

BỘ CÔNG THƯƠNG

**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP TP HCM**

**KHOA CÔNG NGHỆ THÔNG TIN**

🙠🙟🕮🙝🙢

**ĐỒ ÁN TỐT NGHIỆP**

**ĐỀ TÀI 12 :QUẢN LÝ KHÁCH SẠN**

**Chuyên ngành : Kĩ thuật phần mềm**

**Giảng viên hướng dẫn : Gv PHẠM THANH HÙNG**

**Sinh viên thực hiện: BÙI MINH PHỤNG**

**Lớp : DHKTPM12B**

**MSSV : 16066001**

**Sinh viên thực hiện: TRẦN HỒNG LÊ**

**Lớp : DHKTPM12A**

**MSSV : 16033631**

**NIÊN KHÓA: 2016 – 2020**

# LỜI MỞ ĐẦU

# NHẬN XÉT

**(Của giảng viên hướng dẫn)**

# NHẬN XÉT

**(Của giảng viên phản biện)**

# MỤC LỤC

[LỜI MỞ ĐẦU 2](#_Toc36300291)

[NHẬN XÉT 3](#_Toc36300292)

[NHẬN XÉT 4](#_Toc36300293)

[DANH MỤC CÁC BẢNG, SƠ ĐỒ, HÌNH 7](#_Toc36300294)

[KÍ HIỆU CÁC CỤM TỪ VIẾT TẮT NẾU CÓ 8](#_Toc36300295)

[CHƯƠNG 1. TỔNG QUAN 9](#_Toc36300296)

[1.1 Lý do chọn đề tài 9](#_Toc36300297)

[1.2 Đặt vấn đề và hướng giải quyết 9](#_Toc36300298)

[1.3 Phạm vi đề tài 10](#_Toc36300299)

[1.4 Phương pháp nghiên cứu 11](#_Toc36300300)

[1.5 Mô tả yêu cầu chức năng 11](#_Toc36300301)

[CHƯƠNG 2. CƠ SỞ LÝ THUYẾT 15](#_Toc36300302)

[2.1 Ngôn ngữ lập trình java 15](#_Toc36300303)

[2.1.1 Java và lịch sử phát triển 15](#_Toc36300304)

[2.1.2 Java Reflection 17](#_Toc36300305)

[2.2 Spring framework 21](#_Toc36300306)

[2.2.1 Tổng quan về Spring 21](#_Toc36300307)

[2.2.2 Lịch sử phát triển 22](#_Toc36300308)

[2.2.3 Một số khái niệm chính 23](#_Toc36300309)

[2.2.4 Các module 27](#_Toc36300310)

[2.2.5 Spring Core 29](#_Toc36300311)

[2.3 Angularjs 44](#_Toc36300312)

[2.3.1 Tổng quan về AngularJS 44](#_Toc36300313)

[2.3.2 Các đặc trưng của AngularJS 45](#_Toc36300314)

[2.3.2.1 Kiến trúc MVC trong AngularJS 45](#_Toc36300315)

[*2.3.2.2 Two-way binding Two-way binding cùng với Directive* là hai trong những đặc trưng nổi bật mang lại sức mạnh cho AngularJS. 46](#_Toc36300316)

[2.3.2.3 Directives 47](#_Toc36300317)

[2.3.2.4 Templates 48](#_Toc36300318)

[2.3.2.5 Dependency Injection 49](#_Toc36300319)

[2.3.2.6 Services 49](#_Toc36300320)

[2.3.2.7 Expression 50](#_Toc36300321)

[2.3.2.8 Fillters 50](#_Toc36300322)

[2.3.2.9 Scope 50](#_Toc36300323)

[2.4 Oracle 51](#_Toc36300324)

[2.4.1 Lịch sử ra đời 51](#_Toc36300325)

[2.4.2 Kiến trúc cơ bản của Oracle 51](#_Toc36300326)

[2.4.2.1 Lớp dữ liệu (File systems) 52](#_Toc36300327)

[2.4.2.2 Lớp xử lý bên dưới (Background processes) 52](#_Toc36300328)

[2.4.2.3 Lớp bộ nhớ (Memory) 52](#_Toc36300329)

[2.4.3 Giới thiệu Oracle Server 53](#_Toc36300330)

[2.4.4.1 Oracle Instance 54](#_Toc36300331)

[2.4.4.2 Oracle Database 55](#_Toc36300332)

[2.4.4 Cấu trúc vùng nhớ – Memory Structure 55](#_Toc36300333)

[2.4.5 System Global Area 55](#_Toc36300334)

[2.4.6 Tổ chức dữ liệu trên CSDL Oracle 56](#_Toc36300335)

[2.4.6.1 Thành phần vật lý 57](#_Toc36300336)

[2.4.7 Data File 57](#_Toc36300337)

[2.4.8 Redo Log File 58](#_Toc36300338)

[2.4.9 Control File 58](#_Toc36300339)

[2.4.10 Thành phần Logic 59](#_Toc36300340)

[2.4.10.1 Tablespace 59](#_Toc36300341)

[2.4.10.2 Data Block 60](#_Toc36300342)

[2.4.10.3 Extent 60](#_Toc36300343)

[2.4.10.4 Segment 60](#_Toc36300344)

[CHƯƠNG 3. PHÂN TÍCH VÀ THIẾT KẾ 62](#_Toc36300345)

[3.1 Phân tích yêu cầu bằng UML 62](#_Toc36300346)

[3.1.1 Usecase tổng quát 63](#_Toc36300347)

[3.1.2 Danh sách tác nhân và mô tả 65](#_Toc36300348)

[3.1.3 Danh sách các tình huống hoạt động (Use cases) 65](#_Toc36300349)

[3.1.4 Tình huống hoạt động 65](#_Toc36300350)

[3.1.4.1 Đặc tả use case UC001. 65](#_Toc36300351)

[3.1.4.2 Activity diagram UC001. 65](#_Toc36300352)

[3.1.4.3 Sequence diagram cho từng tình huống UC001. 65](#_Toc36300353)

[3.2 Class Diagram 66](#_Toc36300354)

[3.3 Mô hình triển khai hệ thống. 66](#_Toc36300355)

[PHỤ LỤC 67](#_Toc36300356)

[DANH MỤC TÀI LIỆU THAM KHẢO 68](#_Toc36300357)

# DANH MỤC CÁC BẢNG, SƠ ĐỒ, HÌNH

# KÍ HIỆU CÁC CỤM TỪ VIẾT TẮT NẾU CÓ

**CNTT: (size 13) Công nghệ Thông tin**

**MMT : Mạng máy tính**

# CHƯƠNG 1. TỔNG QUAN

## 1.1 Lý do chọn đề tài

## 1.2 Phạm vi đề tài

Tài liệu này sẽ tập trung trình bày kết quả nghiên cứu của chúng tôi về các nội dung sau: Spring framework, Java Message System, AngularJS framework, Oracle, Boostrap và Maven. Mỗi phần chúng tôi sẽ giới thiệu sơ lược và trình bày những nội dung cơ bản nhất, những điểm mạnh hay lợi ích mà nó mang lại cho các nhà phát triển phần mềm.

Sau khi tìm hiểu chúng tôi sẽ vận dụng kết quả tìm hiểu được vào việc xây dựng một ứng dụng nhằm mục đích minh họa cho phần lý thuyết đã trình bày. Chúng tôi sẽ xây dựng một ứng dụng quản lý khách sạn nền web.

Ứng dụng quản lý khách sạn được xây dựng có thể áp dụng vào thực tế cho bất kỳ khách sạn nào, không giới hạn các phòng, có thể thanh toán trực tuyến thông qua ví điện tử VNPay khi đặt online thông qua website.

## 1.3 Phương pháp nghiên cứu

Tìm kiếm và nghiên cứu các tài liệu về Spring Framework và các công nghệ có liên quan đến việc phát triển một ứng dụng web enterprise như JMS, AngularJS, Oracle, Maven của các tác giả trong và ngoài nước, các bài báo, thông tin trên mạng,… sau đó chọn lọc và sắp xếp lại theo ý tưởng của mình.

Dựa trên kết quả tìm hiểu được để xây dựng một ứng dụng web quản khách sạn có áp dụng tất cả những nội dung đã nghiên cứu nhằm mục đích minh họa cho phần cơ sở lý thuyết sẽ trình bày trong nội dung tài liệu này.

## 1.4 Mô tả yêu cầu chức năng

Khách sạn PLHotel là một trong những khách sạn lớn tọa lạc tại trung tâm thành phố Lâm Đồng. Phương thức kinh doanh chủ yếu của khách sạn là cho thuê phòng cho khách vãng lai, khách công ty, khách đi tour từ các công ty du lịch. Bên cạnh đó, khách sạn còn cung cấp các dịch vụ khác như ăn uống, spa, bar,dịch vụ giặt ủi, hồ bơi, phòng tập thể dục giúp khách có một kì nghỉ thoải mái bên gia đình và vẫn giữ được thói quen hàng ngày.

Khi khách hàng có nhu cầu đặt phòng khách sạn thì khách hàng có thể liên hệ với khách sạn thông qua số điện thoại hoặc có thể đến trực tiếp quầy để tiến hành làm thủ tục thuê phòng. Nhân viên trao đổi với khách hàng về thời gian, nhu cầu sử dụng phòng để tim phòng phù hợp trên hệ thống và khách hàng cần cung cấp cho nhân viên hỗ trợ số SDT đăng ký, địa chỉ email đang sử dụng, tên người thuê để làm thủ tục đặt phòng.

Ngoài ra, để đảm bảo tính tiện lợi cho khách hàng, khách sạn còn hổ trợ một website đặt phòng trực tuyến giúp khách hàng có thể tự lựa chọn phòng theo thời gian và đặt phòng tại nhà.

Theo quy định tại khách sạn, khách hàng khi đặt phòng phải đặt cọc tối thiểu 30% hóa đơn cho việc đặt phòng hoặc có thể thanh toàn toàn bộ 100% trên tổng số tiền phòng, các phụ phí sẽ được bổ sung khi trả phòng. Nếu quý khách hủy phòng bên khách sạn không có chính sách hoàn tiền cho các phòng đã đặt.

Hàng ngày hệ thống kiểm tra và thông báo đến nhân viên các khách hàng đến ngày checkin. Khách hàng đến tại quầy của khách sạn để làm thủ tục trực tiếp đồng thời cung cấp CMND, SĐT đã đặt phòng và các thông tin cần thiết khác theo yêu cầu của nhân viên để hoàn tất thủ tục nhận phòng. Nhân viên nhận CMND người đặt phòng, hoàn tất thủ tục.

Khi quá hạn checkin theo lịch đặt phòng của khách hàng, hệ thống tự động hủy thông tin đặt phòng

Khách hàng có thể đặc nhiều phòng cùng một lúc. Nếu khách hàng đặt nhiều phòng nhưng đến nhận phòng ít hơn số lượng đã đặt, nhân viên sẽ checkin những phòng khách hàng yêu cầu tại quầy. Các phòng không được checkin, hệ thống sẽ tự hủy thông tin khi quá hạn checkin

Khi khách hàng có yêu cầu chuyển phòng, nhân viên chọn mục chuyển phòng, hệ thống có chức năng tự đồng tìm kiếm phòng có thời gian phù hợp với phòng khách yêu cầu chuyển đến. Nhân viên làm thủ tục chuyển phòng và yêu cầu khách hàng thanh toán tiền cọc phát sinh nếu có.

Khi đến hạn trả phòng, nhân viên có nhiệm vụ liên hệ, thông báo cho khách hàng. Khi khách trả phòng và hoàn thành các khoản phí nhân viên sẽ xuất cho khách hóa đơn bao gồm tất cả khoản phí trong suốt quá trình khách ở tài khách sạn phí bao gồm tiền phòng, tiền dịch vụ phát sinh. Nhân viên chọn trả phòng và in hóa thì hệ thống sẽ tự động xuất ra giấy cho khách và lưu thông tin trả phòng vào hệ thống.

Hệ thống bao gồm nhân viên quản lý và nhân viên lễ tân. Nhân viên khi vào làm việc sẽ được cấp tài khoản tương đương với cấp bậc của mình.Với nhân viên lễ tân sẽ được cấp một số quyền nhất định.

Hệ thống có chức năng quản lý dịch vụ tại khách sạn, nhân viên quản lý có thể thêm một dịch vụ như thêm, xóa ( cập nhập trạng thái ngưng cung cấp), cập nhập thông tin của một dịch vụ bất kì trong hệ thống. Khi thêm mới dịch vụ nhân viên chỉ cần điền đầy đủ thông tin tại form thêm mới. Khi nhân viên muốn tìm kiếm dịch vụ người dùng chỉ cần nhập tên dịch vụ hệ thống sẽ hiện thị ra thông tin dịch vụ muốn tìm người dùng có thể cập nhập thay đổi tên dịch vụ giá dịch vụ khi cần thiết. Khi dịch vụ ngưng cung cấp quý khách có thể dùng chức năng xóa dịch vụ của hệ thống. Khi đó hệ thống sẽ cập nhập trạng thái cho dịch vụ “đã ngưng cung cấp”. Một số thông tin dịch vụ mà khách hàng có thể tham khảo như:

***Dịch vụ phòng 24/24***: dịch vụ mà khách hàng có thể sử dụng để được phục vụ thức ăn, đồ uống… tại phòng 24/24.

***Dịch vụ giặt ủi quần áo***: dịch vụ kèm theo dành cho những du khách thuê phòng dài ngày, các khách sạn kinh doanh dịch vụ này hướng đến cung cấp đầy đủ cho du khách các dịch vụ cần thiết trong cuộc sống.

***Fitness center*:** nhằm đáp ứng nhu cầu cho những khách hàng thường có thói quen rèn luyện, vận động hàng ngày

***Dịch vụ Spa***: kinh doanh khách sạn kèm theo spa sẽ mang đến cho khách hàng những khoảng thời gian thoải mái.

***Quầy bar:***  một dịch vụ kèm theo mới mẻ trong kinh doanh khách sạn, là nơi khách hàng thường lưu đến để thư giãn, trò chuyện cùng với các vị khách khác.

***Hồ bơi :*** nằm giữa khuyên viên khách sạn, hồ nước xanh biếc, mọi du khách đều được sự dụng dịch vụ khi thuê phòng tại khách sạn.

***Bãi đậu xe:*** Có sức chứ lớn có 2 bãi đậu xe cho xe máy và ô tô

***Wifi miến phí***: khách có thể kết nối wifi bất kì nơi nào tại khuôn viên của khách sạn

Để dể cho việc quản lý hệ thống đưa ra chức năng quản lý theo tầng. Hệ thống sẽ hiện thị giao diện đặt phòng theo từng tầng để quý khách cũng như là nhân viên dể dàng hình dung ví trí của phòng trong tòa nhà. Nhân viên có thể thêm tầng cho toàn nhà thông tin khi thêm một tầng vào hệ thống là tên tầng. Khi tòa nhà muốn sửa chữa nội thất. Người dùng có thể sửa tên tầng trong trường hợp cần thiết và có thể chuyển trạng thái ngưng hoạt động cho thuê tại tầng đó. Thông tin của tầng báo gồm: Tên tầng, trạng thái.

Chức năng Lập báo cáo cho phép người quản trị lấy thông tin doanh thu theo từng tháng, báo cáo tình trạng phòng sẽ được báo cáo theo từng ngày trong tháng thời gian khách thuê và số giờ khách thuê, danh sách khách đã thuê phòng, tổng hợp thành báo cáo hàng tháng hoặc báo cáo khi có yêu cầu về doanh thu, tình trạng phòng, hoặc danh sách khách hàng thuê. Người quản trị đăng nhập vào hệ thống bằng tài khoản của mình, lựa chọn chắc năng cần thống kê. Lựa chọn cách thống kê : theo tháng, tuần hoặc theo thời gian sau đó người quản trị chọn “Lưu” và có thể in báo cáo.

