile.

Triển khai mô hình Microservices với framework Spring

1. Microservices là gì:

1.2 Kiến trúc "nhiều" khối (microservices):

-> Sơ đồ:



Triển khai mô hình Microservices với framework Spring

1. Microservices là gì:

1.2 Kiến trúc "nhiều" khối (microservices):

-> Giao tiếp giữa các service:

- Trước đây, người ta thường dùng REST để giao tiếp. Ưu điểm của REST là dễ triển khai. Tuy nhiên, việc dùng REST có 1 bất lợi khá lớn về performance. Đó là REST dùng giao thức http. Đối với 1 hệ thống lớn các chức năng phải lấy từ nhiều service, việc truyền và nhận dữ liệu thông qua http là khá chậm. Vì vậy, dùng REST để giao tiếp giữa các service là không tối ưu.

- Vài năm gần đây, có những tiến bộ về việc giao tiếp giữa các hệ thống phân tán bằng việc phát triển giao thức gọi hàm từ xa: *Remote Procedure Calls* - RPC. Giao thức RPC truyền thông tin dưới dạng các gói tin binary, vì vậy tốc độ là cực nhanh gấp n lần so với REST. (n bằng bao nhiêu phụ thuộc vào nhiều yếu tố khác như server, đường truyền, băng thông...).

- Nhận thấy tiềm năng đó, google phát triển thành lib gRPC cái mà theo như hãng "marketing" nó nhanh gấp 10 lần REST hiện tại !. Tuy nhiên, con số x10 chưa chắc chắn được, thực tế ghi nhận gRPC nhanh gấp 4-5 lần REST.

Triển khai mô hình Microservices với framework Spring

1. Microservices là gì:

1.2 Kiến trúc "nhiều" khối (microservices):

-> Giao tiếp giữa các service:

- Chúng ta đã quen thuộc với 2 dạng dữ liệu thân thiện với lập trình viên (không trong suốt) đó là JSON, XML. Còn 1 loại thứ 3 nữa nhanh hơn 2 loại trên từ 3-10 lần (là lời giải thích cho tốc độ vượt trội của GRPC so với REST) đó là protobuf.

- Được phát triển bởi google được sử dụng để serialize dữ liệu thành dạng byte stream (các gói tin).

- Không trong suốt với lập trình viên (không thể nhìn thấy vì là binary).

Triển khai mô hình Microservices với framework Spring

1. Microservices là gì:

1.2 Kiến trúc "nhiều" khối (microservices):

-> Giao tiếp giữa các service:

- Để tạo ra protobuf, chúng ta tạo các file .proto định nghĩa, sau đó gen ra thành code (Chi tiết sẽ nói ở phần demo source). Ví dụ 1 file proto đơn giản định nghĩa class Department:

```
syntax = "proto3";

import "google/protobuf/empty.proto";
import "google/protobuf/timestamp.proto";
option java_package = "com.dimageshare.core.autogen.grpc.department";
option java_multiple_files = true;

package department;
// ================


message Department {
    int32 dpId = 1;
    string dpName = 2;
    string dpDescription = 3;
}

message DepartmentSaving {
    string dpName = 1;
    string dpDescription = 2;
}

message DepartmentIdRequest {
    int32 dpId = 1;
}

message DepartmentResponses {
    repeated Department department = 1;
}

service DepartmentService {
    rpc findDepartments (google.protobuf.Empty) returns (DepartmentResponses);
    rpc findDepartmentById (DepartmentIdRequest) returns (Department);
    rpc saveDepartment (DepartmentSaving) returns (google.protobuf.Empty);
    rpc removeDepartmentById (DepartmentIdRequest) returns (google.protobuf.Empty);
}
```

Triển khai mô hình Microservices với framework Spring

1. Microservices là gì:

1.2 Kiến trúc "nhiều" khối (microservices):

-> Nguyên tắc thiết kế:

- Về lý thuyết các service có thể giao tiếp với nhau thoải mái, tuy nhiên nếu để chúng giao tiếp tự do sẽ xảy ra rất nhiều vấn đề. Chẳng hạn: khi một service bị lỗi, kéo theo rất nhiều service khác lỗi theo, nếu các service đó giao tiếp với service này. Việc debug cũng rất khó khăn.

