TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

KHOA CÔNG NGHỆ THÔNG TIN

BÀI TẬP LỚN MÔN: CHUYÊN ĐỀ CÔNG NGHỆ PHẦN MỀM

Tên đề tài:

TÌM HIỂU VỀ JSF VÀ HIBERNATE

Người hướng dẫn:ThS Vũ Đình Hồng

Người thực hiện: **Nguyễn Ngọc Cường - 51303020**

Nguyễn Nghĩa Dinh - 51303254

Lớp: 13050302

Khóa: 17

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2016

LỜI CẢM ƠN

Trong suốt thời gian qua, nhờ sự giảng dạy tận tâm của quý Thầy Cô Khoa Công Nghệ Thông Tin, trường Đại học Tôn Đức Thắng, chúng em đã học hỏi được rất nhiều điều bổ ích và tích lũy cho mình một số kiến thức để hoàn thành bài báo cáo này. Chúng em xin chân thành cảm ơn.

Chúng em xin cảm ơn Thầy Vũ Đình Hồng đã tận tình chỉ bảo chúng em qua những buổi báo cáo đồ án, Thầy đã chỉ chúng em cách thức làm bài, chỉ điểm những chỗ còn sai sót chưa phù hợp cũng như phải làm sao để trình bày bố cục đẹp. Nếu không có những lời hướng dẫn, dạy bảo của thầy thì bài thu hoạch của chúng em cũng rất khó để hoàn thiện. Một lần nữa chúng em xin chân thành cảm ơn Thầy.

Bước đầu đi vào thực tế với nền kiến thức mở rộng, kiến thức chúng em còn hạn chế và nhiều bỡ ngỡ. Vì thế, trong quá trình biên soạn khó tránh khỏi những sai sót, chúng em rất mong nhận được những ý kiến đóng góp quý báu của quý Thầy/Cô và Các bạn để bài báo cáo hoàn thiện hơn.

CAM KẾT

Tôi xin cam đoan đây là sản phẩm bài tập lớn của riêng tôi / chúng tôi và được sự hướng dẫn của thầy Vũ Đình Hồng. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong đồ án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung bài tập lớn của mình. Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

TP. Hồ Chí Minh, ngày tháng năm

Tác giả

(ký tên và ghi rõ họ tên)

Nguyễn Ngọc Cường

Nguyễn Nghĩa Dinh

Mục Lục

CHƯƠNG I. Java Server Faces (JSF)

I. GIỚI THIỆU VỀ JAVA SERVSER FACES

JSF là viết tắt của JavaServer Faces, một bản miêu tả kĩ thuật giúp đơn giản hóa việc phát triển giao diện cho các ứng dụng Web viết bằng Java bằng cách dùng các thành phần dùng lại được (reusable components).

JSF là một bộ khung (framework) phát triển các ứng dụng Web viết bằng Java nhằm làm đơn giản hóa quá trình phát triển giao diện người dùng cho các ứng dụng J2EE. Để tạo ra giao diện hiển thị, JSF dùng dạng cấu trúc cây của các thẻ, mỗi thẻ là một thành phần giao diện (component) và FacesServlet servlet sẽ thực hiện công đoạn chuyển đổi ra giao diện tương ứng cho người dùng với định dạng HTML. Ngoài ra, JSF cũng có thể sử dụng các kĩ thuật hiển thị khác, như XUL. JSF bao gồm:

Một tập các hàm API để biểu diễn các thành phần UI (giao diện người dùng) và quản lí thạng thái của chúng, xử lí các sự kiện và kiểm tra dữ liệu đầu vào, qui định việc di chuyển trang (page navigation), hỗ trợ tính đa ngôn ngữ và hỗ trợ cho người sử dụng (accessibility)

Một số thành phần (component) có sẵn

Hai thư viện chứa các thẻ tùy biến JSP (JavaServer Pages custom tag) để có thể biểu diễn một JavaServer Faces interface bên trong một trang JSP.

Mô hình sự kiện ở phía máy chủ (server-side event model)

Khả năng quản lí trạng thái

Managed Beans (JavaBeans được tạo bằng dependency injection - xem thêm Spring framework)

Bản miêu tả kĩ thuật JSF được qui định trong JSR 127 của Quá trình Cộng đồng Java.

JSF là một môi trường phát triển GUI khá truyền thống, giống như AWT, SWT, và Swing. Một trong những lợi ích chính của nó là nó làm cho việc phát triển Web dễ dàng hơn bằng cách giao những công việc khó khăn cho các nhà phát triển khung công

tác, chứ không phải cho các nhà phát triển ứng dụng. Cứ cho là bản thân JSF phức tạp hơn nhiều so với các khung công tác Web khác, nhưng sự phức tạp này được che giấu không để cho các nhà phát triển ứng dụng biết. Phát triển các ứng dụng Web trong JSF dễ dàng hơn nhiều so với hầu hết các khung công tác khác: nó đòi hỏi viết mã ít hơn, ít phức tạp hơn, và ít việc cấu hình hơn.