Hệ thống bao gồm các chức năng sau được phân quyền theo từng loại nhân viên như sau:

Quyền sử dụng:

* Khách hàng (C: Customer): đặt phòng, hủy phòng
* Nhân viên ( E: Employee ) : lập hóa đơn, quản lý thuê dịch vụ
* Nhân viên quản lý ( M :manager ): quản lý nhân viên, xem báo cáo thông kê doanh thu, quản lý tài khoản

Chức năng:

***1. Quản lý thê phòng****.(C, E, M)*

a) Đặt phòng: Hệ thống có chức năng đặt phòng tại quầy(E, M)và đặt online trên website(C) và thanh toán hóa đơn tiền cọc, khi khách hàng đặt phòng có thể đặt thêm dịch vụ tại chức năng số 2

* Đối với khách hàng thanh toán hóa đơn tiền cọc tại quầy, khách hàng trả trực tiếp bằng tiền mặt. Sau khi đặt phòng thành công, hệ thống gửi email thông báo bao gồm thông tin đặt phòng, địa chỉ, thời gian checkin....
* Đối với khách hàng đặt qua điện thoại sau khi nhân viên làm thủ tục đặt phòng hoàn tất, hệ thống sẽ gữi một email thông báo bao gồm thông tin đặt phòng, địa chỉ, thời gian checkin... cùng với đường link dẫn đến website thanh toán hóa đơn tiền cọc của khách sạn. Sau 6h (theo quy đinh của khách sạn) nếu khách hàng chưa hoàn tất thủ tục thanh toán tiền cọc , hệ thống thông báo đến nhân viên danh sách khách hàng chưa thanh toán cọc. Nhân viên có nhiệm vụ liên hệ khách hàng thông báo. Nếu chưa hoàn tất, hệ thống tự động hủy thông tin đặt phòng. Nhân viên có thể trao đổi với khách hàng để lập phiếu đặt phòng khác.
* Đối với khách hàng đặt tại online thông qua website, sau khi đồng ý đặt phòng, hệ thống chuyển đến trang thanh toán tiền cọc. Sau khi hoàn tất, hệ thống sẽ gửi một email thông báo bao gồm thông tin đặt phòng, địa chỉ, thời gian checkin....

b) Nhận phòng(E, M): khi khách hàng nhận phòng và đang ở có thể đặt thêm dịch vụ tại chức năng số 2, nếu phòng quá hạn ngày nhận, hệ thống thực hiện chức năng số 8.

c) Trả phòng – lập hóa đơn(E, M)

***2. Chức năng thuê dịch vụ(C, E)***

Đối với khách hàng đặt online trên web site, hệ thống cho phép khách hàng đặt thêm dịch vụ tại giao diện đặt phòng(C).

Đối với khách hàng đặt phòng tại quầy hoặc qua điện thoại, hệ thống cho phép nhân viên đặt thêm dịch vụ tại giao diện đặt phòng hoặc giao diện tình trạng phòng hiện tại.(E, M)

***3. Chức năng quản lý dịch vụ(E, M)***

a) Thêm dịch vụ(M)

b) Cập nhật dịch vụ(M)

c) Tạm ngưng dịch vụ(E)

***4. Quản lý phòng – tầng:*** Mỗi phòng có 3 trạng thái (da-dat, dang-o, dang-don-dep) tuy vào các chức năng sử dụng có sự thay đổi phù hợp(M)

a) Thêm phòng

b) Cập nhật thông tin phòng

c) Xóa phòng

***5. Chức năng quản lý khách hàng(E, M)***

a) Thêm khách hàng.(E)

b) Cập nhật thông tin khách hàng.(E)

c) Xóa khách hàng(M)

***6. Chức năng quản lý nhân viên(M)***

a) Thêm nhân viên

b) Cập nhật thông tin nhân viên.

c) Xóa nhân viên.

***7. Các chức năng tự động của hệ thống***:

- Khi chuyển phòng, hệ thống có chức năng tự tìm danh sách phòng phù hợp khi nhân viên chọn phòng cần chuyển.

- Hệ thống có chức năng tự động hủy thông tin đặt phòng(chuyển trạng thái “da dat → trong”) khi quá hạn checkin.

- Hiện danh sách thông tin đặt phòng chưa thanh toán cọc.

- Hiện danh sách phòng đang ở đến ngày trả phòng

**8. Chức năng báo cáo thống kê(M):**

- Thống kê số lượng dịch thuê vụ được theo tháng

- Thống kê hiệu xuất từng phòng theo tháng.

- Thống kê doanh thu theo tháng, theo năm .

Hệ thống hiển thị với dạng biểu đồ.

**8. Yêu cầu phi chức năng:**

- Hệ thống hoạt động 24/7.

- Hệ thống giao diện thân thiện, dễ sử dụng.

# CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

## 2.1 Ngôn ngữ lập trình java

### 2.1.1 Java và lịch sử phát triển

Java là một ngôn ngữ lập trình dạng lập trình hướng đối tượng (OOP). Khác với phần lớn ngôn ngữ lập trình thông thường, thay vì biên dịch mã nguồn thành mã máy hoặc thông dịch mã nguồn khi chạy, Java được thiết kế để biên dịch mã nguồn thành bytecode, bytecode sau đó sẽ được môi trường thực thi (runtime environment) chạy.

Cú pháp Java được vay mượn nhiều từ C và C++ nhưng có cú pháp hướng đối tượng đơn giản hơn và ít tính năng xử lý cấp thấp hơn. Do đó việc viết một chương trình bằng Java dễ hơn, đơn giản hơn, đỡ tốn công sửa lỗi hơn.

Java được khởi đầu bởi James - Gosling và bạn đồng nghiệp ở Sun Microsystems năm 1991. Ban đầu ngôn ngữ này được gọi là Oak (có nghĩa là cây sồi) do bên ngoài cơ quan của ông Gosling có trồng nhiều loại cây này.

Java được phát hành vào năm 1994. Sau khi Oracle mua lại công ty Sun

Microsystems năm 2009 - 2010, Oracle đã mô tả họ là "người quản lý công nghệ Java với cam kết không ngừng để bồi dưỡng một cộng đồng tham gia và minh bạch".

Lịch sử phiên bản java:

* JDK 1.0 (23 tháng 01, 1996)
* JDK 1.1 (19 tháng 2 năm 1997)
  + JDK 1.1.5 *(Pumpkin)* 03 tháng 12 năm 1997
  + JDK 1.1.6 *(Abigail)* 24 tháng 4 năm 1998
  + JDK 1.1.7 *(Brutus)* 28 tháng 9 năm 1998
  + JDK 1.1.8 *(Chelsea)* 08 tháng 4 năm 1999
* J2SE 1.2 *(Playground)* 08 tháng 12 năm 1998
  + J2SE 1.2.1 *(không có)* 30 tháng 3 năm 1999
  + J2SE 1.2.2 *(Cricket)* 08 tháng 7 năm 1999
* J2SE 1.3 *(Kestrel)* 08 tháng 5 năm 2000
  + J2SE 1.3.1 *(Ladybird)* 17 tháng 5 năm 2001
* J2SE 1.4.0 *(Merlin)* 06 tháng 02, 2002
  + J2SE 1.4.1 *(Hopper)* 16 tháng 9 năm 2002
  + J2SE 1.4.2 *(Mantis)* 26 tháng 6 năm 2003
* J2SE 5 (1.5.0) *(Tiger)* 30 tháng 9 năm 2004
* Java SE 6 (còn gọi là *Mustang*), được công bố [11 tháng 12](https://vi.wikipedia.org/wiki/11_th%C3%A1ng_12) năm [2006](https://vi.wikipedia.org/wiki/2006). Các bản cập nhật 2 và 3 được đưa ra vào năm 2007, bản cập nhật 4 đưa ra tháng 1 năm 2008.
* JDK 6.18, 2010
* Java SE 7 (còn gọi là *Dolphin*), được bắt đầu từ tháng 8 năm 2006 và công bố ngày 28 tháng 7 năm 2011.
* JDK 8, 18 tháng 3 năm 2014
* JDK 9, 21 tháng 9 năm 2017
* JDK 10, 20 tháng 3 năm 2018

### 2.1.2 Java Reflection

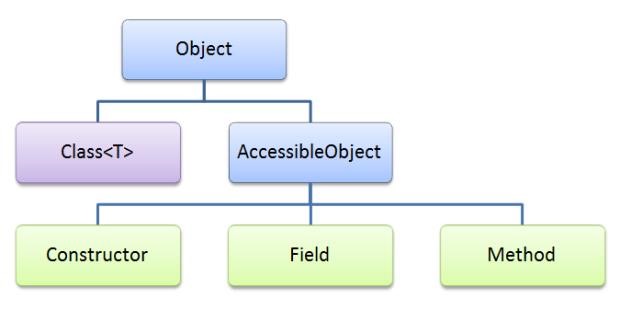
Trong phần này sẽ trình bày về *Reflection* trong Java. Java reflection là nền tảng cho cách thức làm việc của tất cả các dependency injection framework trong Java như Dagger, Google Guice, PicoContainer và Spring, chính vì vậy việc hiểu về Java reflection sẽ giúp chúng ta dễ dàng tiếp cận Spring hơn.

*Reflection* là kĩ thuật rất cần thiết để lấy các thông tin của một kiểu dữ liệu. Dựa vào đó ta có thể kích hoạt (gọi các phương thức) hoặc tạo thể hiện của kiểu dữ liệu đó. Một ứng dụng quan trọng của reflection mà chúng ta có thể biết là Java Bean. Nhờ đó, các IDE (như NetBeans) hoặc các framework có thể lấy được các thông tin và thiết lập giá trị cho các đối tượng trong môi trường run-time.

**Kiến trúc của Java Reflection API**

Các lớp được dùng trong reflection nằm trong hai package là *java.lang* và *java.lang.reflect*. Package *java.lang.reflect* bao gồm ba lớp chính mà chúng ta cần biết là *Constructor*, *Field* và *Method*:

* ***Class<T>:*** Lớp này đại diện cho các lớp, interface và chứa các phương thức dùng để lấy các đối tượng kiểu Constructor, Field, Method.
* ***AccessibleObject***: Các kiểm tra về phạm vi truy xuất (public, private, protected) của field, method, constructor sẽ được bỏ qua. Nhờ đó chúng ta có thể dùng reflection để thay đổi, thực thi các thành phần này mà không cần quan tâm đến phạm vi truy xuất của nó.
* ***Constructor***: Chứa các thông tin về một constructor của lớp.
* ***Field***: chứa các thông tin về một field của lớp, interface.
* ***Method***: chứa các thông tin về một phương thức của lớp, interface.



*Hình 2.1: Kiến trúc tổng quát Java Reflaction API* Dưới đây là một ví dụ đơn giản dùng để in ra thông tin của một lớp trong môi trường run time chỉ cần chúng ta biết được tên đầy đủ (bao gồm cả phần package) của lớp đó. Giả sử chúng ta có class User như sau:

|  |  |  |
| --- | --- | --- |
| User.java | | |
| **package** com.nthienan;    **public** **class** User { **private** String username; **private** String password;    **public** User() { }    **public** User(String username, String password) {  **this**.username = username; **this**.password = password;  }    **public** **void** printHello(String fullName){ System.***out***.println("Hello" + fullName);  } | | |
|  |  | **public** String getUsername() { **return** username;  }  **public** **void** setUsername(String username) {  **this**.username = username;  }  **public** String getPassword() { **return** password;  }  **public** **void** setPassword(String password) { **this**.password = password; |
| } |  | } |

Hàm main như sau:

|  |
| --- |
| Main.java |
| **package** com.nthienan;    **import** java.lang.reflect.Constructor; **import** java.lang.reflect.Field; **import** java.lang.reflect.Method;    **public** **class** Main {  **public** **static** **void** main(String[] args) {  **try** {  Class<?> c = Class.*forName*("com.nthienan.User");    System.***out***.println("\*\*\*\*\*Class name\*\*\*\*\*");  System.***out***.println("Name: " + c.getName());  System.***out***.println("Simple name: " + c.getSimpleName());    Field[] fields = c.getDeclaredFields(); System.***out***.println("\n\*\*\*\*\*Field\*\*\*\*\*");  **for** (Field f : fields) {  System.***out***.println(f);  }    Constructor[] constructors = c.getConstructors(); System.***out***.println("\n\*\*\*\*\*Constructor\*\*\*\*\*"); **for** (Constructor constructor : constructors) {  System.***out***.println(constructor);  }    Method[] methods = c.getDeclaredMethods(); System.***out***.println("\*\*\*\*\*Method\*\*\*\*\*");  **for** (Method m : methods) {  System.***out***.println(m);  }    } **catch** (ClassNotFoundException e) { e.printStackTrace();  }  } |

}

Kết quả:

|  |
| --- |
| \*\*\*\*\*Class name\*\*\*\*\*  Name: com.nthienan.User  Simple name: User    \*\*\*\*\*Field\*\*\*\*\*  private java.lang.String com.nthienan.User.username private java.lang.String com.nthienan.User.password    \*\*\*\*\*Contructor\*\*\*\*\* public com.nthienan.User()  public com.nthienan.User(java.lang.String,java.lang.String)  \*\*\*\*\*Method\*\*\*\*\* public java.lang.String com.nthienan.User.getPassword() public void com.nthienan.User.printHello(java.lang.String) public void com.nthienan.User.setPassword(java.lang.String) public java.lang.String com.nthienan.User.getUsername() public void com.nthienan.User.setUsername(java.lang.String) |

Chúng ta có thể tạo ra một thể hiện (instance) của class này một cách dễ dàng bằng một trong hai cách như ví dụ sau. Vì phương thức *newInstance( )* trả về một đối tượng kiểu Object nên cần phải ép kiểu đối tượng này về kiểu User.

Class<?> c = Class.*forName*("com.phung.User");

//tạo một đối tượng với constructor không có tham số

User instance = (User) c.newInstance();

//tạo đối tượng với constructor có tham số

Constructor<?> con = c.getConstructor(String.**class**, String.**class**);

User instance1 = (User) con.newInstance("phungbminh", "123");

Để thực thi một phương thức cụ thể, chúng ta cần sử dụng hai phương thức sau: *Class.getMethod(String name, Class[] parameterTypes)*: trả về đối tượng Method đại diện cho một phương thức của lớp. Phương thức này được xác định qua tên và các kiểu tham số.

*Method.invoke(Object obj, Object[] args)* thực thi phương thức tương ứng của đối tượng *obj* với các tham số *args*.

Ví dụ sau thực thi phương thức *printHello(String fullName)* của lớp User bằng cách tạo một đối tượng User và truyền vào làm tham số đầu tiên trong phương thức *Method.invoke( )*. Nếu phương thức *printHello( )* là static chỉ cần truyền *null* vào làm tham số đầu tiên của phương thức *Method.invoke( ).*

//tạo một đối tượng với constructor không có tham số

User instance = (User) c.newInstance();

Method method = c.getMethod("printHello", String.**class**); method.invoke(instance, "World");

Kết quả:

Hello World

## 2.2 Spring framework

### 2.2.1 Tổng quan về Spring

Spring framework, hay ngắn hơn là Spring, là một cấu trúc dùng để xây dựng chương trình ứng dụng mã nguồn mở dành cho ngôn ngữ lập trình Java. Phiên bản đầu tiên của nó do Rod Johnson viết và đã được ông xuất bản cùng với quyển sách đầu tay Expert One-on-One JavaEE Design and Development (Nhà xuất bản Wrox Press, Tháng 10 năm 2002).