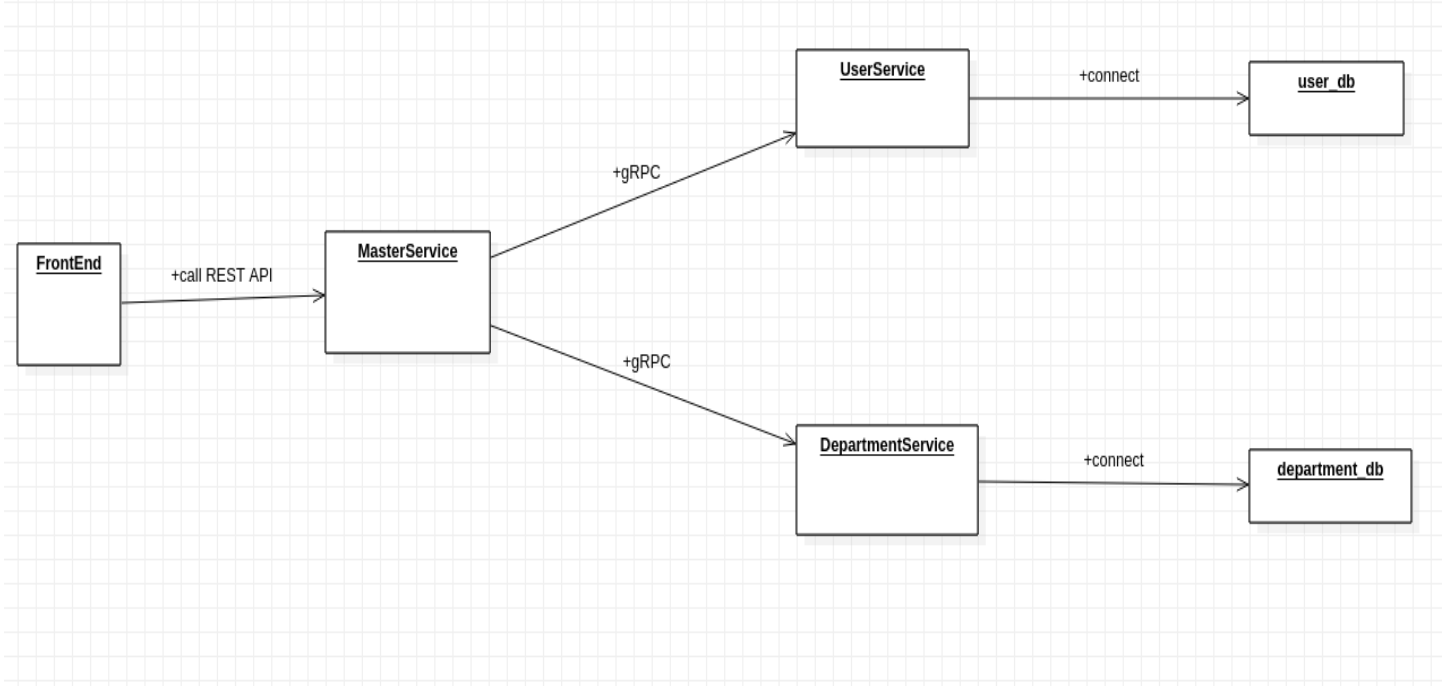
- Do đó để khắc phục vấn đề này, chúng ta sẽ thiết kế theo mô hình node-master. Một service master sẽ chịu trách nhiệm gọi tới các service node để lấy thông tin, sau đó tổng hợp và thực hiện. (Public ra ngoài thành API chẳng hạn). Các service node sẽ không giao tiếp với nhau mà chỉ có thể giao tiếp với master service mà thôi.