JSF là khung công tác dễ nhất để tìm hiểu. Nó được định hướng để tạo các ứng dụng Web (không chỉ là các trang web). Nó cho phép tập trung vào việc mã hóa Java mà không cần đối phó với các đối tượng yêu cầu, các đối tượng phiên, các thông số yêu cầu, hoặc đối phó với các tệp tin XML phức tạp. Với JSF, nhiều thứ thực hiện nhanh hơn so với các khung công tác Web Java khác.

II. ƯU ĐIỂM CỦA JSP

❖ Đánh giá chung thì ưu điểm của JSF:

- Tách biệt hoàn toàn giữa hành vi và cách trình bày
- Kiểm soát tính có trạng thái (statefulness) ở mức thành phần
- Các sự kiện dễ dàng được liên kết với mã phía máy chủ
- Sử dụng các khái niệm thành phần UI và tầng Web (Web-tier) quen thuộc
- Cung cấp nhiều dụng cụ của nhà sản xuất phần mềm đã tiêu chuẩn hóa
- Hỗ trợ IDE tuyệt vời cùng với cung cấp nhiều dụng cụ của nhà sản xuất phần mềm đã tiêu chuẩn hóa
- Sử dụng các khái niệm thành phần giao diện và phân lớp mô hình Website (Web-tier) quen thuộc cùng với các sự kiện dễ dàng liên kết với mã phía máy chủ.
- Tách biệt hoàn toàn giữa hành vi và cách trình bày (action and present) thêm vào đó là kiểm soát tính có trạng thái (statefulness) ở mức độ thành phần.
- JSF bao gồm:

+ Một mô hình sự kiện-xuất bản(event – public) cùng với một bộ chứa (container) nhẹ, đảo ngược-điều khiển (IoC)

+ Các thành phần cho hầu hết mọi đặc tính GUI phổ biến khác, bao gồm (nhưng không hạn chế):

* Công cụ biểu hiện (rendering) cắm chạy được

* Trình duyệt tính hợp lệ phía máy chủ

* Biến đổi dữ liệu

* Quản lý dẫn hướng trang

Một ứng dụng JSF điển hình bao gồm các phần sau đây:

+ JavaBeans để quản lý trạng thái và hành vi của ứng dụng

+ Các thành phần GUI có trạng thái

+ Phát triển hướng sự kiện (thông qua các trình nghe-listener) giống như trong phát triển GUI truyền thống

+ Các trang biểu diễn các khung nhìn theo phong cách Model-View-Controller (MVC); các trang web tham khảo các gốc khung nhìn (view roots) thông qua cây thành phần JSF

III. CÁC PHIÊN BẢN CỦA JSF

- JSF 1.0 - bản miêu tả kĩ thuật của JSF chính thức đầu tiên

- JSF 1.1 - bản sửa lỗi chính thức. Không có thay đổi về kĩ thuật hay HTML renderkit.

- JSF 1.2 - bản chính thức chuẩn bị ra mắt và được miêu tả bởi JSR 252.

- cung cấp các XML Schema cho các file cấu hình, thay cho việc dùng DTD

- các cải tiến để cho phép các 'faces applications có thể xử lí nhiều khung (multi-frame), hay các thiết kế đa cửa sổ người dùng (multi-window UI)

- các cải tiến thư viện thẻ f: để nâng cao 'TCK coverage, các sự kiện liên quan chu trình sống của f:view, và một số đặc tính nhỏ khác

- các cải tiến trong việc hỗ trợ trang trí cho các đối tượng API

- cải tiến an ninh cho phía trình khách trong việc lưu giữ trạng thái

- giải quyết vấn đề "duplicate button press"

- tổ chức lại bản miêu tả kĩ thuật theo từng phần tiêu chuẩn (normative), và không tiêu chuẩn, để giúp cho việc hiện thực dễ dàng hơn.

- Các sữa lỗi cho portlet

- Một sỗ sửa lỗi đòi hỏi sự thay đổi chút ít trong bản miêu tả kĩ thuật

IV. JSF KẾT HỢP VỚI MVC

JSF là kết quả của các bài đã học thu được qua nhiều năm phát triển của các kỹ thuật phát triển Web trên nền tảng Java. Xu hướng này bắt đầu với công nghệ JSP, cho dù có nhiều ưu điểm, đã làm cho quá dễ dàng trộn lẫn mã Java với mã HTML (và mã giống như HTML). Bước tiến tiếp theo là kiến trúc Mô hình 1 trong đó các nhà phát triển đẩy hầu hết mã mặt sau vào trong các thành phần JavaBeans và sau đó nhập khẩu các thành phần JavaBeans vào các trang Web với thẻ <jsp:useBean>. Điều này đã hoạt động tốt với các ứng dụng Web đơn giản, nhưng nhiều nhà phát triển Java không thích kết hợp công nghệ JSP với các đặc tính C++ ví dụ như là lệnh bao gồm (include) tĩnh. Vì vậy, kiến trúc Mô hình 2 đã được giới thiệu.