Kiến trúc của Spring framework được ra mắt công chúng lần đầu tiên hồi tháng 6 năm 2003 dưới Giấy phép Apache - phiên bản 2.0. Phiên bản 1.0 đánh dấu mốc thành đạt đầu tiên được xuất bản vào tháng 3 năm 2004 và tiếp đó vào tháng 9 năm 2004, tháng 3 năm 2005.

Tuy Spring framework không bắt buộc người ta phải tuân theo một mô hình lập trình cụ thể nào, song nó lan truyền rộng rải trong cộng đồng những người viết chương trình dùng Java, như một hình thức chủ yếu thay thế cho mô hình Enterprise Java Bean. Theo thiết kế, bộ framework này giải phóng lập trình viên dùng Java, cho phép họ nhiều quyền tự do hơn và đồng thời cung cấp một giải pháp tiện lợi, đầy đủ dẫn chứng bằng tài liệu, dễ dàng sử dụng, phù hợp với những thực hành thông dụng trong công nghệ phần mềm.

Bên cạnh những đặc trưng nền tảng của Spring framework là những cái có thể dùng được trong bất cứ một chương trình ứng dụng Java nào, rất nhiều các mở rộng và tiến bộ trong việc kiến tạo các trình ứng dụng dành cho nền tảng mạng web (web-based application) dựa trên nền Java Enterprise cũng tồn tại nữa. Spring framework nổi tiếng cũng một phần do chính đặc thù kể trên và được giới thương mại công nhận như một nền tảng kiến trúc có tầm quan trọng trong chiến lược kiến tạo phần mềm.

### 2.2.2 Lịch sử phát triển

Phần đầu tiên của Spring framework ban đầu nguyên được Rod Johnson viết vào năm 2000. Vào năm 2001, những mô hình lập trình cho các trình ứng dụng Web được đại đa số sử dụng đều do Java Servlet API và Enterprise Java Bean cung cấp.

Năm 2003 thành lập dự án phát tại Sourceforge để phát triển Spring. Sau khi phát triển trên nền tảng ban đầu hơn một năm họ đã phát hành phiên bản đầu tiên (1.0) vào tháng 3 năm 2004.

Spring framework đã làm cho những kỹ thuật vốn không được biết đến mấy trở nên những kỹ thuật được mọi người ưa chuộng trong một thời gian ngắn ngủi. Một trong những kỹ thuật nổi tiếng hơn cả là kỹ thuật “đảo ngược quyền điều khiển” (Inversion of Control, IoC).

Năm 2005 cho thấy mức độ hưởng ứng nâng cao hơn những năm trước, nguyên do cũng vì những phiên bản mới được giới thiệu với những cột mốc đáng kể và những tính năng mới được thêm vào. Diễn đàn Spring (Spring Forum) ra đời cuối năm 2004 cũng góp phần không nhỏ trong việc nâng cao tính phổ biến của bộ framework và từ đó đến nay đã sinh trưởng trở thành một nguồn thông tin quan trọng, giúp đỡ cho người sử dụng.

Vào tháng 12 năm 2005, hội thảo đầu tiên về Spring Framework đã được tổ chức tại Miami, Florida thu hút 300 nhà phát triển trong 3 ngày và tiếp theo đó cuộc hội thảo ở Antwerp vào tháng 6 năm 2006, thu hút hơn 400 người.

### 2.2.3 Một số khái niệm chính

Hai trong những thành phần chủ chốt và là nền tảng tạo nên sức mạnh của Spring chính là IoC và DI.

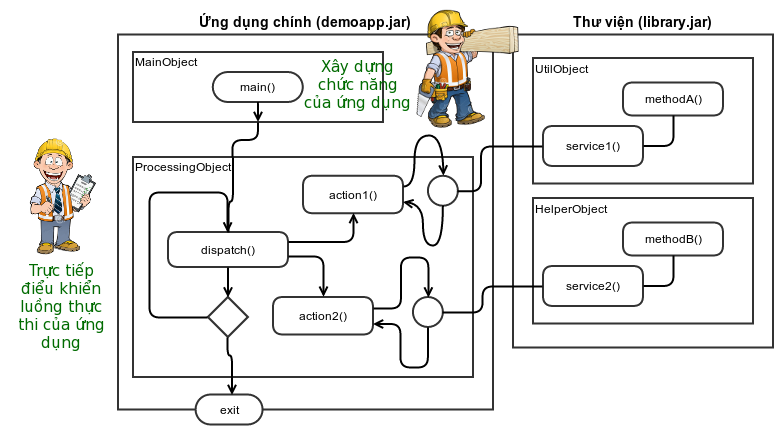
2.2.3.1 Inversion of Control (IoC)

IoC Container trong Spring được xây dựng dựa trên nguyên lý Inversion of Control (đảo ngược điều khiển) đã xuất hiện khá lâu trong các mẫu hình thiết kế (design pattern), và được phổ biến rộng rãi nhờ Robert C. Martin và Martin Fowler. Để hiểu về Spring, trước tiên chúng ta cần hiểu khái niệm IoC là gì? Muốn vậy, chúng ta cần trả lời câu hỏi: Control (điều khiển) trong chương trình phần mềm là gì, và Inversion (sự đảo ngược) đối với điều khiển trong ngữ cảnh này được hiểu như thế nào?

Khái niệm Control Flow (tạm dịch là luồng thực thi) được sử dụng cho trình tự thực hiện các câu lệnh, chỉ thị hoặc lời gọi hàm trong một chương trình, khi chương trình này thực thi.

Do chương trình ngày càng phức tạp, nên các lập trình viên áp dụng phương pháp lập trình hướng đối tượng nhằm phân loại, chia tách các chức năng và gom thành các đối tượng. Các lập trình viên còn tạo dựng các thư viện tạo sẵn để có thể sử dụng lại. Luồng thực thi của chương trình, trong những tình huống cần xem xét ở mức tổng thể, không còn quan tâm đến các bước thực thi câu lệnh cụ thể nữa, mà chỉ xem xét đến quá trình gọi phương thức của các đối tượng trong ứng dụng cũng như các đối tượng của thư viện dựng sẵn.

Các lập trình viên, khi xây dựng ứng dụng từ đầu, đã thực hiện hai nhiệm vụ: trực tiếp điều khiển luồng thực thi của chương trình và xây dựng các chức năng để đáp ứng nghiệp vụ của ứng dụng. Thực tế, có nhiều chương trình hoặc bộ phận trong chương trình có luồng thực thi rất giống nhau, chẳng hạn phần tương tác với HTTP trong các ứng dụng web, phần unit testing trong các ứng dụng,… Việc trực tiếp tạo dựng và kiểm soát luồng thực thi của chương trình lặp đi lặp lại khi xây dựng nhiều ứng dụng sẽ làm mất nhiều công sức, chi phí, tạo ra sự nhàm chán và dễ phát sinh lỗi. Điều này tạo ra động lực cũng như môi trường để nguyên lý đảo ngược điều khiển nảy nở và phát triển.



*Hình 2.2: Luồng điều khiển của chương trình bình thường*

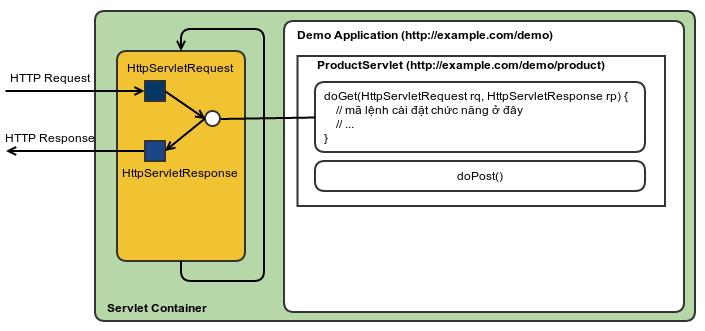
Vậy sự “đảo ngược” (inversion) luồng điều khiển chương trình trong ngữ cảnh này được hiểu như thế nào? Chúng ta hãy xem xét 2 ví dụ dưới đây.

*Ví dụ 1: Ứng dụng web trong Java với JavaServlet*

Khi lập trình ứng dụng web bằng JavaServlet cần thực hiện các bước lập trình như sau:

* Tạo lớp đối tượng kế thừa từ HttpServlet, override các phương thức doGet(), doPost(),…
* Đăng ký trong file cấu hình Deployment Descriptor tương ứng Servlet này với đường dẫn xác định.
* Lớp đối tượng Servlet chúng ta tạo ra sẽ được gọi đến khi có một truy vấn HTTP có đường dẫn “khớp” với đường dẫn khai báo trong Deployment Descriptor.

Vậy ai kích hoạt ứng dụng chúng ta viết để đáp ứng mỗi khi có HTTP Request gửi đến? Ai chịu trách nhiệm chuyển đổi các thông điệp HTTP (HTTP Request và HTTP Response) thành các đối tượng Java (HttpServletRequest và HttpServletResponse) để truyền cho các hàm doGet(), doPost()? Đó chính là Servlet Container.

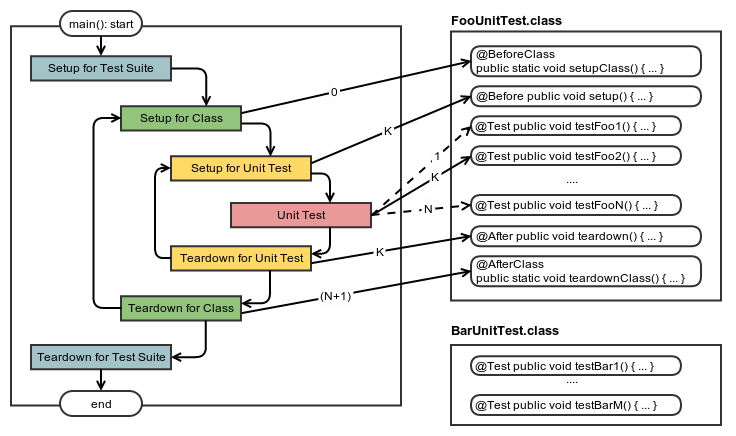


*Hình 2.3: Servlet Container*

*Ví dụ 2: Lập trình kiểm thử đơn vị (Unit testing) với Junit*

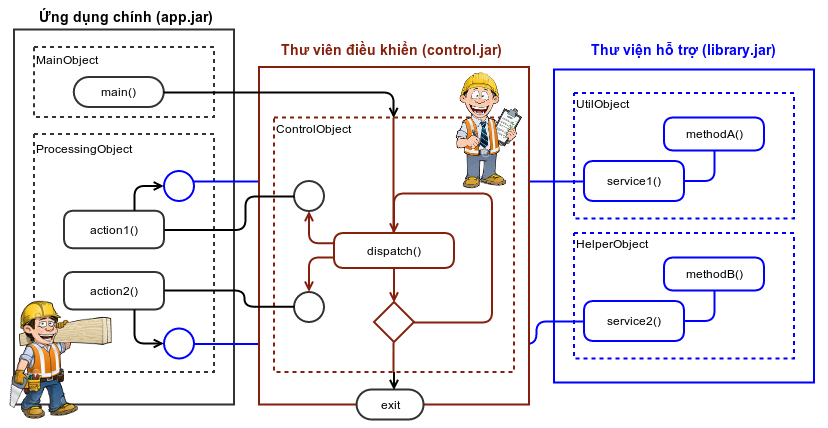
Trong quá trình phát triển các thành phần chức năng của ứng dụng, chúng ta thường áp dụng kiểm thử đơn vị để đảm bảo chức năng đó vẫn chạy đúng trong suốt quá trình ứng dụng được mở rộng và phát triển thêm. Để tạo bộ unit test, chúng ta chỉ cần tạo một lớp đối tượng, định nghĩa các phương thức khởi tạo, phương thức kết thúc và các phương thức test. Sau đó, chúng ta chỉ việc chạy bộ test để kiểm thử.

Việc điều khiển trình tự thực thi các phương thức được giao cho thư viện bên ngoài đảm nhiệm chẳng hạn như TestNG hoặc JUnit.



*Hình 2.4: Kiểm thử đơn vị với JUnit*

Với hai ví dụ trên, chúng ta nhận thấy trong các ứng dụng đã có sự thay đổi vai trò. Ứng dụng không còn ôm đồm vừa trực tiếp tạo dựng và kiểm soát luồng thực thi, vừa xây dựng chức năng nghiệp vụ. Việc kiểm soát luồng thực thi được tách khỏi chức năng nghiệp vụ và bị đẩy ra bên ngoài. Người lập trình đã ủy thác việc kiểm soát luồng thực thi ứng dụng cho một thành phần (thường là thư viện dựng sẵn) bên ngoài đảm nhiệm, chỉ còn tập trung vào chức năng chính của ứng dụng.



*Hình 2.5: Mô hình ứng dụng áp dụng IoC*

Như vậy, khái niệm “đảo ngược” ở đây chính là chuyển nhiệm vụ kiểm soát lưu trình thực thi từ ứng dụng cho một thành phần chuyên trách (thường là một thư viện phần mềm khung – framework – dựng sẵn ở bên ngoài). Ứng dụng chính chúng ta quan tâm phát triển không kiểm soát việc điều khiển luồng thực thi nữa, mà chỉ tập trung vào việc định nghĩa chức năng. Thư viện phần mềm khung chuyên trách kiểm soát điều khiển sẽ dựa trên mô tả trong cấu hình của ứng dụng để thay mặt ứng dụng điều phối luồng thực thi trong chương trình.

Dễ dàng nhận thấy thư viện phần mềm khung này khác với các thư viện thông thường ở chỗ: thư viện thông thường cung cấp các chức năng và chờ được ứng dụng gọi đến, còn thư viện phần mềm khung tạo dựng luồng thực thi và gọi đến các chức năng của ứng dụng.

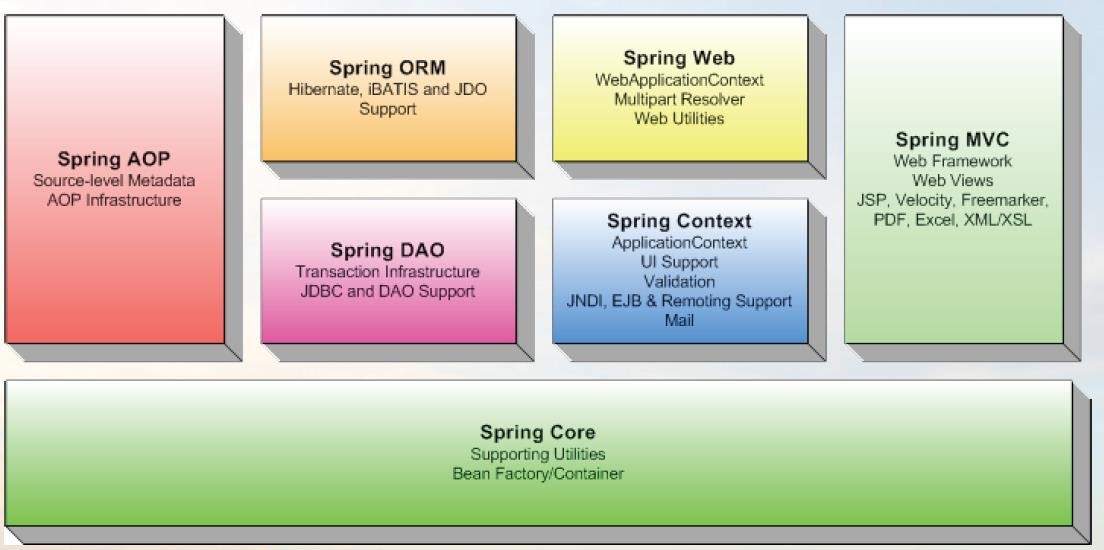
Nói một cách ngắn gọn IoC là một design pattern và tập hợp các kỹ thuật lập trình liên quan, trong đó luồng thực thi của một hệ thống bị đảo ngược so với cách tương tác truyền thống. IoC trong Spring cũng hoàn toàn mang ý nghĩa như trên. Trong Spring các đối tượng chính để xây dựng ứng dụng thì được quản lý bởi Spring IoC container. Và IoC container gọi các đối tượng đó là các bean. Một bean chỉ đơn giản là một đối tượng được khởi tạo và quản lý bởi Spring IoC container.