Triển khai mô hình Microservices với framework Spring

1. Microservices là gì:

1.2 Kiến trúc "nhiều" khối (microservices):

-> Sơ đồ 1 hệ thống microservice đơn giản:



Triển khai mô hình Microservices với framework Spring

2. Giới thiệu framework Spring:

2.1 Nguyên lý SOLID:

- Là những nguyên lý thiết kế trong OOP, được đúc kết bởi rất nhiều developer, trải qua hàng ngàn dự án thành công/thất bại. Một project áp dụng những nguyên lý này sẽ có code dễ đọc, dễ test, rõ ràng hơn. Và việc quan trọng nhất là việc maintain code sẽ dễ hơn rất nhiều (Ai có kinh nghiệm trong ngành IT đều biết thời gian code chỉ chiếm 20-40%, còn lại là thời gian để maintainance: thêm bớt chức năng và sửa lỗi).

- SOLID là viết tắt các chữ cái của 5 nguyên tắc sau:

1. Single responsibility principle (Một class chỉ nên giữ 1 trách nhiệm duy nhất)
2. Open/closed principle (Có thể thoải mái mở rộng 1 class, nhưng không được sửa đổi bên trong class đó)
3. Liskov Substitution Principle (Trong một chương trình, các object của class con có thể thay thế class cha mà không làm thay đổi tính đúng đắn của chương trình).
4. Interface Segregation Principle (Thay vì dùng 1 interface lớn, ta nên tách thành nhiều interface nhỏ, với nhiều mục đích cụ thể)
5. Dependency inversion principle :

1. Các module cấp cao không nên phụ thuộc vào các modules cấp thấp. Cả 2 nên phụ thuộc vào abstraction

2. Interface (abstraction) không nên phụ thuộc vào chi tiết, mà ngược lại.(Các class giao tiếp với nhau thông qua interface, không phải thông qua implementation.)

Triển khai mô hình Microservices với framework Spring

2. Giới thiệu framework Spring:

2.2 Sự ra đời của framework Spring:

- Tại sao lại là SOLID ? Bởi vì 1 trong số các nguyên lý thiết kế spring nằm trong SOLID.

- Spring ra đời năm 2002 phát hành phiên bản đầu tiên bởi Rod Johnson.

- Spring ra đời và phát triển cũng phần lớn là nhờ người tiền nhiệm của nó: nền tảng EJB (Enterprise Java Bean).

- EJB là mô hình có nhiệm vụ chính là xử lý các nghiệp vụ logic và truy suất dữ liệu, có thể hiểu EJB là Model trong mô hình MVC.

- Tuy nhiên EJB ngày càng nặng và phức tạp, và spring khắc phục được rất nhiều nhược điểm của EJB, do đó ngày càng phát triển, trở thành một trong những framework java nổi tiếng nhất hiện nay.

Triển khai mô hình Microservices với framework Spring

2. Giới thiệu framework Spring:

2.3 Nguyên lý xây dựng framework Spring:

- 2 nguyên lý: Dependency Injection và Aspect Oriented Programming.

1. Dependency Injection:

- Là một khái niệm không mới, nhưng lại vô cùng quan trọng, không chỉ trong Spring mà còn trong rất nhiều các Framework khác.

- Chính là một "cách thức thực hiện" nguyên lý thứ 5 của SOLID (Dependency inversion principle).

- Dịch nghĩa đen: *tiêm nhiễm sự phụ thuộc* - khá tối nghĩa. Vậy cái gì phụ thuộc cái gì, cái gì tiêm cái gì ?