Về cơ bản, kiến trúc Mô hình 2 là phiên bản MVC thấp hơn cho các ứng dụng Web. Trong kiến trúc Mô hình 2, trình điều khiển được thể hiện bằng các servlet (hoặc các Action) và việc hiển thị được giao cho các trang JSP. Struts của Apache là một thực thi Mô hình 2 đơn giản hoá, nơi mà các Action thế chỗ các servlet. Trong Struts, logic của trình điều khiển ứng dụng được tách ra khỏi các dữ liệu của nó (được biểu diễn bằng ActionForms). Các lời phàn nàn chủ yếu chống lại Struts là có thể cảm thấy nó có nhiều tính thủ tục hơn hướng đối tượng ("COBOL cho Web"). WebWork và Spring MVC là hai kiến trúc Mô hình 2 khác để cải tiến Struts để ít tính thủ tục hơn, nhưng cả hai đều không được chấp nhận rộng rãi như là Struts. Và cả hai cũng không cung cấp một mô hình thành

phần có trạng thái như JSF đã làm. (Struts 2 được xây dựng bên trên WebWork, và cơ sở mã lệnh Struts ban đầu đã bị loại bỏ. Ngay cả Struts cũng không muốn).

Vấn đề thực sự với hầu hết các khung công tác Mô hình 2 là ở chỗ mô hình sự kiện quá đơn giản (chủ yếu là một MVC thu nhỏ nhiều), và nó không có thành phần GUI có trạng thái, để lại quá nhiều công việc cho nhà phát triển. Một mô hình thành phần và sự kiện phong phú hơn sẽ làm cho dễ dàng tạo ra các loại tương tác mà hầu hết người dùng mong đợi. Cũng giống như công nghệ JSP, hầu hết các khung công tác của Mô hình 2 cũng làm cho quá dễ dàng trộn lẫn bố trí và định dạng HTML với các thẻ tùy biến GUI, hoạt động hơi giống như các thành phần, ngoại trừ là chúng không có trạng thái. Và một số kiến trúc Mô hình 2 (như Struts kinh điển) phạm lỗi tách biệt hành vi với trạng thái, đã mang lại cho nhiều nhà phát triển Java cảm giác như họ đang lập trình COBOL.

Trong thực thi MVC của JSF, việc ánh xạ beans quản lý sẽ làm trung gian giữa khung nhìn và mô hình. Vì thế, điều quan trọng là hạn chế logic nghiệp vụ và logic lâu bền trong các beans quản lý, được gắn với JSF. Một giải pháp thay thế chung là giao logic nghiệp vụ cho mô hình ứng dụng. Trong trường hợp này, các bean quản lý cũng ánh xạ với các đối tượng của mô hình, ở nơi khung nhìn có thể hiển thị chúng (như các thuộc tính của bean quản lý). Có xu hướng tách biệt beans quản lý của ra thành hai loại: beans quản lý được gắn với JSF (trình điều khiển) và beans quản lý không được gắn với JSF (các đối tượng mô hình).

Không giống như công nghệ JSP, thực thi của khung nhìn JSF là một mô hình thành phần có trạng thái. Khung nhìn JSF gồm hai phần gốc khung nhìn (view root) và các trang JSP. Gốc khung nhìn là một sưu tập các thành phần UI để duy trì trạng thái của UI. Giống như Swing và AWT, các thành phần JSF sử dụng một mẫu thiết kế phức hợp để quản lý một cây các thành phần (nói đơn giản: một bộ chứa chứa các thành phần; một bộ chứa là một thành phần). Trang JSP liên kết các thành phần UI với các trang JSP và cho phép liên kết các thành phần trường với các thuộc tính của beans hậu thuẫn (hay đúng hơn là các thuộc tính của thuộc tính) và liên kết các nút nhấn với trình xử lý sự kiện và các phương thức hành động.

V. MỤC TIÊU CỦA JSF

8 mục tiêu thiết kế sau là lí do cho sự ra đời của JSF:

- Tạo ra một bộ khung gồm các thành phần giao diện người dùng chuẩn (standard GUI component framework) nhằm giúp cho các công cụ phát triển dễ dàng hơn cho người dùng trong việc tạo GUI chất lượng cao đồng thời quản lí các kết quả của GUI với xử lí thực thi của chương trình.
- Định ra một tập các lớp cơ sở của Java (lightweight Java base classes) biểu diễn cho các thành phần UI, trạng thái mỗi thành phần, và các sự kiện đầu vào. Những lớp này sẽ xử lí những vấn đề liên quan đến chu kì sống của GUI, đặc biệt là quản lí trạng thái trong suốt chu trình sống của một trang của thành phần GUI đó.
- Cung cấp một tập các thành phần GUI chung, bao gồm các thành phần HTML form input. Những thành phần này sẽ được dẫn xuất từ tập đơn giản các lớp cơ sở (đề cập ở #1) đề từ đó có thể định ra các thành phần mới.
- Cung cấp một mô hình JavaBeans để có thể truyền đi (dispatch) các sự kiện từ các GUI controls phía máy khách đến các xử lí hiện thực cụ thể từ phía ứng dụng máy chủ.
- Định ra các hàm APIs để kiểm chứng dữ liệu nhập, bao gồm hỗ trợ kiểm chứng từ phía máy chủ.
- Chỉ định một mô hình để có thể đa ngôn ngữ hóa hay địa phương hóa các GUI.
- Khởi tạo tự động dữ liệu ra phù hợp cho máy khách đích, dựa vào mọi dữ liệu cấu hình ở máy khách đó, bao gồm cả dựa vào phiên bản trình duyệt, ví dụ.
- Việc khởi tạo tự động dữ liệu ra còn kèm theo các đòi hỏi về hỗ trợ người dùng (accessibility), được qui định bởi WAI.