2.2.3.2 Dependency Injection (DI)

Dependency Injection là khả năng liên kết giữa các thành phần lại với nhau, đó chính là việc các thuộc tính trong một đối tượng được “tiêm chích” (injection) để tham chiếu lần lượt đến các đối tượng khác. Dependency Injection trong Spring hoạt động dựa trên Java Reflection.

### 2.2.4 Các module

Spring được xây dựng với 7 module chính (Hình 2-6).



*Hình 2.6: Các module của Spring framework*

* **Spring Core**: Core package là phần cơ bản nhất của Spring, cung cấp những đặc tính như IoC (Inversion of Control) và DI (Dependency Injection). Khái niệm cơ bản là BeanFactory, một cài đặt của Factory pattern, cho phép “móc nối” sự phụ thuộc giữa các đối tượng trong file cấu hình.
* **Spring Context**: Spring context là một file cấu hình để cung cấp thông tin ngữ cảnh của Spring. Spring context cung cấp các service như JNDI access, EJB integration, e-mail, internalization, validation, và scheduling functionality.
* **Spring AOP (Aspect – Oriented Programming)**: Spring AOP module tích hợp chức năng lập trình hướng khía cạnh vào Spring framework thông qua cấu hình của nó. Spring AOP module cung cấp các dịch vụ quản lý giao dịch cho các đối tượng trong bất kỳ ứng dụng nào sử dụng Spring. Với Spring AOP chúng ta có thể tích hợp declarative transaction management vào trong ứng dụng mà không cần dựa vào EJB component. Spring AOP module cũng đưa lập trình metadata vào trong Spring. Sử dụng cái này chúng ta có thể thêm annotation vào source code để hướng dẫn Spring nơi và làm thế nào để liên hệ với aspect.
* **Spring DAO (Data Access Object)**: Tầng JDBC và DAO đưa ra một cây phân cấp exception để quản lý kết nối đến database, điều khiển exception và thông báo lỗi được ném bởi vendor của database. Tầng exception đơn giản điều khiển lỗi và giảm khối lượng code mà chúng ta cần viết như mở và đóng kết nối.

Module này cũng cung cấp các dịch vụ quản lý giao dịch cho các đối tượng trong ứng dụng Spring.

* **Spring ORM (Object Relational Mapping):** Spring có thể tích hợp với một vài ORM framework để cung cấp Object Relation tool bao gồm: JDO, Hibernate, OJB và iBatis SQL Maps.
* **Spring Web**: Nằm trên application context module, cung cấp context cho các ứng dụng web. Spring cũng hỗ trợ tích hợp với Struts, JSF và Webwork. Web module cũng làm giảm bớt các công việc điều khiển nhiều request và gắn các tham số của request vào các đối tượng domain.
* **Spring MVC**: MVC Framework thì cài đặt đầy đủ đặc tính của MVC pattern để xây dựng các ứng dụng Web. MVC framework thì cấu hình thông qua giao diện và chứa được một số kỹ thuật view bao gồm: JSP, Velocity, Tiles và generation of PDF và Excel file.

### 2.2.5 Spring Core

**2.2.5.1 Bean**

Trong Spring các object được quản lý bởi IoC container và được gọi là bean. Một bean thì đơn giản là một đối tượng được khởi tạo, phân phát và được quản lý bởi IoC container. Sự phụ thuộc giữa chúng được phản ánh trong configuration metadata.

Một Spring container sẽ quản lý một hoặc nhiều bean. Các bean được tạo ra nhờ vào thông tin cấu hình được chỉ ra trong thẻ <bean /> nếu sử dụng XML config. Bên trong container các bean được biểu diễn như một đối tượng của lớp BeanDefinition, nó chứa các thông tin sau:

* Class của đối tượng mà bean được cấu hình (bao gồm cả phần package)
* Các thông số cấu hình hành vi của bean như: phạm vi (scope), vòng đời

(lifecycle),…

* Các tham chiếu đến các bean khác, các tham chiếu này được gọi là collaborators hoặc dependencies
* Các thiết lập khác như số connection kết nối đến bean,…

Các thuộc tính sau đây thường được sử dụng để định nghĩa một bean: class (instantiating beans), name (naming beans), scope, constructor arguments, properties, autowiring mode, lazy-initialization mode, initialization method, destruction method 2.2.5.1.1 Tên bean (Naming beans)

Mỗi bean có một hoặc nhiều định danh (indentifiers), các định danh này phải là duy nhất trong một container. Một bean thường chỉ có duy nhất một định danh, tuy nhiên nếu cần nhiều hơn một định danh thì có thể sử dụng bí danh (alias).

Trong cấu hình dạng XML, có thể sử dụng thuộc tính *id* và/hoặc *name* để chỉ định định danh cho bean. Thuộc tính *id* cho phép chỉ định chính xác một định danh duy nhất cho bean.

Chúng ta có thể không cần chỉ định *id* hoặc *name* cho bean, khi đó container sẽ tự tạo ra một định danh duy nhất cho bean. Tuy nhiên nếu muốn sử dụng tham chiếu ở bean khác bằng cách sử dụng thuộc tính *ref* thì cần phải chỉ định một cách rõ ràng.

Thông thường các bean được đặt tên giống như khai báo biến trong Java, theo quy ước camel-cased. Ví dụ như: accountManager, userDao, loginController…Đặt tên cho bean sẽ giúp file cấu hình dễ đọc và hiểu hơn và nếu sử dụng Spring AOP nó sẽ giúp rất nhiều khi thêm các advice vào bean thông qua tên của bean.

**2.2.5.1.2 Khởi tạo bean**

Một bean thực chất là một “công thức” cho việc tạo ra một hoặc nhiều đối tượng [6, tr34]. Nếu sử dụng configuration metadata dạng XML (XML-based configuration metadata) cần phải chỉ định rõ class của đối tượng sẽ được bean quản lý và khởi tạo khi có yêu cầu. Có 3 cách để khởi tạo một bean:

***Khởi tạo bean với hàm dựng (constructor)***

Thông thường khi tạo một bean bằng phương pháp sử dụng constructor, tất cả các class đều có thể dùng được và phù hợp với Spring [6, tr34]. Có nghĩa là các class không cần phải thực thi (implement) bất kỳ giao diện (interface) cụ thể nào. Tuy nhiên, tùy thuộc vào loại IoC có thể cần một constructor mặc định (không đối số).

Spring IoC container có thể quản lý hầu như bất kỳ class nào mà chúng ta muốn nó quản lý [6, tr34]. Hầu hết người dùng Spring thích sử dụng cách tạo một bean bằng việc sử dụng một constructor mặc định (không có đối số) và các setter và getter [6, tr34].

***Khởi tạo bean với phương thức tĩnh (static factory method)***

Để tạo một bean sử dụng phương thức tĩnh, thuộc tính *class* chỉ ra lớp chứa phương thức tĩnh và thuộc tính *factory-name* chỉ ra tên của phương thức này được định nghĩa bên trong class vừa chỉ định. Ví dụ dưới đây sẽ cho thấy rõ việc tạo một bean bằng cách gọi phương thức tĩnh.

|  |
| --- |
| **public** **class** MusicService { **private** **static** MuisicService musicService = **new** MuisicService();  **private** MuisicService() { }  **public static** MuisicService createInstance() { **return** musicService;  }  } |

<bean id=*"*musicService*"* class=*"com.nthienan.MusicService"* factory-method=*"*createInstance*"* />

Lưu ý thuộc tính *class* trong tag bean không phải chỉ định kiểu (class) của đối tượng mà thật ra chỉ là chỉ định class chứa phương thức tĩnh được chỉ định ở thuộc tính *factoty-method* được gọi để tạo ra đối tượng mà thôi, trong ví dụ này chính là phương thức *createInstance()* trong lớp ***MusicService***.

***Khởi tạo bean sử dụng phương thức của đối tượng (instance factory method)***

Tương tự như việc sử dụng phương thức tĩnh, với phương pháp này sử dụng một phương thức non-static của một bean để tạo ra bean mới. Để sử dụng cách này thuộc tính *class* của tag bean sẽ không được chỉ định, thay vào đó thuộc tính *factory-bean* chỉ ra tên của bean mà có chứa phương thức dùng để tạo ra đối tượng. Tên của phương thức này được chỉ định trong thuộc tính *factory-method*. Ví dụ dưới đây sẽ chỉ ra cách sử dụng phương pháp này.

|  |
| --- |
| **public** **class** MusicService { **private** **static** SongService songService = **new** SongService(); **private** MuisicService() { }  **public** SongService createSongServiceInstance() { **return** songService;  }  } |

*<!-- factory bean -->*

<bean id=*"*musicService*"* class=*"com.nthienan.MusicService"*>

*<!-- các cấu hình khác cho bean này ở đây -->*

</bean>

*<!-- bean được tạo thông qua factory bean -->*

<bean id=*"*songService*"* factory-bean=*"*musicService*"*

Factory-method=*"*createSongServiceInstance*"* />

*2.2.5.1.3 Phạm vi của bean (Bean scopes)*

Khi định nghĩa một bean chúng ta đã đưa ra một “công thức” cho việc tạo một thể hiện thực sự của một class [6, tr57]. Việc khởi tạo một hay nhiều thể hiện, khi nào thì một thể hiện được tạo ra sẽ phụ thuộc vào phạm vi của bean. Spring đưa ra năm phạm vi của một bean, trong đó ba phạm vi chỉ có khi xây dựng các ứng dụng web.

Năm phạm vi của một bean được trình bày tóm tắt trong bảng dưới đây:

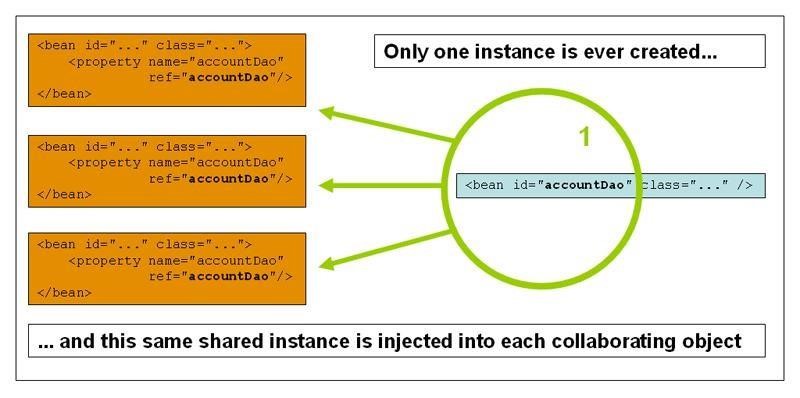
|  |  |
| --- | --- |
| **Phạm vi** | **Giải thích** |
| Singleton | Chỉ có một thể hiện duy nhất được tạo ra cho bean trong mỗi Spring IoC container. Đây là phạm vi mặc định của mỗi bean |
| Prototype | Ngược lại với singleton, prototype cho phép nhiều hơn một đối tượng được tạo ra trên mỗi container. |
| Request | Một thể hiện cụ thể sẽ được tạo ra cho mỗi HTTP request. Phạm vi này chỉ có trong container của các ứng dụng web  (WebApplicationContext) |
| Session | Tương tự như request nhưng, một thể hiện cụ thể sẽ được tạo ra cho mỗi HTTP session. Phạm vi này chỉ có trong container của các ứng dụng web (WebApplicationContext) |
| Global session | Một thể hiện cụ thể sẽ được tạo ra cho một global HTTP session. Phạm vi này chỉ có trong container của các ứng dụng web. |
| Appplication | Một thể hiện cụ thể sẽ được tạo ra cho một Servlet context. Phạm vi này chỉ có trong container của các ứng dụng web. |

*Bảng 2-1: Phạm vi của bean*

Thông thường thì singleton và prototype là hai phạm vi của bean được sử dụng nhiều nhất. Dưới đây xin trình bày chi tiết về 2 phạm vi này.

**Singleton scope.**

Khi khai báo một bean với phạm vi là singleton điều này có nghĩa là chỉ có duy nhất một thể hiện của class được chỉ ra trong bean trong một container. Thể hiện này sẽ được lưu trữ trong cache của Spring IoC container, tất cả các yêu cầu tiếp theo tham chiếu đến bean này thì thể hiện được lưu trữ trong cache sẽ được trả về.



*Hình 2.7: Singleton scope*

Ví dụ dưới đây cho thấy cách khai báo một bean với phạm vi singleton trong XML:

|  |
| --- |
| <bean id=*"*accountDao*"* class=*"com.phung.AccountDao"*></bean>  *<!-- hoặc -->*  <bean id=*"*accountDao*"* class=*"com.phung.AccountDao"* scope=*"*singleton*"*></bean> |

**Prototype scope**

Ngược lại với singleton, một bean với phạm vi là prototype thì mỗi khi có một yêu

c

ầ

u tham chi

ế

u đ

ế

n bean này thì m

ộ

t th

ể

hi

ệ

n c

ụ

th

ể

c

ủ

a bean s

ẽ

đư

ợ

c t

ạ

o ra.

Kh

ai báo m

ộ

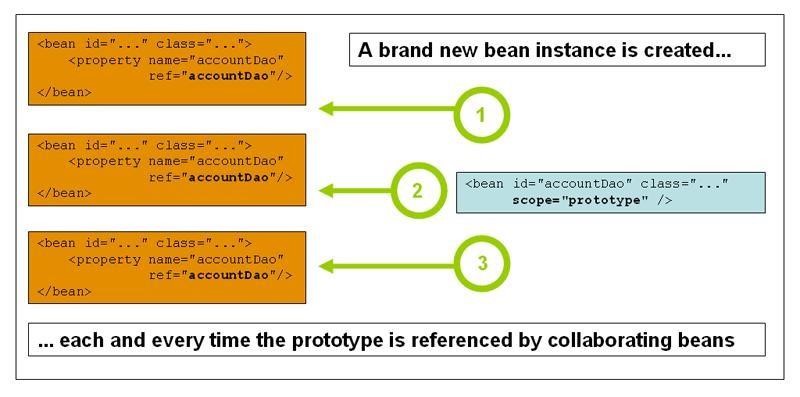
t bean v

ớ

i ph

ạ

m vi prototype trong XML như sau:



*Hình*

*2*

*.*

*8*

*:*

*Prototype scope*

<bean id=*"*accountDao*"* class=*"com.phung.AccountDao"* scope=*"*prototype*"*></bean>

2.2.5.2 Spring IoC Container

Spring là một framework thực thi theo nguyên tắc Inversion of Control, IoC cũng được biết đến như Dependency Injection [6, tr27]. Nó là một quá trình xử lý ở nơi các object định nghĩa sự phụ thuộc (dependency). Khi các đối tượng hoạt động với nhau chỉ thông qua các tham số của constructor, tham số của các method hoặc các property để thiết lập thể hiện sau khi được khởi tạo. Container sau đó “tiêm” các đối tượng phụ thuộc khi nó được tạo ra từ các bean. Quá trình này về cơ bản là sự đảo ngược, các bean sẽ điều khiển các thể hiện hoặc vị trí phụ thuộc bằng cách khởi tạo trực tiếp từ class của chúng.