Triển khai mô hình Microservices với framework Spring

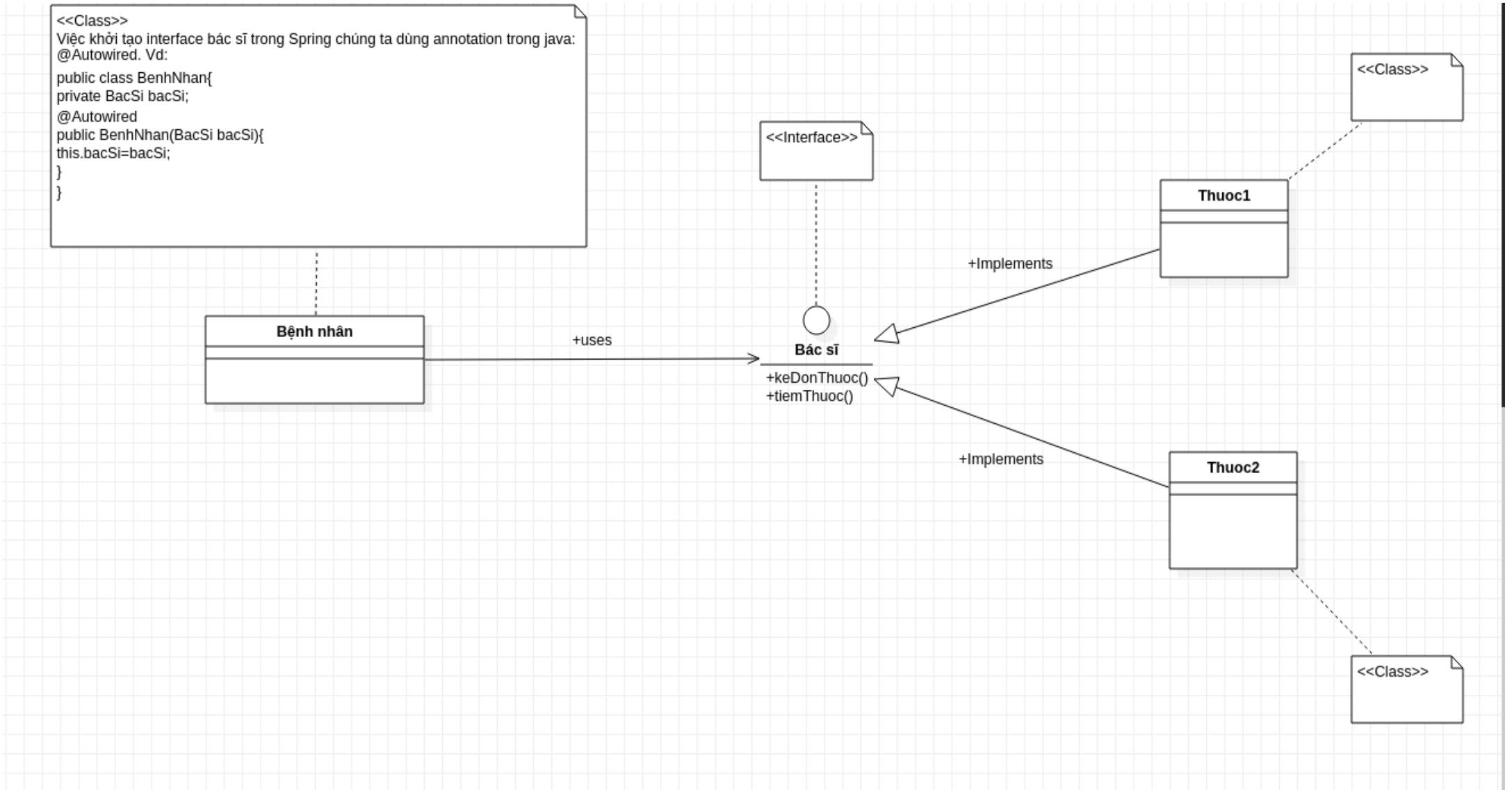
2. Giới thiệu framework Spring:

2.3 Nguyên lý xây dựng framework Spring:

- 2 nguyên lý: Dependency Injection và Aspect Oriented Programming.

1. Dependency Injection:

- Lấy 1 ví dụ thực tế: Một bệnh nhân đến gặp bác sĩ, điều anh ta muốn là khỏi bệnh, chứ không cần quan tâm đến việc những loại thuốc nào được kê đơn, cho uống, tiêm... vào người anh ta. Soi sang nguyên lý thứ 5 của SOLID, class phụ thuộc ở đây là bệnh nhân, class cấp cao là bác sĩ. Việc bác sĩ kê đơn, cho uống hay tiêm ... gì vào người bệnh nhân chính là injection.