VI. VAI TRỎ CỦA JSF

Bởi vì việc phân chia công việc được cho phép bởi thiết kế công nghệ JSF, việc phát triển và sửa chữa các ứng dụng JSF có thể xử lý dễ dàng và nhanh chóng. Các thành viên củamột nhóm phát triển thông thường bao gồm một danh sách dưới đây. Trong nhiều nhóm, các nhà phát triển riêng biệt đóng nhiều hơn một trong những vai trò dưới đây

PageAuthors: người sử dụng ngôn ngữ đánh dấu, giống như HTML, để tạo ra các trang cho ứng dụng Web. Khi sử dụng framework công nghệ JSF, page authors sẽ hầu hết sử dụng thư viện thẻ

Application Developers: người lập trình mô hình các thành phần, các xử lý sự kiện, các kiểm tra, và navigation của trang. Application developer có thể cung cấp các lớp helper mở rộng

Component Writers: người có kinh nghiệm lập trình UI và đề nghị tạo ra các thành phần tùy biến sự dụng ngôn ngữ lập trình.

Những người này có thể tạo ra các thành phần của riêng họ trực tiếp từ các lớp thành phần, hoặc họ có thể kế thừa các thành phần chuẩn cung cấp bởi công nghệ JSF

Tool Verdors: người cung cấp các công cụ nhằm tạo ra công nghệ JSF xây dựng UI phía server dễ dàng hơn. Những thành viên chính của công nghệ JSF sẽ là page authors và application developers.

,

VII. CÁC THÀNH PHẦN CƠ BẢN

Như hầu hết các công nghệ, Faces có những quy định của nó nhằm đưa ra các khái niệm cơ bản cho những đặc điểm mà nó cung cấp. Chúng tôi đang nói vế những elements như các component UI, validator và rederer. Bạn có thể có một ý kiến tốt về chúng là cái gì, nhưng qui định viết các ứng dụng Faces, bạn phải hiểu được chúng là gì trong thế giới JSF. Phần này, chúng tôi trình bày về những khái niệm cốt lõi và giải thích mối quan hệ của chúng với những phần khác. Có tám khái niệm khi bạn phát triển các ứng dụng JSF

UI Component (còn gọi là một control hay đơn giản là component) : một đối tượng có trạng thái, được chứa trên server, cung cấp các chức năng cụ thể để tương tác với một người dùng cuối. UI component là những JavaBean với các thuộc tính, phương thức, sự kiện. Chúng được tổ chức thành một cây các component thường hiển thị như một trang

Rederer: Trả lời cho việc hiển thị một UI component và trao đổi một dữ liệu nhập của user vao giá trị của component. Renderer có thể được thiết kế để làm việc với một hoặc nhiều UI component, và một UI component có thể tập hợp với nhiều renderer khác nhau.

Validator: Trả lời cho việc chắc chắn rằng giá trị nhập vào bởi user được chấp nhận. Một hoặc nhiều validator có thể được tập hợp với một UI component

Backing beans: Các Java Bean xác định tập hợp các giá trị từ các UI component và bổ sung các phương thức listener cho event. Chúng cũng có thể nắm giữ các tham chiếu đến các UI component

Converter: Chuyển đổi một giá trị của component thành và từ một chuỗi để hiển thị. Một UI component có thể được tập hợp với một converter duy nhất.

Event/listener: JSF sử dụng mô hình event/listener JavaBeans (cũng được sử dụng cho Swing). UI component (và những đối tượng khác) tạo ra các event, và các listener có thể đăng ký để xử lý các sự kiện

Messages: Thông tin hiển thị cho user. Chỉ bất kỳ phần ứng dụng nào (backing beans, validators, converter …) có thể tạo ra thông tin hoặc thông điệp lỗi nhằm hiển thị cho user

Navigation: Khả năng di chuyển từ một trang đến trang khác. JSF có một hệ thống navigation mạnh mẽ tích hợp với những event listeners.