Package *org.springframework.beans* và *org.springframework.context* là 2 package cơ bản cho IoC container của Spring. Interface BeanFactory cung cấp kỹ thuật để cấu hình nâng cao và quản lý bất kỳ loại object nào. ApplicationContext kế thừa BeanFactory, ApplicationContext thêm vào một số tính năng như tích hợp để dễ dàng hoạt động với các tính năng của Spring AOP (Aspect Oriented Programming) như Message resource handling, event publication; và một số lớp context đặc biệt như WebApplicationContext được sử dụng trong các ứng dụng web. Nói một cách ngắn gọn, BeanFactory cung cấp cơ chế cấu hình cho framework và các chức năng cơ bản. ApplicationContext thêm vào một số chức năng nâng cao. Trong phần này sẽ sử dụng ApplicationContext để để mô tả cho IoC container của Spring.

Interface *org.springframework.context.ApplicationContext* chịu trách nhiệm khởi tạo, cấu hình và phân phát các bean. Container lấy các chỉ dẫn cho việc khởi tạo, cấu hình và phân phát bằng cách đọc các configuration metadata. Các configuration metadata được trình bày bằng nhiều cách khác nhau như trong file XML, Java annotaion hoặc trong Java code. Điều này cho phép biểu diễn các đối tượng và sự phụ thuộc lẫn nhau giữa chúng trở nên phong phú hơn.

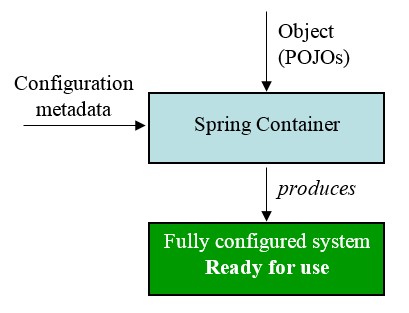
Một số class thực thi (implementation) interface ApplicationContext như Class

PathXmlApplicationContext, FileSystemXmlApplicationContext,

AnnotationConfigApplicationContext,… XML là cách truyền thống và thường được sử dụng để định nghĩa các configuration metadata, chúng ta cũng có thể khai báo để container sử dụng các annotation như configuration metadata hoặc trực tiếp trong code Java bằng cách cấu hình thêm trong file XML cho phép sử dụng các annotation như sau: <context:annotation-config/>

Hình 2-9 mô tả cách thức làm việc của Spring. Các class trong ứng dụng sẽ kết hợp với các configuration metadata để sau khi ApplicationContext được tạo ra và khởi tạo chúng ta có được một cấu hình đầy đủ và có thể thực thi được.

Như hình chúng ta thấy các configuration metadata sẽ nói với Spring container khởi tạo, cấu hình và phân phối các object trong ứng dụng của chúng ta như thế nào.



*Hình 2.9: Spring IoC Container*

Thông thường các configuration metadata được biểu diễn dưới dạng XML, vì vậy trong phần này sẽ sử dụng configuration metadata dưới dạng này để nói về các khái niệm và tính năng của IoC container.

Lưu ý rằng không phải chỉ có các configuration metadata dưới dạng XML mới được Spring container chấp nhận, như đã nói ở phần trên ngoài các metadata dạng XML chúng ta còn có thể sử dụng Java annotaion hoặc trực tiếp trong code Java (Java config). Kể từ phiên bản 2.5 Spring đã hổ trợ kiểu cấu hình dựa trên các Java annotation. Và kể từ Spring 3.0 nhiều tính năng đã được cung cấp bằng cách sử dụng Java config và nó trở thành một phần của Spring core. Vì vậy chúng ta có thể định nghĩa bean bên ngoài các class của ứng dụng bằng cách sử dụng Java code hơn là file XML. Để sử dụng tính năng này xem thêm các annotation như @Configuration, @Bean, @Import và @DependOn. Cấu hình trong Spring bao gồm ít nhất là một và thường là nhiều hơn một định nghĩa bean mà container cần phải quản lý. Trong metadata dạng XML để khai báo và cấu hình một bean ta sử dụng thẻ <bean /> bên trong thẻ <beans />. Nếu sử dụng dạng Java config thông thường sẽ sử dụng annotaion @Bean bên trong một class với annotaion @Configuration.

Ví dụ dưới đây là sẽ cho thấy cấu trúc cơ bản của một file XML trình bày các configuration metadata.

|  |
| --- |
| <?xml version=”1.0” encoding=”UTF-8”?>  <beans xmlns=*"http://www.springframework.org/schema/beans"* xmlns:xsi[=*http://www.w3.org/2001/XMLSchema-instance*](http://www.w3.org/2001/XMLSchema-instance)  xsi:schemaLocation=*"http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"*>  <bean id=*"bean1"* class=*"com.phung.Bean1"*>  *<!-- cấu hình cho bean ở đây -->*  </bean>    <bean id=*"bean2"* class=*"com.phung.Bean2"*>  *<!-- cấu hình cho bean ở đây -->*  </bean>  </beans> |

Thuộc tính id chỉ ra định danh của bean đó, thuộc tính class chỉ ra class của bean này (lưu ý sử dụng tên đầy đủ của class bao gồm cả phần package). Giá trị của thuộc tính id sẽ được các bean khác sử dụng nếu cần thiết.

*2.2.5.2.1 Khởi tạo container*

Việc khởi tạo một Spring container đơn giản như việc chúng ta khởi tạo một đối tượng, cần chỉ rõ đường dẫn của file XML cấu hình trong khi khởi tạo một container. Ví dụ dưới đây sẽ cho chúng ta thấy rõ hơn:

ApplicationContext context = **new** ClassPathXmlApplicationContext(**new** String[] {*"*services.xml*"*,

*"*daos.xml*"*} );

Trong đó file config “services.xml” như sau:

|  |
| --- |
| <?xml version=”1.0” encoding=”UTF-8”?>  <beans xmlns=*"http://www.springframework.org/schema/beans"* xmlns:xsi[=*http://www.w3.org/2001/XMLSchema-instance*](http://www.w3.org/2001/XMLSchema-instance)  xsi:schemaLocation=*"http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"*>  <bean id=*"service"* class=*"com.phung.ServicesImpl"*>  <property name=*"*accountDao” ref=”accountDao*"* />  <property name=*"*personDao*"* ref=*"*personDao*"* />  *<!-- các cấu hình khác ở đây -->*  </bean>  </beans> |

Và file config “daos.xml” như sau:

|  |
| --- |
| <?xml version=”1.0” encoding=”UTF-8”?>  <beans xmlns=*"http://www.springframework.org/schema/beans"* xmlns:xsi[=*http://www.w3.org/2001/XMLSchema-instance*](http://www.w3.org/2001/XMLSchema-instance)  xsi:schemaLocation=*"http://www.springframework.org/schema/context*  *http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"*>  <bean id=*"*accountDao*"* class=*"com.phung.AccountDao"*>  </bean>    <bean id=*"*personDao*"* class=*"com.phung.PersonDao"*>  </bean>  </beans> |

Ở ví dụ trên class ***ServicesIpml*** được cấu hình trong file *“services.xml”* và có hai thuộc tính truy cập dữ liệu thuộc hai lớp là ***AccountDao*** và ***PersonDao***. Thẻ

*<property />* dùng để tham chiếu đến một bean khác, trong đó thuộc tính *name* là tên của thuộc tính trong class ***ServicesIpml*** và giá trị của *ref* là *id* của bean cần tham chiếu đến. Thường thì mỗi file cấu hình XML sẽ cấu hình cho một tầng hoặc một module trong ứng dụng. Việc chia nhỏ như thế giúp dễ dàng kiểm soát hơn.

*2.2.5.2.2 Sử dụng container*

Ví dụ dưới đây sẽ chỉ ra cách đọc các định nghĩa của bean và truy cập đến chúng.

|  |
| --- |
| ApplicationContext context =  **new** ClassPathApplicationContext(**new** String[] {*"*services.xml*"*, *"*daos.xml*"*} );  *//nhận về đối tượng đã được config*  ServicesImpl services = (ServicesImpl)context.getBean(*"service"*);    //sử dụng đối tượng  List<String> lists= services.getListPerson(); |

2.2.5.3 Dependency Injection

*2.2.5.3.1 Constructor – based dependency injection*

Constructor – based DI là phương pháp “tiêm sự phuộc” cho các đối tượng bằng cách gọi hàm tạo (constructor method) với các tham số chính là các “sự phụ thuộc” (dependentcies) của đối tượng. Ví dụ sau đây sẽ đưa ra một class chỉ có thể tiêm sự phụ thuộc bằng phương pháp này.

|  |
| --- |
| **public** **class** MessageSender {    **private** Message message;    **public** MessageSender(Message message) { **this**.message = message;  }  } **public** **class** Message {    **public** Message() {    }  } |

Khi đó để tiêm sự phụ thuộc cho lớp MessageSender cần khai báo các bean như sau:

|  |
| --- |
| <bean id=*"message"* class=*"com.phung.Message"* />    <bean id=*"sender"* class=*"com.phung.MessageSender"*>  <constructor-arg ref=*"message"* /> </bean> |

Thẻ <constructor-arg /> trong ví dụ này sẽ cho Spring container biết đây là một bean mà các dependentcy sẽ được tiêm vào bằng cách gọi hàm tạo của lớp này. Đối với các hàm tạo với nhiều đối số thì thứ tự của các thẻ <constructor-arg /> sẽ giống với thứ tự các đối số của hàm, giống như ví dụ dưới đây

|  |
| --- |
| **public** **class** ClassA {    **private** ClassB b;  **private** ClassC c;    **public** ClassA(ClassB b, ClassC c) {  **this**.b = b;  **this**.c = c;  }  } |

|  |
| --- |
| <bean id=*"a"* class=*"com.phung.ClassA"*>  <constructor-arg ref=*"b"* />  <constructor-arg ref=*"c"* /> </bean> |

<bean id=*"b"* class=*"com.phung.ClassB"* />

<bean id=*"c"* class=*"com.phung.ClassC"* />

Ở các ví dụ trên các dependentcy là những kiểu dữ liệu do người dùng định nghĩa nên thẻ <constructor-arg /> sử dụng thuộc tính *ref* để tham chiếu đến bean khác. Riêng đối với các kiểu dữ liệu nguyên thủy (int, boolean, string,…) thì khai báo bean đơn giản hơn.

|  |
| --- |
| **public** **class** ClassZ {    **private** **int** b;  **private** String c;  **private** **boolean** d;    **public** ClassA(ClassB b, ClassC c, ClassD d) {  **this**.b = b;  **this**.c = c;  **this**.d = d;  }  } |

Các khai báo bean dưới đây là tương tự nhau.

|  |
| --- |
| <bean id=*"z"* class=*"com.phung.Z"*>  <constructor-arg value=*"123"* />  <constructor-arg value=*"abc"* />  <constructor-arg value=*"true"* />  </bean>    <!-- hoặc -->  <bean id=*"z"* class=*"com.phung.ClassZ"*>  <constructor-arg value=*"123"* type=*"int"*/>  <constructor-arg value=*"abc"* type=*"java.lang.String"*/>  <constructor-arg value=*"true"* type=*"boolean"*/>  </bean>    <!-- hoặc -->  <bean id=*"z"* class=*"com.phung.ClassZ"*>  <constructor-arg index=*"0"* value=*"123"* />  <constructor-arg index=*"1"* value=*"abc"* />  <constructor-arg index=*"2"* value=*"true"* />  </bean>    <!-- hoặc -->  <bean id=*"z"* class=*"com.phung.ClassZ"*>  <constructor-arg name=*"b"* value=*"123"* />  <constructor-arg name=*"c"* value=*"abc"* />  <constructor-arg name=*"d"* value=*"true"* /> </bean> |

*2.2.5.3.2 Setter – based dependency injection*

Thông thường với mỗi thuộc tính XXX trong một lớp, sẽ có các phương thức getXXX() và setXXX() tương ứng với nó. Qua phương thức setXXX(), thuộc tính XXX sẽ được gán cho một giá trị nào đó (giá trị này có thể thuộc kiểu primitive hay kiểu tham chiếu). Spring cũng cung cấp cho chúng ta phương pháp để gán giá trị đến một thuộc tính qua phương thức setter của nó.

Setter – based DI là phương pháp tiêm sự phuộc cho các đối tượng bằng cách gọi các setter của một class sau khi hàm dựng mặc định (không đối số) được gọi để khởi tạo bean. Ví dụ dưới đây sẽ đưa ra một class chỉ có thể tiêm sự phụ thuộc bằng các setter.

|  |
| --- |
| **public** **class** Person {    **private** String fullName;  **private** **int** age;    **public** **void** setFullName(String fullName) { **this**.fullName = fullName;  }    **public** **void** setAge(**int** age) {  **this**.age = age;  }  } |

Khi đó để tiêm sự phụ thuộc cho lớp Person cần khai báo các bean như sau:

|  |
| --- |
| <bean id=*"person"* class=*"com.phung.Person"*>  <property name=*"fullName"*>  <value>Nguyen Thien An</value>  </property>  <property name=*"age"*>  <value>22</value>  </property>  </bean>    <!-- hoặc ngắn gọn hơn -->  <bean id=*"person"* class=*"com.phung.Person"*>  <property name=*"fullName"* value=*"Bùi Minh Phụng"* />  <property name=*"age"* value=*"22"* /> </bean> |

Thẻ <property> được dùng để biểu diễn cho một phương thức setter. Thuộc tính *name* của thẻ <property /> chỉ định thuộc tính mà setter của nó sẽ được gọi để gán giá trị được chỉ định trong thẻ <value /> (hoặc thuộc tính *value* như cách khai báo bean thứ 2). Thuộc tính *name* phải khớp với phần đuôi của phương thức setter. Ví dụ ta có phương thức là *setAField()*, thì lúc khai báo sẽ tương ứng là *name=”aField”*.

Chúng ta có thể kết hợp cả 2 phương pháp DI này trong khai báo một bean. Thông thường Constructor – based DI được sử dụng cho những dependency bắt buộc và Setter – based DI cho những dependency tùy chọn.

Dễ dàng nhận thấy rằng tất cả các class ở những ví dụ trên đều là những POJOs (những class Java bình thường), chúng hoàn toàn không kế thừa từ bất cứ một class hoặc thực thi một interface đặc biệt nào, đều này cho thấy được sức mạnh của Spring trong việc xây dựng một ứng dụng từ các POJOs là nhờ vào DI và IoC container.

Spring container xử lý việc tiêm sự phụ thuộc cho các bean như sau:

* Spring container ApplicationContext được tạo ra và khởi tạo những siêu dữ liệu cấu hình (configuration metadata) cho tất cả các bean. Các configuration metadata có thể là XML, code Java hoặc các annotaion.
* Đối với từng bean các dependency của nó được thể hiện dưới dạng là các property, các đối số của hàm tạo. Những dependency này sẽ phải được cung cấp cho bean khi bean thực sự được khởi tạo.
* Đối với từng property hoặc các đối số của hàm được gán giá trị để thiết lập cho bean hoặc được tham chiếu đến một bean khác trong container.
* Giá trị của từng property và đối số của hàm tạo sẽ được chuyển thành kiểu dữ liệu phù hợp, Spring sẽ chuyển chúng từ kiểu String sang các kiểu dữ liệu khác như int, long, String, boolean,…

Spring container sẽ kiểm tra tính hợp lệ của các thông tin cấu hình cho mỗi bean khi container được tạo. Tuy nhiên các thuộc tính của bean sẽ không được thiết lập cho đến khi bean thật sự được tạo ra và một bean chỉ được tạo khi có một yêu cầu đến nó.