<<Class>>
Việc khởi tạo interface bác sĩ trong Spring chúng ta dùng annotation trong java:
@Autowired. Vd:
public class BenhNhan{
private BacSi bacSi;
@Autowired
public BenhNhan(BacSi bacSi){
this.bacSi=bacSi;
}
}
<<Interface>>
<<Class>>
Thuoc1
+Implements
Bệnh nhân
+uses
Bác sĩ
+keDonThuoc()
+tiemThuoc()
+Implements
Thuoc2
<<Class>>

Triển khai mô hình Microservices với framework Spring

2. Giới thiệu framework Spring:

2.3 Nguyên lý xây dựng framework Spring:

1. Dependency Injection:

   - Có vài cách để tạo Dependency Injection:

- **Constructor Injection**: Các dependency sẽ được container truyền vào (inject vào) 1 class thông qua constructor của class đó. Đây là cách thông dụng nhất.
- **Setter Injection**: Các dependency sẽ được truyền vào 1 class thông qua các hàm Setter.
- **Fields/ properties**: Các dependency sẽ được truyền vào 1 class một cách trực tiếp vào các field, file config.
- **Interface Injection**: Class cần inject sẽ implement 1 interface. Interface này chứa 1 hàm tên Inject. Container sẽ injection dependency vào 1 class thông qua việc gọi hàm Inject của interface đó. (Trong con **batch farm8** anh Thọ đã dùng cách này ở task domain async location thông qua interface "repository" khởi tạo các interface (LocationRepository, ArticleRepository...) khác :D).

Triển khai mô hình Microservices với framework Spring

2. Giới thiệu framework Spring:

2.3 Nguyên lý xây dựng framework Spring:

2. Aspect Oriented Programming (Lập trình hướng khía cạnh):

- AOP là một kỹ thuật lập trình cho phép phân tách chương trình thành cách module riêng rẽ, không phụ thuộc nhau. Khi hoạt động, chương trình sẽ kết hợp các module lại để thực hiện các chức năng nhưng khi sửa đổi chức năng thì chỉ cần sửa đổi trên một module cụ thể.

- AOP phát triển dựa trên OOP chứ không phải ra đời nhằm thay thế OOP, đó là phần bổ sung cho OOP khi mà có những việc với cách làm việc theo OOP không giải quyết được vấn đề.

- Trong AOP có khái niệm “lát cắt”, “điểm cắt”, tạm hiểu là module A sẽ xen vào module B để thực hiện 1 chức năng nào đó mà không làm ảnh hưởng module B, “điểm cắt” là vị trí mà module A xen vào module B.