CHƯƠNG II. Hibernate

I. TỔNG QUAN VỀ HIBERNATE

1. Hibernate framework là gì

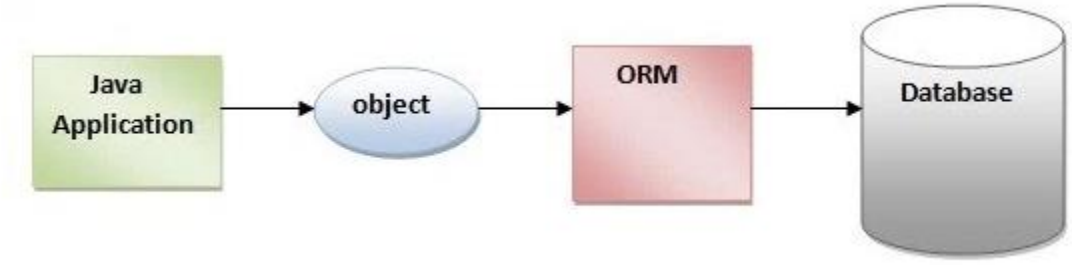
Hibernate là một trong những ORM Framework. Hibernate framework là một framework cho persistence layer. Như vậy, nhờ có Hibernate framework mà giờ đây khi bạn phát triển ứng dụng bạn chỉ còn chú tâm vào những layer khác mà không phải bận tâm nhiều về persistence layer nữa. Hibernate giúp lưu trữ và truy vấn dữ liệu quan hệ mạnh mẽ và nhanh. Hibernate cho phép bạn truy vẫn dữ liệu bằng ngôn ngữ SQL mở rộng của Hibernate (HQL) hoặc bằng SQL thuần.

2. Giới thiệu Hibernate Framework

Hibernate framework bắt đầu vào năm 2001 bởi Gavin King như một thay thế cho thực thể bean phong cách EJB2. Phiên bản ổn định của Hibernate cho đến 16 tháng 7 năm 2014, là hibernate 4.3.6. Đó là hữu ích cho người mới bắt đầu và những người có kinh nghiệm.

Hibernate framework đơn giản hóa việc phát triển các ứng dụng java để tương tác với cơ sở dữ liệu. Hibernate là một mã nguồn mở, nhẹ ORM (Object Relational Mapping) công cụ,.

Một công cụ ORM đơn giản hóa việc tạo ra các dữ liệu, thao tác dữ liệu và truy cập dữ liệu. Đây là một kỹ thuật lập trình mà các bản đồ các đối tượng để các dữ liệu được lưu trữ trong cơ sở dữ liệu.



3. Ưu điểm của Hibernate framework

Mã nguồn mở và nhẹ: Hibernate framework mã nguồn mở theo giấy phép LGPL và trọng lượng nhẹ.

Thực hiện nhanh: Hiệu suất của hibernate framework là nhanh vì bộ nhớ cache được sử dụng trong nội bộ framework hibernate. Có hai loại bộ nhớ cache trong bộ nhớ cache ngủ đông khuôn khổ mức độ đầu tiên và bộ nhớ cache cấp độ thứ hai. Bộ nhớ cache cấp độ đầu tiên được kích hoạt bydefault.

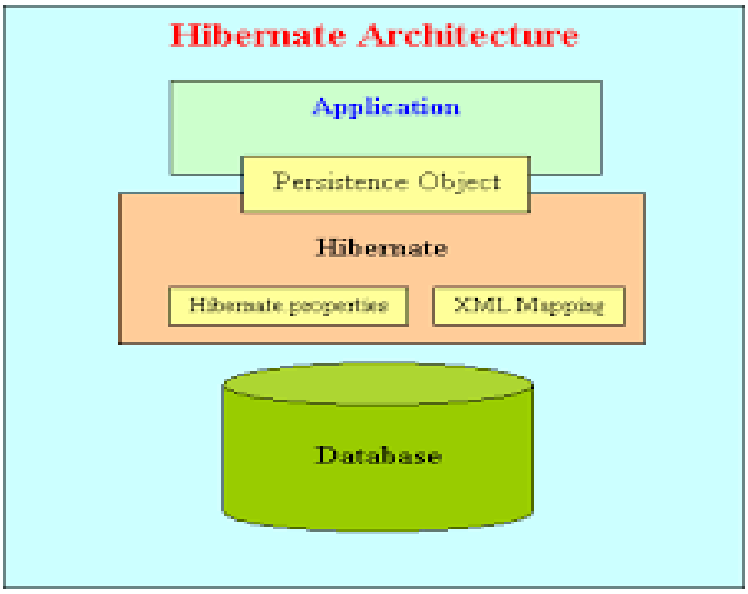
Cơ sở dữ liệu truy vấn độc lập: HQL (Hibernate Query Language) là phiên bản hướng đối tượng của SQL. Nó tạo ra các truy vấn cơ sở dữ liệu độc lập. Vì vậy, bạn không cần phải viết các truy vấn cơ sở dữ liệu cụ thể. Trước khi Hibernate, Nếu cơ sở dữ liệu được thay đổi cho dự án, chúng ta cần phải thay đổi các truy vấn SQL cũng dẫn đến các vấn đề bảo trì.

Tạo bảng tự động: Hibernate framework cung cấp các thiết bị để tạo ra các bảng của cơ sở dữ liệu tự động. Vì vậy, không cần thiết phải tạo ra các bảng trong cơ sở dữ liệu bằng tay.