*2.2.5.3.3 Autowiring*

Khi một bean A cần tham chiếu đến bean B (B là một dependency của A) chúng ta cần phải cấu hình cho bean A để có thể “tiêm” B vào thông qua Constructor – based DI hoặc Setter – based DI. Tuy nhiên nếu chúng ta cấu hình cho bean A autowire thì Spring container có thể thực hiện việc này một cách “tự động” bằng cách Spring container sẽ kiểm tra trong ApplicationContext để tìm ra B và “tiêm” cho A.

Autowiring thực sự hữu ích khi phát triển một ứng dụng mà việc chỉnh sửa mã nguồn là thường xuyên [6, tr52], ví dụ khi thêm một thuộc tính mới cho một class, nếu sử dụng autowire thì cấu hình bean của class này không cần phải thay đổi.

Để cấu hình cho một bean có thể autowire chỉ cần thêm thuộc tính *autowire* trong thẻ <bean />. Autowire có 4 loại vì vậy cần phải chỉ rõ loại của autowire đối với mỗi bean. Dưới đây là 4 loại của autowire:

* Autowire by name.Spring sẽ tìm kiếm bean có tên giống như tên của property cần autowire. Ví dụ nếu bean A được chỉ định là autowire by name có property tên *master* (class A cần phải có phương thức setMasster(…)) , Spring sẽ tìm một bean có tên (id) là *master* và tiêm nó vào bean A.
* Autowire by type.Autowire by type cho phép một property được autowire nếu có một bean có kiểu (class) là kiểu của property này. Nếu có nhiều hơn một bean thì một exception được ném ra, ngược lại nếu không tồn tại bất cứ bean nào phù hợp thì không có exception nào được ném ra và property này không được thiết lập (set) giá trị.
* Autowire by constructor. Tương tự như autowire by type, tuy nhiên autowire by constructor chỉ áp dụng cho các đối số của hàm tạo. Nếu không có chính xác một bean có kiểu là kiểu của các đối số trong hàm tạo thì một exception sẽ được ném ra.
* No autowire. Không autowire, đây là thiết lập mặc định của các bean. Các bean tham chiếu đến nhau thông qua thuộc tính *ref.* No autowire được khuyến cáo nên sử dụng, bởi vì việc chỉ định các dependency một cách rõ ràng sẽ tốt hơn và ở một mức độ nào đó nó được xem như tài liệu cho cấu trúc của hệ thống.

Lưu ý là các dependency được chỉ định một cách tường minh sẽ ghi đè lên lên autowire. Đối với các property có kiểu dữ liệu nguyên thủy như String, Classes… thì không thể autowire.

## 2.3 Angularjs

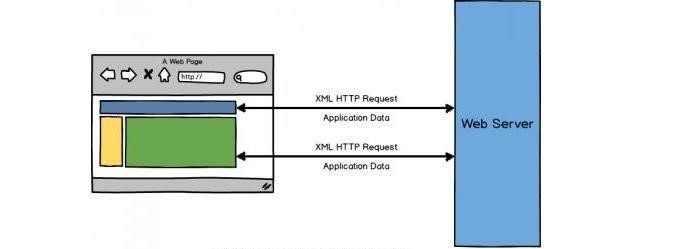
### 2.3.1 Tổng quan về AngularJS

Công nghệ HTML hỗ trợ tốt cho các trang web tĩnh. Khi chúng ta xây dựng 1 trang web với PHP, Node/Express, hay Ruby thì nó cũng chỉ là một trang web tĩnh với nội dung được thay đổi khi chúng ta gửi request về máy chủ, máy chủ sẽ render một trang với nội dung tương ứng. Tuy nhiên mọi thứ đã thay đổi nhiều từ sự phát triển của HTML5, nhất là khi có sự hổ trợ từ những tập đoàn công nghệ lớn như Google, Yahoo, Facebook, và sự tập hợp đông đảo của cộng đồng mã nguồn mở. Ngoài ra sự thành công của jQuery đã khiến JavaScript được nhiều người yêu thích vì tính đơn giản và dễ sử dụng. Việc phát triển một website sử dụng AJAX thì không khó, chúng ta có thể dùng jQuery để làm việc này với $.ajax tuy nhiên làm thế nào để xây dựng một phần mềm có thể mở rộng, dễ test, nâng cấp và bảo trì thì không hề đơn giản vì bản thân JavaScript không được thiết kế ngay từ đầu để làm những việc này. Do đó sự ra đời của những framework hỗ trợ lập trình viên xây dựng ứng dụng web 1 cách có hệ thống đã ra đời như Sencha, Ember, Knockout, Backbone, CanJS và AngularJS.

AngularJS là framework JavaScript được bắt đầu phát triển từ năm 2009, do lập trình viên Misko Hevery tại Google viết ra như là một dự án kiểu “viết cho vui”. Misko và nhóm lúc này đang tham gia vào 1 dự án của Google tên là Google Feedback. Với AngularJS, Misko đã rút ngắn số dòng code front-end từ 17000 dòng còn chỉ khoảng 1500. Với sự thành công đó, đội ngũ của dự án Google Feedback quyết định phát triển AngularJS theo hướng mã nguồn mở. Theo thông số từ Github hiện tại dự án AngularJS đang có gần 11.000 người theo dõi và hơn 2.000 lượt fork.

Ngày nay các ứng dụng web hiện đại thường được xây dựng như một Single Page Application (SPA) và AngularJS là một framework tuyệt vời để xây dựng 1 SPA.

Một Single Page Application là một ứng dụng web hiển thị vừa vặn trên một mặt của trang với mục đích giúp người dùng có trải nghiệm giống như đang dùng ứng dụng trên desktop và làm tăng hiệu suất của ứng dụng [3, tr7]. Hay nói một cách khác là ứng dụng chạy bên trong trình duyệt mà không yêu cầu phải tải lại toàn bộ trang web mỗi lần sử dụng.

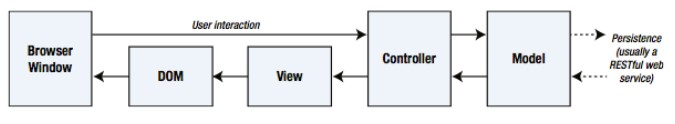


*Hình 2.18: Single Page Application*

2.3.2 Các đặc trưng của AngularJSDưới đây là những đặc trưng nổi bật của AngularJS: kiến trúc MVC, Two-way binding, Templates, Expressions, Modules, Scopes, Dependency injection, Directives, Routing, Services, Filters, Form validation, Testing.

#### 2.3.2.1 Kiến trúc MVC trong AngularJS

MVC (Model - View - Controller) là một kiến trúc phần mềm hay mô hình thiết kế được sử dụng trong kỹ thuật phần mềm. Nó giúp cho các nhà phát triển tách ứng dụng của họ ra 3 thành phần khác nhau Model (xử lý, truy xuất database), View (giao diện) và Controller (điều hướng yêu cầu từ người dùng). Mỗi thành phần có một nhiệm vụ riêng biệt và độc lập với các thành phần khác. MVC thể hiện tính chuyên nghiệp trong lập trình, phân tích thiết kế. Do được chia thành các thành phần độc lập nên giúp phát triển ứng dụng nhanh, đơn giản, dễ nâng cấp, bảo trì.



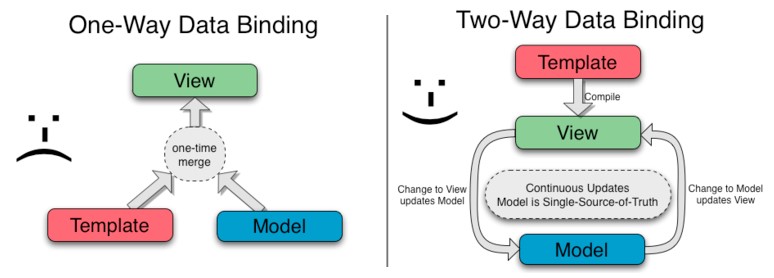
*Hình 2.19: Kiến trúc MVC trong AngularJS*

Đối với Angular, View sẽ là DOM, Controller là các lớp JavaScript, còn Model sẽ là dữ liệu được lưu ở thuộc tính của các đối tượng trong JavaScript.

#### 2.3.2.2 Two-way binding Two-way binding cùng với Directive là hai trong những đặc trưng nổi bật mang lại sức mạnh cho AngularJS.

Để hiểu rõ hơn về two-way binding ta sẽ xem xét one-way binding trước. Khi làm việc với các mô hình MVC framework khác chúng ta sẽ rơi vào trường hợp model và view chỉ được ghép với nhau một lần duy nhất, vì vậy mọi sự cập nhật dữ liệu từ một trong hai lớp đó đều không liên quan gì đến lớp còn lại. Hệ quả là developer phải viết thêm phần “đồng bộ hóa” giữa hai lớp này. Đó chính là one-way binding.

Tuy nhiên AngularJS có thể thực hiện được cả hai chiều, đó chính là two-way binding. Hay nói một cách khác two-way binding trong AngularJS là tính năng đồng bộ hóa giữa model và view. Quá trình đồng bộ luôn xảy ra giữa model và view. Mọi sự thay đổi dữ liệu trong lớp này, ngay lập tức được cập nhật vào lớp kia [7, 11]. Chúng ta chỉ cần “bind” một lần, và mọi thứ hoạt động trơn tru. Hình 2-19 sẽ cho thấy rõ hơn về one-way binding và two-way binding.



*Hình 2.20: One-way binding và Two-way binding*

Cũng vì dữ liệu mà view hiển thị thực chất là sự ánh xạ từ model, nên controller cũng hoàn toàn không cần các thao tác data binding phức tạp và phụ thuộc vào cấu trúc DOM của view, kết quả là controller sẽ không quan tâm xem view của chúng ta là gì. Và dĩ nhiên, logic giữa controller và view được bóc tách rất rõ ràng. Code của chúng ta sẽ rất dễ để debug, viết unit-test.

#### 2.3.2.3 Directives

Tính năng mới này thực sự là đặc điểm tạo nên một khoảng cách lớn giữa Angular và bất kỳ framework JavaScript khác. Directives đề cập đến một tính năng để mở rộng HTML, việc dùng directive sẽ giảm thiểu được số lượng thẻ HTML, code HTML nhìn sẽ gọn gàng và sáng sủa hơn.

Mặc dù HTML có thể có nhiều thẻ, nhưng nó không đủ mạnh để tạo ra một ứng dụng web hiện đại. Với directive, chúng ta có thể tạo ra những thẻ HTML mới hoặc các thuộc tính mới cho tất cả các thẻ trong HTML để trình duyệt hiểu nó có nghĩa là gì và nên làm gì.

Ví dụ với Angular chúng ta có thể tạo ra thẻ như sau *<draggable> drag me </ draggable >*. Các văn bản bên trong tag *<draggable></ draggable>* bây giờ sẽ trở thành draggable trong trình duyệt, sau đó chỉ cần viết một định nghĩa cho directive draggable này trong ứng dụng Angular của chúng ta là xong. Đây là một ví dụ đơn giản của Angular, nó có thể làm rất nhiều thứ hữu ích hơn.

Directtive không chỉ giới hạn một phần tử HTML mà còn có thể là thuộc tính, lớp hoặc các chú thích HTML. AngularJS cung cấp cho chúng ta 3 loại directive: - Directive dạng element (một thẻ HTML) viết tắt là E:

|  |
| --- |
| <my-directive>...</my-directive> |

* Directive dạng attribute (thuộc tính của một thẻ HTML) viết tắt là A, dạng này là mặc định.

|  |
| --- |
| <div my-directive>...</div> |

* Directive dạng class(class CSS) viết tắt là C.

|  |
| --- |
| <div class=*"my-directive"*>...</div> |

#### 2.3.2.4 Templates

Trong quá trình xây dựng hệ thống lớn, file HTML trở nên rất phức tạp, để giải quyết vấn đề này ta cần chia thành nhiều phần khác nhau, AngularJS cung cấp cho chúng ta một giải pháp hữu ích đó là template.

Angular template là một đặc tả dạng declarative, cùng với thông tin từ model và controller, template được render thành một view mà để hiển thị trên trình duyệt. Template bao gồm các static DOM chứa HTML, CSS và các thành phần, thuộc tính riêng của Angular. Các thành phần Angular và các thuộc tính giúp Angular thêm các hành vi và biến đổi template thành dynamic view DOM.

Trong Angular, chúng ta có 2 cách để tạo một template:

* ***Dùng file script:*** Chúng ta có thể tích hợp template vào file HTML hiện hành thông qua thẻ script với type là text/ng-template

<script type=*"text/ng-template"* id=*"message.html"*>

<h3>Hello, {{name}}!</h3>

</script>

* ***Dùng file ngoài:*** Chúng ta cũng có thể tạo một file html với nội dung sau:

|  |
| --- |
| message.html |
| <web-app>  <display-name>Music Manager</display-name> |

* ***Sử dụng template:*** Có nhiều cách để sử dụng template, tuy nhiên trong AngularJS có 2 cách thông dụng để dùng template đó là ng-include và ng-route. Ví dụ dưới đây sẽ dùng ng-include để sử dụng template:

|  |
| --- |
| index.html |
| <!DOCTYPE html>  <html>  <head>  <script src=*"http://code.angularjs.org/1.2.12/angular.min.js"*> </script>  </head>  <body>    <div ng-controller=*"ExampleCtrl"*>  <div ng-include src=*"message.html"*></div>  </div>    </body>  </html> |

Tuy nhiên cách sử dụng ng-include chỉ thích hợp trong các ứng dụng nhỏ, đối với các ứng dụng lớn ng-route (được trình bày trong phần xây dựng ứng dụng) được sử dụng nhiều hơn.

#### 2.3.2.5 Dependency Injection

Dependency Injection làm cho ứng dụng Angular dễ dàng hơn để phát triển, có thể sắp xếp mã (code) thành các module. Bằng cách này, chúng ta có thể tạo ra các thành phần tái sử dụng mà có thể được gọi về sau, bất cứ khi nào cần thiết. Ý tưởng này là đặc biệt tốt khi ứng dụng lớn và cần phải được mở rộng, dễ dàng bảo trì và testing. Dependency Injection là một cách để ghép các đoạn code với nhau, điều này đòi hỏi phải phân chia code theo các chức năng riêng biệt như service, controller, hoặc provider.

#### 2.3.2.6 Services

Service trong AngularJS là một object hoặc một function phụ trách một tác vụ nào đó. Việc viết service chỉ là việc gom các đoạn xử lý logic vào object hoặc function để dễ quản lý hơn. Cũng giống như mô hình MVC tách phần xử lý, điều hướng và hiển thị riêng biệt.

Trong AngularJS service sẽ chứa tất cả các phần xử lý, controller nhận yêu cầu và gọi các service cần thiết để xử lý, model lưu trữ dữ liệu từ các control, view hiển thị dữ liệu, route điều hướng request tới controller.

AngularJS cũng xây dựng rất nhiều service vô cùng tiện lợi như: $http, $scope,

$window,…

#### 2.3.2.7 Expression

Expression được sử dụng trong tất cả các phần của AngularJs, expression được bao bởi kí hiệu {{ }}. Một biến được khởi tạo từ controller thông qua $scope cũng được coi là một expression.

Expression được xử lý bởi $parse service.

Ví dụ: <body> 2 + 2 = {{2 + 2}} </body>. Angular sẽ tính toán giá trị trong biểu thức và xuất ra màn hình kết quả “2 + 2 = 4”.

#### 2.3.2.8 Fillters

Trong AngularJS, một filter cung cấp một định dạng dữ liệu để hiển thị tới người dùng. AngularJS cung cấp một số filter được xây dựng sẵn như: lowercase, date, number, currency, limitTo, orderBy… Một số fillter như:

{{ "chuỗi kí tự viết thường" | uppercase }}

{{ {name: "An", age: 22} | json }}

{{ 1504375948024 | date }}

{{ 1504375948024 | date:"MM/dd/yyyy @ h:mma" }}

Kết quả:

CHUỖI KÍ TỰ VIẾT THƯỜNG

{ "name": "Phung", "age": 22 }

Feb 23, 2020

01/23/2020 @ 5:39PM

#### 2.3.2.9 Scope

Scope là đối tượng tham chiếu tới model được sử dụng trong controller. Là thành phần gắn kết giữa view và controller. Scope cung cấp các APIs để theo dõi các thay đổi của model (services $watch) và các APIs để truyền bất kỳ thay đổi nào của model tới view (services $apply). Scope giúp tự động đồng bộ dữ liệu giữa model và view.

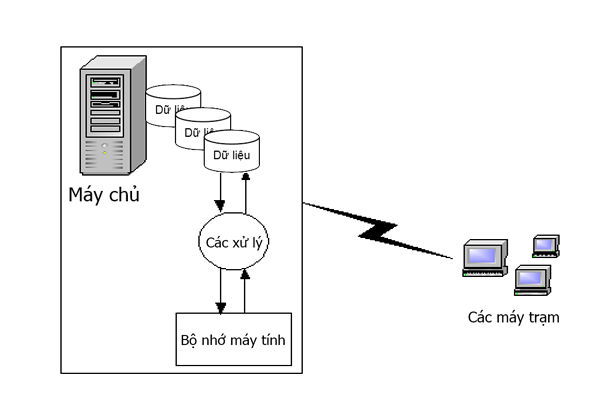
## 2.4 Oracle

### ****2.4.1 Lịch sử ra đời****

Năm 1977 một chàng trai trẻ có tên là Lawrence Ellision sau khi đã đọc và tìm hiểu về mô hình quan hệ (relational model) của công ty máy tính IBM đã quyết định thành lập công ty Oracle. Sau đó hai năm (1979), công ty này đã phát hành sản phẩm đầu tay của mình chính là CSDL Oracle phiên bản 2.0, nó không những là một sản phẩm CSDL quan hệ mà còn tích hợp ngôn ngữ truy vấn có cấu trúc SQL. Năm 1983 phiên bản 3 được phát hành với các thay đổi trong ngôn ngữ SQL, mở rộng hiệu suất và các cải tiến. Năm 1985 phiên bản 5 ra đời đánh dấu một sự kiện quan trọng vì nó được áp dụng theo công nghệ khách chủ. Năm 1997 Oracle giới thiệu phiên bản 8, tiếp đến là phiên bản 8i (ver 8.1.5) hỗ trợ nhiều tính năng mới và việc phát triển ứng dụng CSDL trên Internet. Và với phiên bản [Oracle 12c](https://csc.edu.vn/lap-trinh-va-csdl/Introduction-to-SQL-12C_90) ra đời năm 2014 với nhiều tính năng nổi bật như hỗ trợ điện toán đám mây, dữ liệu lớn (big data),… , Oracle đã có thể cung cấp đa dạng các giải pháp lưu trữ và xử lý dữ liệu chuyên nghiệp cho các khách hàng theo nhiều hướng ứng dụng khác nhau.

### ****2.4.2 Kiến trúc cơ bản của Oracle****

Oracle được kiến trúc theo mô hình 3 lớp: lớp dữ liệu, lớp xử lý và lớp bộ nhớ.



***Hình 2.21: Mô hình 3 lớp***

2.4.2.1 Lớp dữ liệu (File systems)

Lớp dữ liệu bao gồm các tập tin dữ liệu được tổ chức lưu trữ tại các đĩa cứng của một hoặc nhiều máy chủ khác nhau (tính phân tán của CSDL Oracle). Khi có các yêu cầu truy xuất từ phía các máy trạm, các thành phần bên trong dữ liệu đã được các xử lý bên dưới nạp đúng phần dữ liệu cần truy xuất trước đó từ đĩa cứng vào bên trong bộ nhớ của máy chủ. Điều này sẽ giúp cho tốc độ truy xuất được hiệu quả hơn.

Thông thường bên trong một CSDL Oracle sẽ có nhiều loại tập tin dữ liệu khác nhau. Tiêu biểu là một số loại tập tin sau:

* Tập tin tham số (init file): là tập tin chứa các thông tin cơ sở liên quan đến tên CSDL, vị trí của các tập tin điều khiển, các tham số ...
* Tập tin điều khiển (control file): là tập tin chứa các thông tin liên quan đến ngày giờ tạo CSDL, vị trí của CSDL...
* Tập tin dữ liệu (database file): là tập tin chứa dữ liệu thật sự của CSDL.
* Tập tin lưu vết (redo log file): là tập tin chứa các hành động cập nhật dữ liệu (thêm, sửa, huỷ) bên trong các giao tác.

#### ****2.4.2.2 Lớp xử lý bên dưới (Background processes)****

Lớp xử lý bên dưới tại máy chủ sẽ đảm bảo cho mối quan hệ giữa phần CSDL vật lý và phần hiển thị trong bộ nhớ được khớp nhau. Cũng giống như lớp dữ liệu, các xử lý được chia ra làm nhiều loại khác nhau. Một số xử lý tiêu biểu như sau:

Xử lý ghi vào CSDL (database writer): xử lý này được tự động thực hiện khi dữ liệu trên vùng đệm của bộ nhớ máy tính bị đầy, khi đó hệ thống sẽ đọc và ghi xuống CSDL các dòng dữ liệu bị thay đổi và sau đó giải phóng vùng đệm bộ nhớ máy tính.

Xử lý ghi vào tập tin log (log writer): xử lý này được tự động thực hiện để ghi nhận xuống tập tin log các thông tin dữ liệu bên trong quá trình thực hiện giao tác. Điều này sẽ làm an toàn dữ liệu hơn khi giao tác chưa kết thúc nhưng thông tin vẫn được lưu trữ.

#### ****2.4.2.3 Lớp bộ nhớ (Memory)****

Lớp bộ nhớ bao gồm nhiều thành phần khác nhau được tổ chức lưu trữ trên vùng đệm bộ nhớ của máy tính nhằm tăng tốc độ xử lý trong Oracle, khái niệm này còn được biết đến với một tên khác là vùng toàn cục hệ thống (system global area). Một số vùng đệm tiêu biểu như sau:

* Vùng đệm lưu trữ CSDL (database buffer cache): là nơi lưu trữ các thông tin dữ liệu đã được đọc từ các tập tin dữ liệu trên đĩa cứng.
* Vùng đệm lưu trữ các thông tin chung thường dùng (dictionary cache): là nơi lưu trữ các bảng hệ thống của CSDL Oracle, bản thân Oracle thường xuyên sử dụng các bảng này do đó bên trong hệ thống Oracle đã quyết định đặt chúng trên bộ nhớ nhằm tránh đi việc đọc đĩa thường xuyên.
* Vùng đệm lưu trữ lệnh SQL (SQL area): là nơi lưu trữ các lệnh SQL được thực hiện tại máy chủ mà các máy trạm truyền đến. Tại đây các lệnh sẽ được phân tích cú pháp, kiểm tra tính đúng sai trước khi thực hiện và trả về kết quả cho các máy trạm.

### ****2.4.3 Giới thiệu Oracle Server****

Oracle Server là một hệ quản trị CSDL cung cấp nhiều công cụ trong việc quản lý thông tin. Oracle Server là một tập hợp các tập tin vật lý, tiến trình và những cấu trúc bộ nhớ dùng để thực hiện các yêu cầu truy xuất dữ liệu, thực hiện các thao tác phục hồi và lưu trữ dữ liệu khi cần thiết.

Oracle Server có thể chạy trên một hay nhiều máy tính với những mô hình khác nhau:

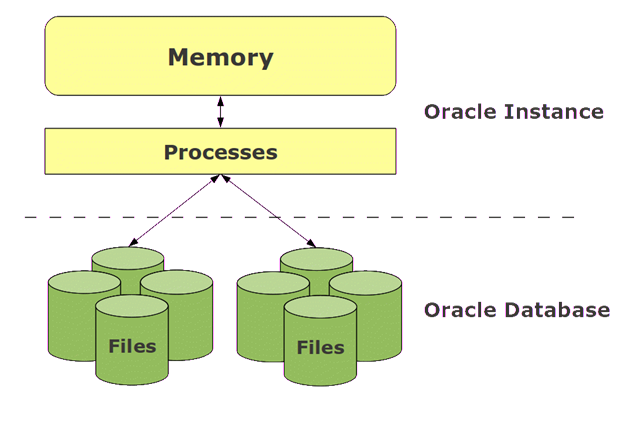
Client – Application Server – Server: đây là mô hình 3 lớp, người dùng truy cập từ máy đơn (Client) vào máy chứa dữ liệu (Server) thông qua một trạm trung gian gọi là Application Server, dùng để thực hiện việc xử lý các yêu cầu của ứng dụng khi cần truy xuất dữ liệu.

Client – Server: thông qua hệ thống mạng, người dùng sẽ dùng máy tính (Client) để truy xuất CSDL được đặt trên một máy tính khác (Server) trong hệ thống mạng, truy xuất dữ liệu theo dạng này được gọi là mô hình 2 lớp.

Host – Base: người dùng truy xuất dữ liệu ngay tại máy dùng làm Server lưu trữ CSDL.

**2.4.4 Những thành phần chính của Oracle Server**

Oracle Server gồm có một Oracle Instance và một Oracle Database.  
***Hình 2.22: Những thành phần chính của Oracle Server***



#### ****2.4.4.1 Oracle Instance****

Là thành phần liên kết giữa người dùng và thông tin trong Oracle Database. Một Oracle Instance chỉ được mở cho duy nhất một Oracle Database cần truy xuất.

Oracle Instance được tạo nên bởi sự kết hợp của hai thành phần là cấu trúc bộ nhớ và các tiến trình nền (Background Process) dùng để quản lý CSDL.

Vùng nhớ của Oracle Instance gọi là SGA (System Global Area) có vai trò lưu trữ thông tin của Oracle Database và được chia sẻ cho những người đang truy xuất vào cơ sở dữ liệu.

Tiến trình trong Oracle có 2 loại: tiến trình của người dùng (User Process) và tiến trình của Oracle (Oracle Process). User Process được hình thành khi người dùng sử dụng công cụ hay ứng dụng kết nối vào Oracle Instance.

Khi đó, trên máy tính chứa Oracle Server sẽ phát sinh một tiến trình gọi là Server Process dùng để giao tiếp giữa User Process và Oracle Instance. Những xử lý bên trong Oracle Instance sẽ do các tiến trình nền (Background Process) thực hiện.

#### ****2.4.4.2 Oracle Database****

Oracle Database dùng để lưu trữ và cung cấp thông tin cho người dùng. Việc quản lý dữ liệu trong Oracle Database thông qua cấu trúc lưu  trữ logic và vật lý.

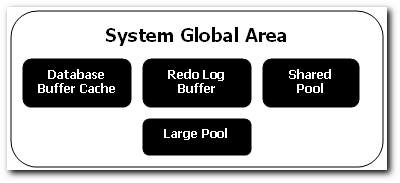
Cấu trúc lưu trữ logic là những thành phần đặc trưng của Oracle Server, cấu trúc vật lý là những tập tin hệ thống tồn tại trong hệ điều hành, nơi Oracle Server đang chạy.

### ****2.4.4 Cấu trúc vùng nhớ – Memory Structure****

Oracle sử dụng cấu trúc vùng nhớ để lưu trữ thông tin trong hệ thống trong khi đang thực thi, như là mã nguồn của một chương trình đang chạy, thông tin về kết nối của các phiên làm việc, lưu vết dữ liệu vật lý trong quá trình xử lý,…

Cấu trúc vùng nhớ của Oracle bao gồm SGA và PGA. SGA (System Global Area) là một thành phần cơ bản trong Oracle Instance và được cấp phát khi Instance khởi động. Thông tin lưu trữ trong SGA có thể được sử dụng bởi tất cả các Server Process và Background Process.

PGA (Program Global Area) được cấp phát cho User Process kết nối đến Oracle Server.



***Hình 2.23: SGA***

### ****2.4.5 System Global Area****

SGA được cấp phát tại vùng nhớ ảo của máy tính đang chạy Oracle. SGA gồm có một số cấu trúc vùng nhớ dùng để chứa dữ liệu và quản lý các thông tin về Oracle Instance.

SGA còn được biết như là một vùng nhớ chia sẻ, thông tin về Database có thể được truy xuất bởi các tiến trình đang làm việc trên Database đó. Nếu có nhiều người dùng kết nối với một Instance thì SGA của Instance sẽ chia sẻ cho những người dùng đó.

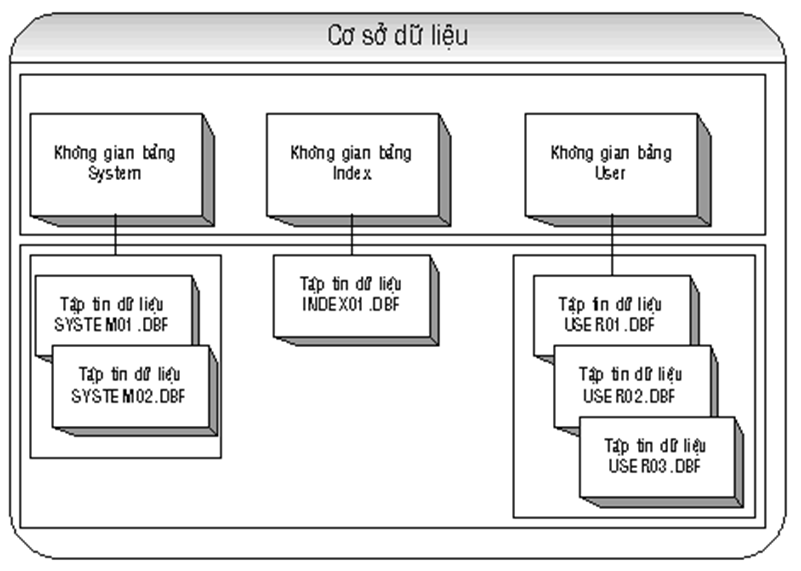
SGA bao gồm một số cấu trúc vùng nhớ chính như: Shared Pool, Database Buffer Cache, Redo Log Buffer. Ngoài ra, còn có hai cấu trúc vùng nhớ mà SGA có thể sử dụng hay bỏ qua là Large Pool và Java Pool. Kích thước vùng nhớ của SGA phụ thuộc vào kích thước của các thành phần tạo nên nó.

SGA có một đặc tính quan trọng là ta có thể thay đổi kích thước mà không cần phải ngừng Instance của SGA cần sửa đổi. Điều này có nghĩa là kích thước của các Database Buffer Cache, Shared Pool và Large Pool của SGA có thể được thay đổi trong khi Instance đang chạy.

### ****2**.4.6 Tổ chức dữ liệu trên CSDL Oracle**

Vai trò quan trọng nhất của Oracle Server là lưu trữ và cung cấp thông tin cho người dùng thông qua Oracle Database. Oracle Database quản lý thông tin lưu trữ thông qua hai thành phần lưu trữ vật lý và logic.