Triển khai mô hình Microservices với framework Spring

2. Giới thiệu framework Spring:

2.3 Nguyên lý xây dựng framework Spring:

2. Aspect Oriented Programming (Lập trình hướng khía cạnh):

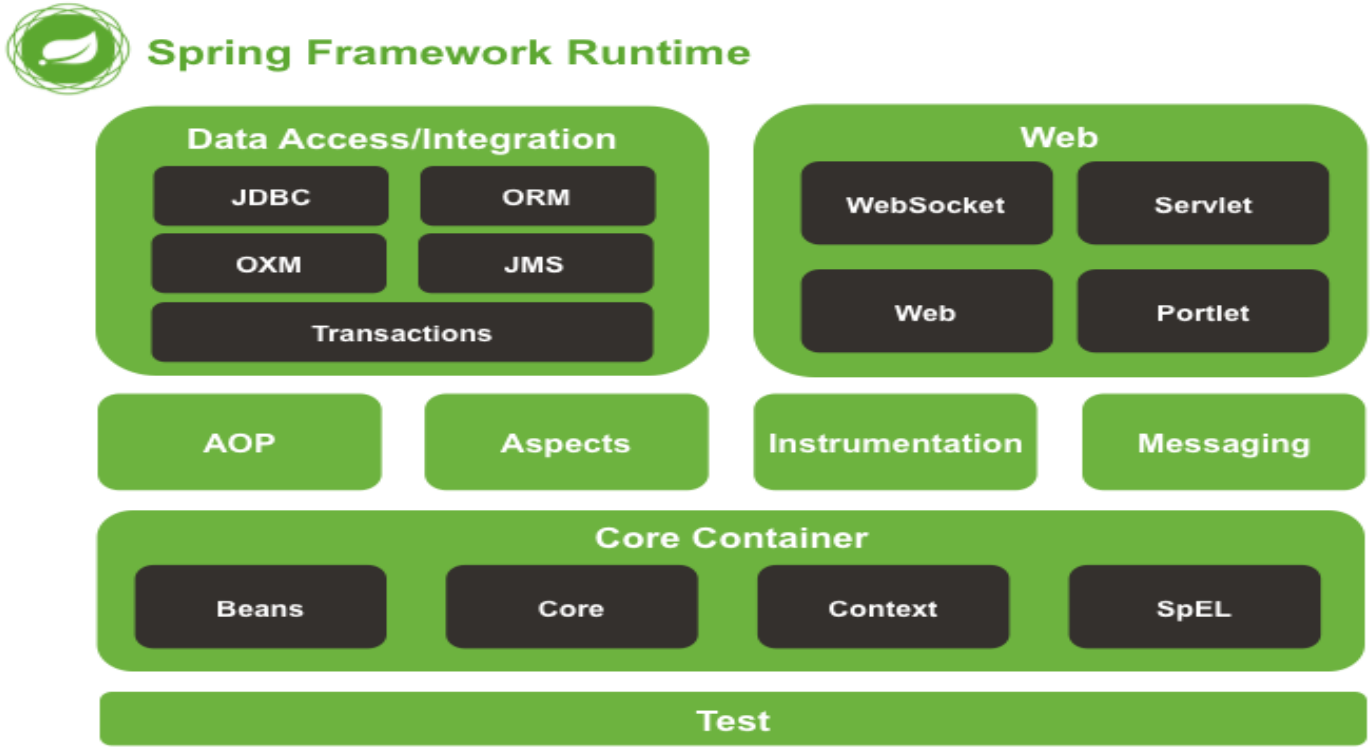
- Ví dụ thực tế: Để tới được văn phòng công ty làm việc, chúng ta phải đi qua 1 chốt chặn trước tòa nhà, đó là bác bảo vệ. Bác cho phép chúng ta gửi xe rồi lên văn phòng thì chúng ta mới được lên, còn không sẽ bị giữ lại. Giả sử bác bảo vệ đưa ra luật là sau 9h sẽ không cho bạn dựng xe trong hầm gửi xe nữa. Hiển nhiên, việc sửa lại luật này chỉ có bác bảo vệ thực hiện, về phía bạn không có bất kỳ sự thay đổi nào. Và khi bác sửa lại luật, bạn phải tuân theo.

- Trong Spring, có 1 khái niệm là interceptor, nguyên lý hoạt động cũng na ná servlet filter trong web vậy. Mỗi request muốn tới controler đều phải đi qua interceptor. Chúng ta sẽ code các rule cho interceptor, để chặn những request nào không thỏa mãn. **Muốn thay đổi rule nào, chúng ta chỉ cần sửa ở interceptor, không phải sửa ở từng luồng request**. Đây chính là một hình thức sử dụng AOP.

Triển khai mô hình Microservices với framework Spring

2. Giới thiệu framework Spring:

2.4 Các thành phần cấu thành nên framework Spring:



Triển khai mô hình Microservices với framework Spring

2. Giới thiệu framework Spring:

2.4 Các thành phần cấu thành nên framework Spring:

1. Spring Core là thành phần cốt lõi của Spring Framework. Đây chính là nền tảng để xây dựng nên các thành phần khác trong hệ sinh thái của Spring Framework.
2. Spring Bean có thể được hiểu là các đối tượng Java đơn giản. Là trung tâm của Spring Core.
3. Spring Context kết nối mọi thứ lại với nhau.
4. Spring Expression Language là một ngôn ngữ ngắn gọn giúp cho việc cấu hình Spring Framework trở nên linh hoạt hơn. Ví dụ cách lấy data trong Thymeleaf (một template trong java thao tác, tạo HTML, XML, Javascript, CSS và text.). Soi sang "manager" của PHP, Thymeleaf giống file tpl. Còn cách lấy giá trị từ form giống như Spring Expression Language.

Triển khai mô hình Microservices với framework Spring

2. Giới thiệu framework Spring:

2.5 Các dự án trong hệ sinh thái Spring:

1. **Spring MVC**

Spring MVC được thiết kế dành cho việc xây dựng các ứng dụng nền tảng web.