Đơn giản hóa phức tạp join: Để lấy dữ liệu thành nhiều bảng là dễ dàng trong hibernate framework.

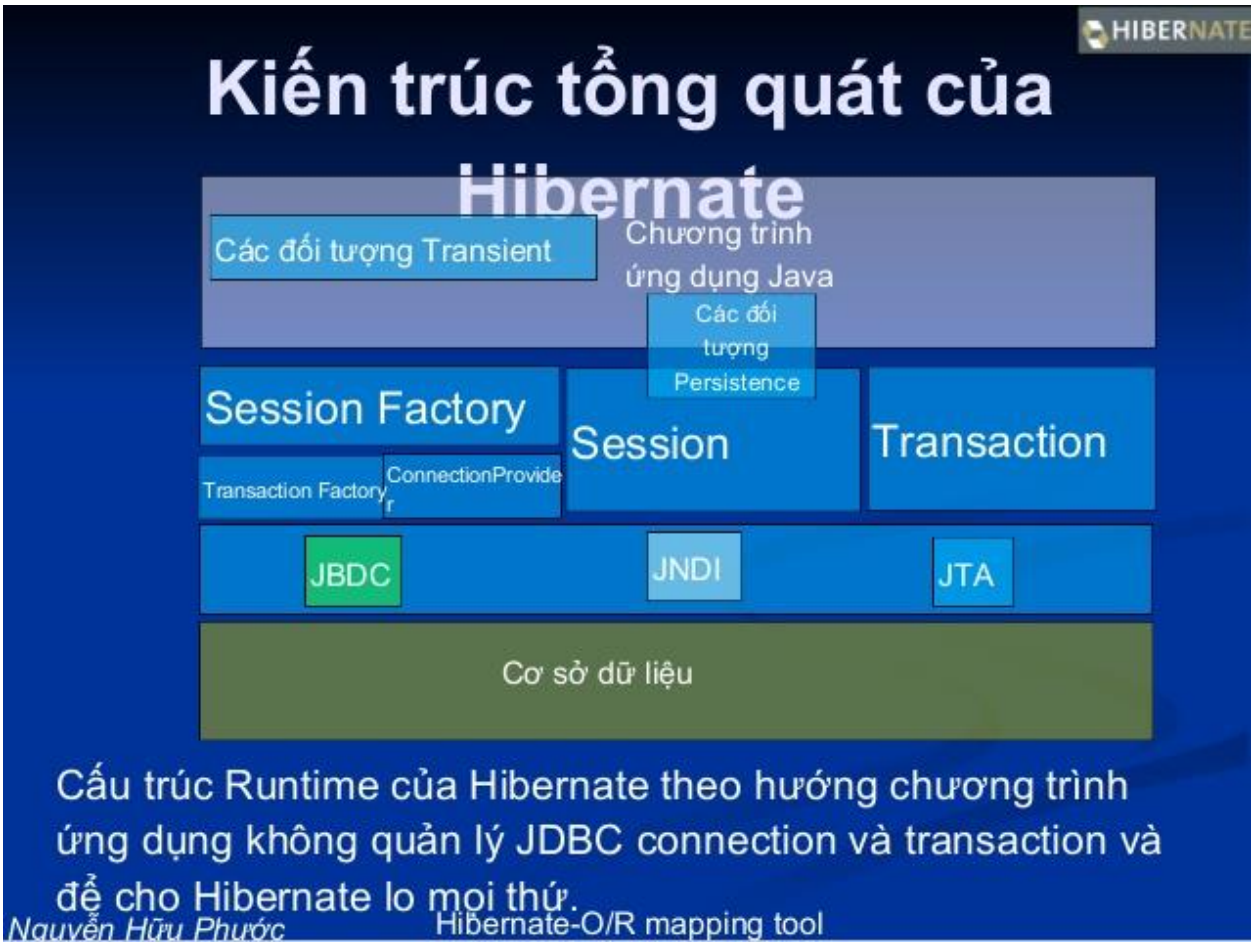
Cung cấp số liệu thống kê truy vấn và tình trạng cơ sở dữ liệu: Hibernate hỗ trợ Cache truy vấn và cung cấp các số liệu thống kê về tình trạng cơ sở dữ liệu truy vấn.

II. CẤU TRÚC HIBERNATE



Hình 2.2.1 – Cấu trúc tổng quanhibernate.

Ở trên là biểu đồ mô tả tổng thể về cấu trúc Hibernate, hình ảnh mô tả chi tiết ở trên ta dễ nhận thấy rằng Hibernate sử dụng cơ sở dữ liệu và các file cấu hình(Hibernate.properties và XML Mapping file) để cung cấp dịch vụ persistence cho ứng dụng.



Hình 2.2.2 – Cấu trúc chi tiết của Hibernate.

Ở hình trên cho cái nhìn chi tiết về cấu trúc Hibernate trong ứng dụng. Hibernate sử dụng nhiều API của Java như JDBC, Java Transaction, Java Naming and Directory Interface. JDBC cho phép bất kỳ cơ sở dữ liệu nào với một trình điều khiển JDBC đều được hỗ trợ bởi Hibernate.

Phần dưới đây mô tả ngắn gọn các thành phần tham gia trong cấu trúc Hibernate mà cần nắm vững để có thể sử dụng trong việc xây dựng các ứng dụng sau này, nắm vững cấu trúc Hibernate có lợi thế hơn.

1. Cấu hình đối tượng

Các đối tượng cấu hình chỉ tạo ra trong quá trình khởi chạy ứng dụng lần đầu tiên, nó đại diện cho một tập tin cấu hình, thuộc tính theo yêu cầu của Hibernate. Các đối tượng cấu hình cung cấp hai thành phần.

- **Kết Nối Cơ Sở Dữ Liệu:** Được xử lý thông qua một hoặc nhiều tập tin cấu hình được hỗ trợ bởi Hibernate. Những tập tin này là hibernate.properties và hibernate.cfg.xml.
- **Cài Đặt Lớp Mapping:** Thành phần này tạo ra sự kết nối giữa các lớp Java và các bảng cơ sở dữ liệu.

2. Đối tượng SessionFactory

Cấu hình đối tượng được sử dụng để tạo ra một đối tượng SessionFactory do đó cấu hình Hibernate của ứng dụng sử dụng các tập tin cấu hình và cho phép khởi tạo đối tượng session.

SessionFactory là đối tượng lớn trong ứng dụng nên thường nó được tạo ra trong suốt quá trình khởi động và giữ để sử dụng sau. Mỗi cơ sở dữ liệu sử dụng một tập tin cấu hình riêng biệt và ừng với một cơ sở dữ liệu sẽ là một đối tượng SessionFactory, nếu đang sử dụng nhiều cơ sở dữ liệu thì sẽ phải tạo ra nhiều đối tượng SessionFactory.

3. Đối tượng Session

Session được sử dụng để kết nối với cơ sở dữ liệu. Đối tượng Session được khởi tạo khi tương tác với cơ sở dữ liệu. đối tượng liên tục được lưu lại và lấy thông qua một đối tượng Session. Các đối tượng Session không nên giữ mở trong một thời gian dài bởi vì không an toàn.

4. Đối tượng Transaction

Đây là một đối tượng tùy chọn và ứng dụng Hibernate có thể chọn không sử dụng, thay vì quản lý giao dịch trong mã ứng dụng của riêng mình.

5. Đối tượng Query

Đối tượng truy vấn sử dụng SQL hoặc Hibernate Query Language (HQL) để lấy dữ liệu từ cơ sở dữ liệu.

6. Tiêu chuẩn đối tượng

Các tiêu chí đối tượng được sử dụng để tạo ra và thực hiện theo hướng đối tượng, tiêu chí truy vấn để lấy đối tượng.

III. CẤU HÌNH HIBERNATE

Hibernate yêu cầu phải biết trước những thông tin mapping trong lớp Java liên quan đến cơ sở dữ liệu. Ngoài ra Hibernate cũng đòi hỏi một tập hợp các thiết lập liên quan đến cơ sở dữ liệu và các thông số liên quan khác. Tất cả những thông tin Hibernate yêu cầu, được khai báo trong file xml có tên hibernate.cfg.xml. Sau đây là những thuộc tính quan trọng cần phải khai báo trong cấu hình Hibernate ở trong file hibernate.cfg.xml.

Hibernate là một framework cho phép người lập trình liên kết các đối tượng lớp trong Java với thể hiện của chúng trong hệ quản trị cơ sở dữ liệu bằng phương pháp (O/R Mapping). Nó giúp người lập trình làm việc với cơ sở dữ liệu một cách linh hoạt và tiện lợi hơn.

STT	Mô tả thuộc tính
1	**hibernate.dialect** Thuộc tính này của Hibernate tạo ra các SQL thích hợp cho các cơ sở dữ liệu được chọn.
2	**hibernate.connection.driver_class** Lớp điều khiển JDBC.
3	**hibernate.connection.url** Các url đến cơ sở dữ liệu.
4	**hibernate.connection.username** Username kết nối cơ sở dữ liệu
5	**hibernate.connection.password** Mật khẩu kết nối cơ sở dữ liệu.
6	**hibernate.connection.pool_size** Giới hạn số lượng kết nối chờ đợi trong Hibernate.

7	**hibernate.connection.autocommit** Cho phép chế độ tự động đẩy dữ liệu cho các kết nối JDBC

IV. SESSION TRONG HIBERNATE

1. Khái niệm Session

Session là một khái niệm phổ biến được dùng trong lập trình các website có kết nối với cơ sở dữ liệu database. Đặc biệt các chức năng như đăng nhập, đăng xuất người dùng sẽ khó có thể thực hiện được nếu không sử dụng Session.

2. Session trong Hibernate

Session được sử dụng để có được một kết nối vật lý với một cơ sở dữ liệu. Các đối tượng Session được khởi tạo mỗi lần tương tác với cơ sở dữ liệu. đối tượng liên tục được lưu lại và lấy thông qua một đối tượng Session. Các đối tượng Session không nên giữ mở trong một thời gian dài bởi vì an toàn và cần được tạo ra và phá hủy khi cần thiết.

3. Các phương thức Session

Có nhiều phương thức được cung cấp bởi Session Interface, ở đây mình chỉ liệt kê những phương thức quan trọng.

STT	Mô tả phương thức Session
1	**Transaction beginTransaction()** Bắt đầu phiên làm việc và trả về đối tượng liên quan.
2	**Void cancelQuery()** Hủy bỏ việc thực hiện các truy vấn hiện tại.
3	**void clear()** Xóa Session khi hoàn thành phiên làm việc.
4	**Connection close()** Kết thúc phiên làm việc bằng cách giải phóng các kết nối JDBC

5	**Criteria createCriteria(Class persistentClass)** Tạo một tiêu chuẩn mới, cho các lớp thực thể nhất định.
6	**Criteria createCriteria(String entityName)** Tạo một tiêu chuẩn mới, với tên thực thể.
7	**Serializable getIdentifier(Object object)** Trả về giá trị định danh của đối tượng.
8	**Query createFilter(Object collection, String queryString)** Tạo một thể hiện mới của truy vấn để thu thập và lọc chuỗi.
9	**Query createQuery(String queryString)** Câu truy vấn HQL
10	**SQLQuery createSQLQuery(String queryString)** Câu truy vấn SQL
11	**void delete(Object object)** Xóa một đối tượng từ cơ sở dữ liệu.
12	**void delete(String entityName, Object object)** Xóa một đối tượng từ cơ sở dữ liệu.
13	**Session get(String entityName, Serializable id)** Trả về đối tượng có tên là entityName nếu id tồn tại, ngược lại trả về null.
14	**SessionFactory getSessionFactory()** Lấy Session Factory để tạo session.
15	**void refresh(Object object)** Refresh lại đối tượng lấy từ cơ sở dữ liệu.
16	**Transaction getTransaction()** Lấy Transaction để gán session vào Transaction.
17	**boolean isConnected()** Kiểm tra Session đang được kết nối.
18	**boolean isDirty()** Đồng bộ với cơ sở dữ liệu.

19	**boolean isOpen()** Kiểm tra Session vẫn đang mở.
20	**Serializable save(Object object)** Lưu đối tượng xuống cơ sở dữ liệu.
21	**void saveOrUpdate(Object object)** Lưu hoặc cập nhật đối tượng xuống cơ sở dữ liệu.
22	**void update(Object object)** Cập nhật đối tượng xuống cơ sở dữ liệu.
23	**void update(String entityName, Object object)** Cập nhật đối tượng dựa vào tên bảng dưới cơ sở dữ liệu, Nếu đối tượng truyền vào có id trùng với id của đối tượng dưới cơ sở dữ liệu thì kết quả cập nhật là thành công, ngược lại không thành công.

V. QUAN HỆ GIỮA CÁC ĐỐI TƯỢNG VỚI HIBERNATE

1. Collections Mappings

Nếu một thực thể hoặc một lớp chứa tập hợp các giá trị cho một biến, thì chúng ta có thể ánh xạ các giá trị vào một tập hợp trong java. Hibernate cung cấp các kiểu java.util.Map,java.util.Set, java.util.SortedMap,java.util.SortedSet, java.util.List và mảng thực thể hoặc giá trị persistent.

Mảng được hỗ trợ bởi Hibernate với <primitive-array> kiểu giá trị nguyên thủy và <array> cho phần còn lại. Tuy nhiên, nó thì hiếm khi được sử dụng. Nếu muốn ánh xạ một thực thể xác định mà không hỗ trợ trực tiếp bởi Hibernate, cần phải tự định nghĩa cho Hibernate hiểu về ngữ nghĩa của các bộ sưu tập tùy chỉnh mà nó thì không hề dễ dàng và mình khuyên không nên sử dụng cách này.

2. Associations Mappings

Sau đây là bốn cách thức biểu diễn các mối quan hệ giữa các đối tượng, nó được biểu diễn theo hướng 1 chiều hoặc 2 chiều.

MAPPING TYPE	DESCRIPTION
Many-to-One	Mapping mối quan hệ nhiều – một trong Hibernate
One-to-One	Mapping mối quan hệ một – một trong Hibernate
One-to-Many	Mapping mối quan hệ một – nhiều trong Hibernate
Many-to-Many	Mapping mối quan hệ nhiều – nhiều trong Hibernate

3. Components Mappings

Nó rất có thể là một lớp thực thể, nó có một tham chiếu đến một lớp khác như là một biến thành viên, nó hoàn toàn phụ thuộc vào vòng đời của các lớp thực thể sở hữu.

TÀI LIỆU THAM KHẢO

- **Google.com**
- **Chuyên đề JavaServer Faces của IBM.**
- **Java Server Faces 2.0:The Complete Reference – Ed Burns and Chris Schalk.**