Thành phần lưu trữ vật lý là các tập tin. Thành phần lưu trữ logic là các không gian bảng(tablespace).



*Hình 2.24:* ***Tổ chức dữ liệu trên CSDL Oracle***

#### ****2.4.6.1 Thành phần vật lý****

Thành phần vật lý dùng để lưu trữ dữ liệu trong Oracle Database là tập hợp một số tập tin tồn tại trên máy tính cài đặt Oracle Server, các tập tin này sẽ được sử dụng khi Oracle Database đang được mở.

Dữ liệu trong Oracle Database sẽ được tổ chức lưu trữ và quản lý bởi ba loại tập tin sau:

* Data File.
* Redo Log File.
* Control File.

### ****2.4.7 Data File****

Mỗi một Oracle Database có 1 hay nhiều Data File, Data File dùng để chứa tất cả dữ liệu của một Database. Những thành phần Logic của Database như Table, Index được lưu trữ vật lý trong các Data File của Database.

Các đặc điểm của Data File:

* Một Data File dùng để lưu trữ dữ liệu của duy nhất một Database.
* Kích thước của các Data File có thể tự động tăng để theo kích thước của Database.
* Một hay nhiều Data File tạo thành một đơn vị Logic của vùng nhớ trong cơ sở trữ dữ liệu gọi là Tablespace, và một Data File chỉ quan hệ với duy nhất một Tablespace mà thôi.

Data File có thể chứa dữ liệu của nhiều Schema Object bên trong Tablespace, lưu trữ vật lý, và những Shema Object có thể được lưu trữ trên nhiều Data File khác nhau. Khi dữ liệu được lưu trữ trên nhiều đĩa thì ta không thể biết được Shema Object được lưu trữ trên Data File nào.

Data File cũng có trạng thái Online và Offline như Tablespace. Trạng thái của Data File thường thay đổi theo Tablespace mà nó liên kết.

Khi Database quản lý vùnng nhớ của Tablespace theo cách Locally Managed thì các Temporary Tablespace sẽ có những Temporary Data Files tương ứng, gọi tắt là TempFile. Những TempFile này không thể chuyển sang trạng thái chỉ đọc và không thể đổi tên của chúng.

Thông tin của TempFile được lưu trữ trong các View của Dictionary là  DBA\_TEMP\_FILES và V$TEMPFILE, không có trong DBA\_DATA\_FILES hay trong view V$DATAFILE.

### ****2.4.8 Redo Log File****

Mỗi một Oracle Database có 2 hay nhiều Redo Log File, tập hợp những File này trong một CSDL được gọi là Redo Log của CSDL đó. Một Redo Log tạo thành những Redo Entry, còn gọi là Redo Record.

Chức năng chính của Redo Log là dùng để ghi lại những thông tin trong CSDL bị thay đổi.

Trong trường hợp Database gặp sự cố trước khi dữ liệu thay đổi được ghi nhận chắc chắn xuống các Data File thì những thông tin trong Redo Log sẽ được sử dụng trong việc phục hồi dữ liệu về trạng thái trước khi dữ liệu thay đổi.

### ****2.4.9 Control File****

Mỗi một Oracle Database có 1 Control File, dùng để chứa tất cả các thông tin về cấu trúc vật lý của một Database như là: tên Database, nơi lưu trữ của Data File và Redo Log File, Time stamp tạo Database. Mỗi Control File chỉ dùng cho một Database duy nhất.

Khi một Instance của Oracale Database được khởi động, Control File của nó được sử dụng để xác định vị trí của các Data File và Redo Log File, các File này phải được mở trong quá trình làm việc với Database.

Nội dung của Control File sẽ được Oracle cập nhật tự động và liên tục khi cấu trúc lưu trữ vật lý của Database có thay đổi.

Nội dung của Control File chỉ có thể bị thay đổi bởi Oracle, bất kỳ DBA hay người dùng nào cũng không thể sửa nội dung Control File của Database.

Khi những thành phần vật l‎ý của Database như Data File, Redo Log File có sự thay đổi như thêm mới File, sửa tên của File hay xóa File ra khỏi Database thì Control File sẽ ghi nhận lại những thông tin đó. Những thông tin này được sử dụng:

* Trong quá trình mở một Database Oracle, tìm được những Data File và Redo Log File của Database.
* Trong trường hợp phục hồi dữ liệu, Oracle có thể tìm được những tập tin được yêu cầu.

Bên cạnh đó, Control File còn lưu thông tin về Checkpoint. Cứ mỗi 3 giây, Checkpoint Process (CKPT) sẽ ghi vào Control File vị trí Checkpoint trong Online Redo Log File. Thông tin này dùng trong quá trình phục hồi dữ liệu của Database, báo cho Database biết là không cần thực hiện việc phục hồi dữ liệu trong trường hợp dữ liệu được ghi chắc chắn vào các Data File.

Control File là một tập tin quan trọng để Oracle có thể mở một Database, do đó người quản trị cần thực hiện việc sao chép và lưu trữ các Control File trên nhiều ổ đĩa vật lý khác nhau để sử dụng trong các trường hợp bị hư hỏng.

### ****2.4.10 Thành phần Logic****

Dữ liệu trong Oracle Database được tổ chức và quản lý dựa vào những thành phần Logic, là những thành phần hỗ trợ DBA và người dùng trong việc lưu trữ và sử dụng thông tin trên Oracle Server.

Thành phần Logic trong Oracle Database bao gồm 2 đơn vị lưu trữ là Tablespace và các Schema Object.

#### ****2.4.10.1**** Tablespace

Một Database được chia thành nhiều đơn vị lưu trữ ở mức Logic, những đơn vị lưu trữ đó gọi là Tablespace.

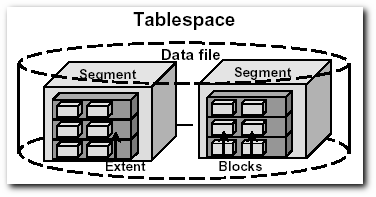
Các đặc điểm của Tablespace:

* Tablespace chỉ thuộc về một Database duy nhất.
* Một Tablespace bao gồm một hay nhiều tập tin vật lý dùng để lưu trữ dữ liệu, đó là Data File.

Trong Oracle Database luôn có một Tablespace do Oracle tạo ra trong quá trình tạo Database có tên là SYSTEM. Đây là vùng nhớ quan trọng của Database, dùng để lưu trữ tất cả những thông tin về Database như định nghĩa cấu trúc của các đối tượng, thông tin cấp phát và sử dụng vùng nhớ của các đối tượng, ....

Tablespace được tạo nên bởi sự kết hợp của một hay nhiều đơn vị lưu trữ Logic gọi là Segment, một Segment được chia thành nhiều Extent và trong Extent thì có nhiều DataBlock liên tục nhau

.



***Hình 2.25: Tổ chức của Tablespace***

#### ****2.4.10.2 Data Block****

Là đơn vị lưu trữ Logic nhỏ nhất được Oracle sử dụng trong việc đọc và ghi dữ liệu trong Oracle Database.

Kích thước của Data Block sẽ được xác định thông qua tham số DB\_BLOCK\_SIZE.

Data Block trong Oracle Database gồm có 3 vùng chính là: Header, Data và Free Space.

Header là vùng chứa thông tin chung của Block, như là địa chỉ của Block và loại Segment chứa Block là Data, Index, Rollback hay Temporary.

Vùng Data dùng lưu trữ dữ liệu của các dòng trong bảng. Các dòng của một bảng dữ liệu có thể nằm trên nhiều Block khác nhau.

Free Space là vùng nhớ nằm giữa Header và Data, vùng nhớ này giúp kích thước của Header và Data có thể tăng lên khi cần, trong trường hợp thêm hay sửa dữ liệu của bảng.

#### ****2.4.10.3 Extent****

Tập hợp nhiều Data Block liên tiếp nhau sẽ tạo thành một đơn vị lưu trữ Logic lớn hơn gọi là Extent. Số lượng Data Block của một Extent tùy thuộc vào kích thước được chỉ định cho Extent khi tạo đối tượng Table.

Khi người dùng tạo một Table thì Oracle sẽ tự động cấp phát cho Table một Data Segment, trong Data Segment sẽ có một Extent với số lượng Data Block tùy thuộc vào kích thước của Extent.

Nếu không gian lưu trữ trong các Data Block của một Extent trong Segment bị đầy thì Oracle sẽ tự động cấp phát thêm một Extent cho Segment. Kích thước của Extent mới cấp phát có thể lớn hơn hay bằng kích thước của Extent ban đầu.

Số Extent được cấp phát cho một Segment có thể được xác định thông qua các tham số khi tạo Table, như là MINEXTENT, MAXEXTENT.

#### 2.4.10.4 ****Segment****

Segment là tập hợp một số Extent dùng để chứa toàn bộ thông tin của cấu trúc lưu trữ Logic bên trong Tablespace, như là Table.

Oracle Database sử dụng 4 loại Segment khác nhau để lưu trữ thông tin:

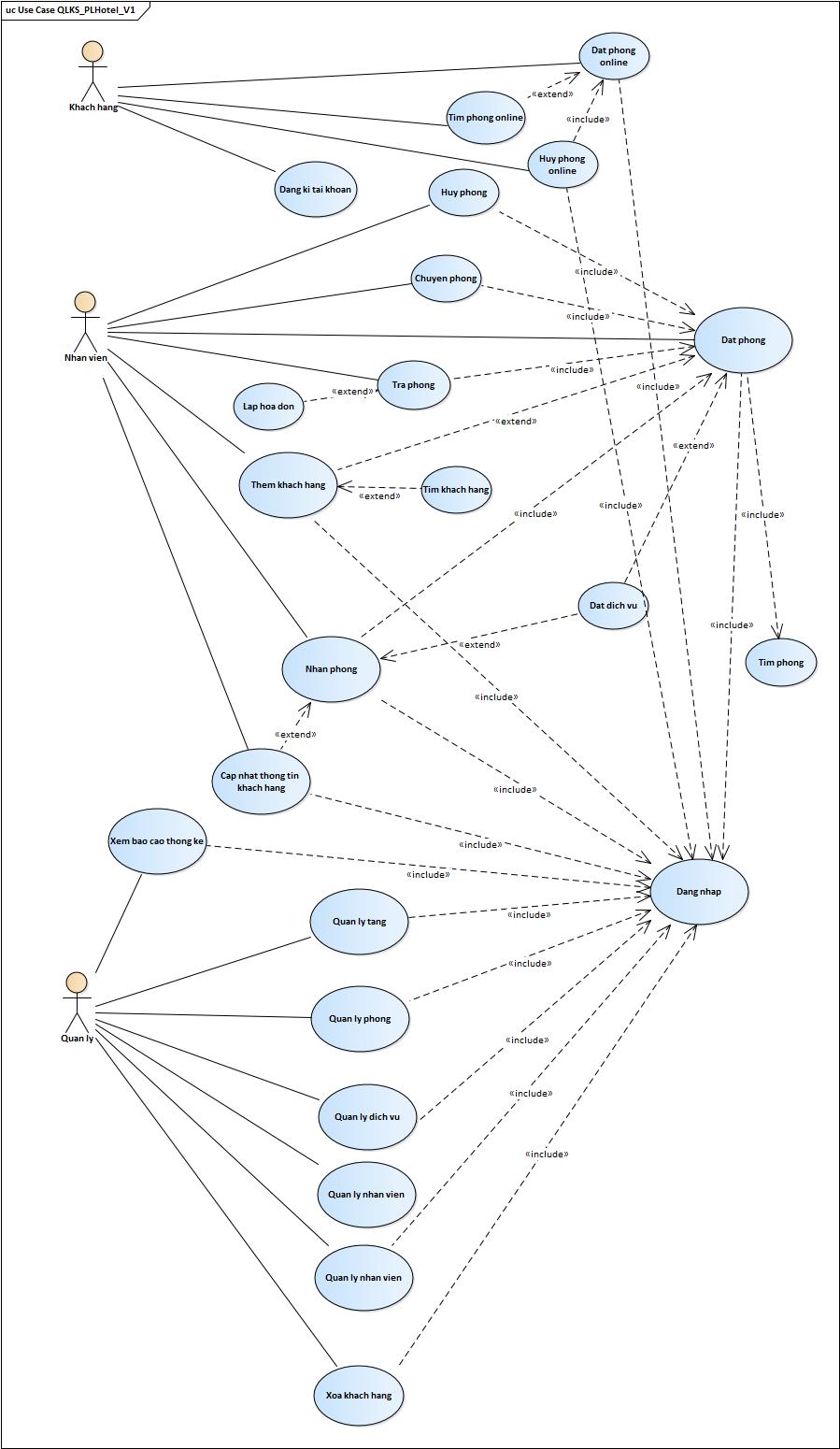
* Data Segment.
* Index Segment.
* Temporary Segment.
* Rollback Segment.

Những Segment này sẽ được cấp phát cho những đối tượng ở những thời điểm khác nhau. Ví dụ Data Segment sẽ được cấp phát cho Table khi thực hiện câu lệnh tạo Table, còn Temporary Segment được cấp phát khi thực hiện các câu lệnh SQL hay câu lệnh tạo bảng tạm.

# CHƯƠNG 3. PHÂN TÍCH VÀ THIẾT KẾ

## 3.1 Phân tích yêu cầu bằng UML

### 3.1.1 Usecase tổng quát



### 3.1.2 Danh sách tác nhân và mô tả

|  |  |
| --- | --- |
| Tác nhân | Mô tả tác nhân |
| Khách hàng | Tham gia chức năng đặt phòng online. |
| Nhân viên |  |
| Admin |  |
|  |  |
|  |  |
| … |  |

### 3.1.3 Danh sách các tình huống hoạt động (Use cases)

### 3.1.4 Tình huống hoạt động

|  |  |
| --- | --- |
| ID | Tên Use case |
| UC001 | Đặt phòng |
| UC002 | Nhận phòng |
| UC003 | Đổi phòng |
| UC004 | Trả phòng |
| UC005 | Xem báo cáo thống kê |
| UC006 | Đăng nhập |
| UC007 | Thêm khách hàng-Đăng ký |
| UC008 |  |
| … |  |

#### 3.1.4.1 Đặc tả use case UC001.

#### 3.1.4.2 Activity diagram UC001.

#### 3.1.4.3 Sequence diagram cho từng tình huống UC001.

## 3.2 Class Diagram

## 3.3 Mô hình triển khai hệ thống.

# PHỤ LỤC

Phần này bao gồm những nội dung cần thiết nhằm minh họa hoặc hỗ trợ cho nội dung Đồ án như: số liệu, mẫu biểu, tranh ảnh…

− Phụ lục không được dày hơn phần chính của Đồ án.

# DANH MỤC TÀI LIỆU THAM KHẢO

Tiếng Việt: [1] Đinh Mạnh Tường. Cấu trúc dữ liệu & Thuật toán. Chương 1 và 8. Nhà xuất bản khoa học và kỹ thuật. Hà nội, 2001.

Tiếng Anh: [1] Aho A.V. , Hopcroft J.E. and Ullman J.D. Data Structures and Algorithms. Pages: 200-345. Addison-Wesley. London, 1983.

Ghi chú: - Font chữ size 13. - Danh mục tài liệu tham khảo phải xếp theo abc tác giả, tên tài liệu, nơi và năm phát hành. - Nếu xếp theo tên tác giả thì xếp abc theo họ, sau đó đến tên tài liệu, nơi và năm phát hành. - Danh mục tài liệu tham khảo xếp cuối cùng, sau các trang phụ lục.