2. **Spring Data**

Cung cấp một cách tiếp cận đúng đắn để truy cập dữ liệu từ cơ sở dữ SQL, NoSQL, caching...

3. **Spring Security**

Dự án này cung cấp các cơ chế xác thực (authentication) và phân quyền (authorization) cho ứng dụng.

4. **Spring Boot**

Spring Boot là một framework giúp chúng ta phát triển cũng như chạy ứng dụng một cách nhanh chóng.

Triển khai mô hình Microservices với framework Spring

2. Giới thiệu framework Spring:

2.5 Các dự án trong hệ sinh thái Spring:

5. **Spring Batch**

Dự án này giúp chúng ta dễ dàng tạo các lịch trình (scheduling) và tiến trình (processing) cho các công việc xử lý theo mẻ (batch job).

6. **Spring Integration**

Spring Integration là một implementation của Enterprise Integration Patterns (EIP). Dự án này thiết kế một kiến trúc hướng thông điệp hỗ trợ việc tích hợp các hệ thống bên ngoài.

7. **Spring XD**

Mục tiêu của dự án này là đơn giản hóa việc phát triển các ứng dụng BigData.

8. **Spring Social**

Để kếtnối ứng dụng của bạn với các API bên thứ ba của Facebook, Twitter, Linkedin … Một bài toán quen thuộc đó là Account Linking, đăng nhập với google, facebook, github...

Triển khai mô hình Microservices với framework Spring

2. Giới thiệu framework Spring:

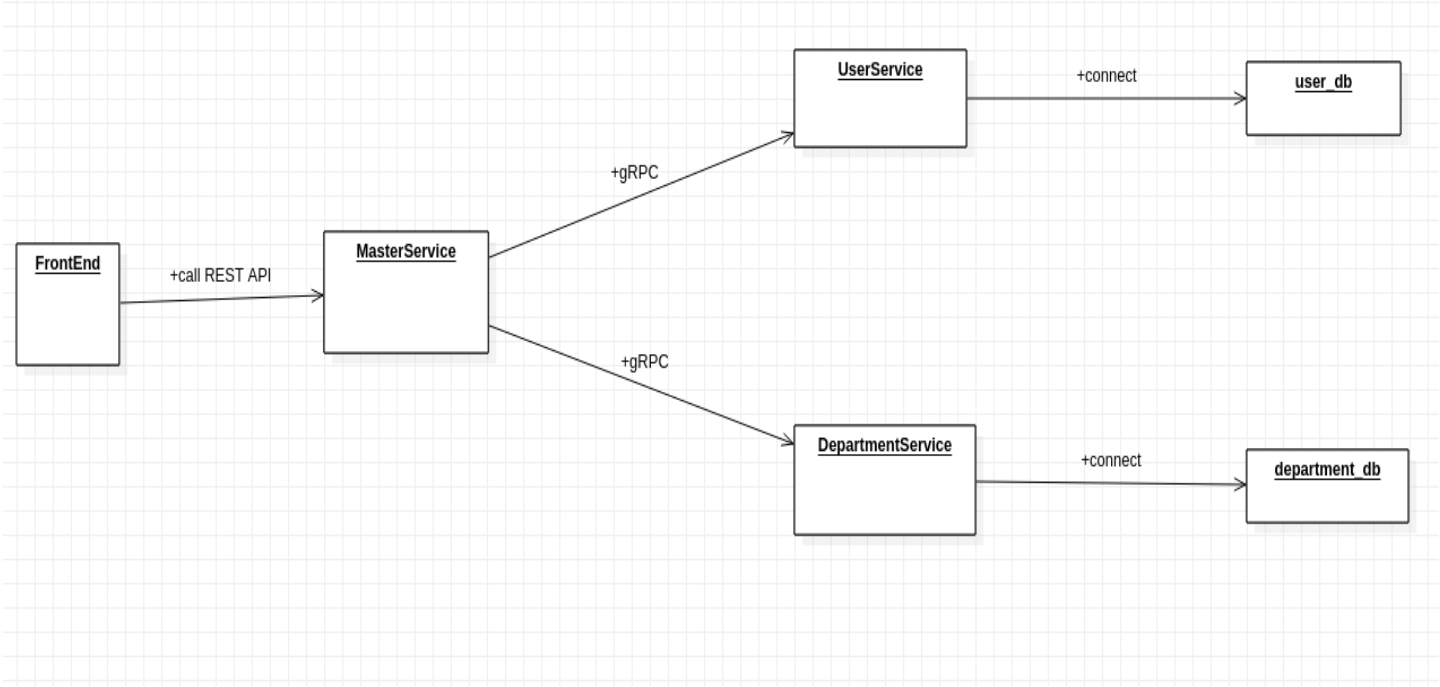
2.6 Đôi nét về Spring Boot:

- Trước đây, việc triển khai một ứng dụng web thường dùng Spring MVC. Tuy nhiên, việc config các file xml...khá cồng kềnh, cộng thêm việc phải có 1 máy chủ (tomcat, glassfish, netty...) để deploy file war trước khá mất thời gian để xây dựng. Vậy thay vì phải deploy file war trên 1 máy chủ, tại sao chúng ta không "gói" máy chủ đó vào ứng dụng để chạy. Đó chính là ý tưởng của Spring Boot. Với Spring Boot chúng ta sẽ chạy các file jar bình thường, và máy chủ sẽ được đóng gói trong đó. Việc triển khai vô cùng nhanh chóng và đơn giản.

Triển khai mô hình Microservices với framework Spring

3. Demo xây dựng hệ thống Microservice với Spring:

3.1 Nhắc lại sơ đồ ứng dụng demo:



Triển khai mô hình Microservices với framework Spring

3. Demo xây dựng hệ thống Microservice với Spring:

3.2 Source code demo:

https://github.com/Silencer05051993/microservices-springboot.git

Tham khảo ứng dụng API backend với Spring Boot:

https://github.com/Silencer05051993/entropydb-backend.git

Triển khai mô hình Microservices với framework Spring

3. Demo xây dựng hệ thống Microservice với Spring:

3.2 Triển khai hệ thống microservice trong thực tế:

- Build source base đã khó khăn (đối với người chưa có kinh nghiệm), việc deploy lại càng khó hơn. Nếu chỉ đơn giản một vài service, chúng ta hoàn toàn có thể build thủ công bằng cơm !. Thế nhưng, một ứng dụng microservice thường có rất nhiều service. Khi đó việc deploy bằng cơm là không ổn. Hơn nữa, do các team dev các service khác nhau, các version thay đổi liên tục, chúng ta cần có những hệ thống CI/CD để tự động hóa việc deploy cũng như quản lý.

- Một hệ thống microservice tốt nên có những công nghệ sau đi kèm:

1. Git-chắc chắn rồi, để quản lý version.

2. Jenkins, đây là tool deploy mã tự động (thay cho cơm), trỏ tới repository của git trên server, chỉ định 1 branch chính vd : developer cho jenkins thì mỗi khi nhánh này có sự thay đổi, Jenkins sẽ deploy lại source. Trong quá trình build, version cũ vẫn được giữ nguyên và chạy. Nếu quá trình deploy bị lỗi, version cũ tiếp tục chạy. Nếu thành công, version mới được update thay cho version cũ.

Triển khai mô hình Microservices với framework Spring

3. Demo xây dựng hệ thống Microservice với Spring:

3.2 Triển khai hệ thống microservice trong thực tế:

3. Docker đóng gói chương trình để chạy các service trên các server khác nhau.

4. Khi số lượng docker tăng lên, chúng ta cũng cần 1 hệ thống để quản lý chúng. Một công nghệ đang nổi vài năm gần đây là Kubernetes.

=> Để master những công nghệ này cần rất nhiều thời gian, cũng như nhân lực. Do đó, microservice chỉ thực sự phù hợp với những hệ thống lớn cũng như những startup chịu đầu tư vào công nghệ, nhỡ sau này may mắn to ra thì scale cho dễ :D.

=> Với 4 công nghệ này, chúng ta sẽ triển khai những hệ thống microservice lớn như sơ đồ sau đây:

Triển khai mô hình Microservices với framework Spring

3. Demo xây dựng hệ thống Microservice với Spring:

3.2 Triển khai hệ thống microservice trong thực tế: