Tiếng nói của chuyên gia về Công nghệ Java

Bí mật Java 6

Tiết lộ những chức năng mới trong Java 6

Phan Thành Nhân

NỘI DUNG

CHAPTER 1: JAVA 6 AT A GLANCE

Java 6 với code name là Mustang được release vào 11/12/2016, Sun đổi tên "J2SE" thành "Java SE" và bỏ ".0" phía sau version number.

Java 6 hầu như không có thay đổi về language, mọi thay đổi tập trung vào Core.

Chúng ta sẽ tìm hiểu chi tiết trong những phần sau.

CHAPTER 2: LANGUAGE AND UTILITY UPDATES

Sức mạnh chính của Java platform là 2 package java.lang và java.util. Tìm hiểu Java 6 qua 2 package này có vẻ rất hợp lý. Nhìn chung, java.lang và subpackage của nó tăng 2 class, còn java.util tăng nhiều hơn một ít, 7 interface, 16 classes và 1 Error class. Chi tiết tham khảo 2 bảng sau:

Bảng 2-1. Package java.lang.*

Package	Ver	Interface	Class	Enum	Throwable	Annotation	Total
lang	5.0	8	35	1	26+22	3	95
lang	6.0	8	35	1	26+22	3	95
lang.annotation	5.0	1	0	2	2+1	4	10
lang.annotation	6.0	1	0	2	2+1	4	10
lang.instrument	5.0	2	1	0	2+0	0	5
lang.instrument	6.0	2	1	0	2+0	0	5
lang.management	5.0	9	5	1	0+0	0	15
lang.management	6.0	9	7	1	0+0	0	17
lang.ref	5.0	0	5	0	0+0	0	0
lang.ref	6.0	0	5	0	0+0	0	0
lang.reflect	5.0	9	8	0	3+1	0	21
lang.reflect	6.0	9	8	0	3+1	0	21
Delta		0	2	0	0+0	0	2

Bảng 2-2. Package java.util.*

Package	Ver	Interface	Class	Enum	Throwable	Total
util	5.0	16	49	1	20+0	86
util	6.0	19	54	1	20+1	95
util.concurrent	5.0	12	23	1	5+0	41
util.concurrent	6.0	16	26	1	5+0	48
...concurrent.atomic	5.0	0	12	0	0+0	12
...concurrent.atomic	6.0	0	12	0	0+0	12
...concurrent.locks	5.0	3	6	0	0+0	9
...concurrent.locks	6.0	3	8	0	0+0	11
util.jar	5.0	2	8	0	1+0	11
util.jar	6.0	2	8	0	1+0	11
util.logging	5.0	2	15	0	0+0	17
util.logging	6.0	2	15	0	0+0	17
util.prefs	5.0	3	4	0	2+0	9
util.prefs	6.0	3	4	0	2+0	9
util.regex	5.0	1	2	0	1+0	4
util.regex	6.0	1	2	0	1+0	4
util.spi	6.0	0	4	0	0+0	4
util.zip	5.0	1	16	0	2+0	17

util.zip	6.0	1	16	0	2+0	19
Delta		7	16	0	0+1	24

Chú thích: ở cột Throwable, ghi 0+1 nghĩa là có 0 Exception class và 1 Error class

Có thể thấy giữa Java 5 và Java 6 dường như không có nhiều sự thay đổi, nhưng thực sự thay đổi được tạo ra từ bên trong class. Nhìn chung, class hay package không được thêm mới, thay vào đó chúng được mở rộng.

I. Package java.lang

Đây vẫn là package cơ bản của Java platform, bạn không cần phải import nó. Package này có 2 thay đổi chính

- Console input và output
- Kiểm tra string empty

1. System.console()

System class có 1 method mới là console(), nó return một instance mới của class Console trong package java.io, method này hỗ trợ việc đọc và ghi system console. Nó có thể là việc với Reader và Writer streams nên có thể chơi với high-order byte characters (những character mà khi gọi System.out.println() có thể không được in đúng). Ngoài ra console cũng hỗ trợ method readLine() để đọc 1 dòng, readPassword() để đọc password từ console.

Listing 2-1. Printing High-Order Bit Strings

```
public class Sample {
	public static void main(String args[]) {
		String string = "Español";
		System.out.println(string);
		System.console().printf("%s%n", string);
	}
}
```

```
Output:

Espa±ol

Español
```

2. Empty Strings

Class String có thêm method mới là isEmpty() để check độ dài chuỗi bằng 0

myString.length() == 0 có thể được viết lại thành myString.isEmpty()

II. Package java.util

Có thể nói đây là package được dùng nhiều nhất, một số thay đổi chính của package này như sau:

- Calendar display name
- Deques
- Navigable maps and sets
- Resource bundle controls
- Array copies
- Lazy atomics

1. Calendar Display Names

Trước đây, không dễ để lấy một list các ngày trong tuần để cho user chọn. Class DateFormatSymbols public một số method để làm được điều này nhưng javadocs lại nói rằng "Typically you shouldn't use DateFormatSymbols directly." Vậy phải làm sao? Thay vì gọi DateFormatSymbols. getWeekdays() thì bây giờ bạn có thể gọi Calendar. getDisplayNames()

```
Map<String, Integer> names = aCalendar.getDisplayNames(

Calendar.DAY_OF_WEEK, Calendar.LONG, Locale.getDefault());
```

Parameter 1: chỉ cái bạn muốn lấy.

Parameter 2: LONG, SHORT, or ALL_STYLES. Ví dụ truyền LONG sẽ trả về Wednesday, Saturday cho các ngày trong tuần, truyền SHORT sẽ trả về Wed, Sat, còn nếu truyền ALL_STYLES thì sẽ lấy về một collection của cả 2 loại trên.

Parameter 3: locale, truyền null không có nghĩa là là lấy theo địa điểm hiện tại nên bạn phải truyền chính xác.

2. Deques

Deque nói ngắn gọn là hàng đợi 2 chiều (double-end queue), trong khi hàng đợi cho phép thêm vào một đầu và lấy ra ở đầu còn lại, thì hàng đợi 2 chiều cho phép thêm vào và lấy ra ở cả 2 đầu, giống như kết hợp giữa hàng đợi và stack vậy. Deque là một interface extends từ Queue, có 3 hiện thực của Deque là LinkedList, ArrayDeque, LinkedBlockingDeque.

- LinkedList: danh sách liên kết 2 chiều, thời gian để thực hiện tác vụ insert và remove là constant, nhưng access element thì phải thực hiện tuần tự. Nói cách khác, thời gian để search phụ thuộc vào kích thước danh sách.
- ArrayDeque: không có giới hạn về kích thước, không thread-safe, không chứa giá trị null, nhanh hơn LinkedList và Stack.
- LinkedBlockingDeque: là deque hỗ trợ thread-safe, sử dụng linked node.

3. Navigable Maps and Sets

Một điểm mới khác của Java Collections Framework chính là 2 interface mới NavigableMap và NavigableSet thừa kế từ 2 interface SortedMap và SortedSet. Chức năng của 2 interface mới này là thêm tùy chọn search cho interface chúng thừa kế.

- SortedSet: là Set mà các phần tử của nó được sort ngay từ thời điểm tạo set, nghĩa là thứ tự của chúng đã được định sẵn theo natural ordering hoặc Comparator cung cấp ở creation time. Các phần tử không trùng nhau. SortedSet implement Iterable<T>, khi lặp thì sẽ lặp theo thứ tự tăng dần. Một số phương thức hỗ trợ bởi SortedSet:
    - `Comparator<? super E> comparator()`: trả về Comparator dùng để sắp xếp set, trường hợp set này sử dụng natural ordering thì trả về null.
    - `E first()`: Trả về giá trị đầu tiên (nhỏ nhất)
    - `E last()`: Trả về giá trị cuối cùng (lớn nhất)
    - `SortedSet<E> headSet(E toElement)`: Trả về set nhỏ hơn toElement
    - `SortedSet<E> tailSet(E fromElement)`: Trả về set lớn hơn fromElement
    - `SortedSet<E> subSet(E fromElement, E toElement)`: Trả về set từ fromElement (include) tới toElement (exclusive)
- NavigableSet extends SortedSet, do đó nó thừa kế những gì SortedSet đã có, đồng thời nó cũng hỗ trợ điều hướng và descending iterator:
    - `lower(E)`: lấy phần tử lớn nhất nhỏ hơn E
    - `floor(E)`: lấy phần tử lớn nhất nhỏ hơn hoặc bằng E

    - `ceiling(E)`: lấy phần tử nhỏ nhất lớn hơn hoặc bằng E
    - `higher(E)`: lấy phần tử nhỏ nhất lơn hơn E
    - `descendingSet()`: trả về set với thứ tự ngược lại (giảm dần)
    - `descendingIterator()`: trả về iterator lặp theo hướng ngược lại (giảm dần)
    - `pollFirst()`: lấy và xóa phần tử đầu tiên
    - `pollLast()`: lấy và xóa phần tử cuối cùng
    - `headSet(E toElement, boolean inclusive)`: như của SortedSet, thêm option cho phép include hoặc exclude phần tử toElement
    - `tailSet(E fromElement, boolean inclusive)`: như của SortedSet, thêm option cho phép include hoặc exclude phần tử fromElement
    - `subSet(E fromElement, boolean fromInclusive, E toElement, boolean toInclusive)`: thêm tùy chọn include/exclude fromElement/toElement
- SortedMap, NavigableMap: tương tự SortedSet/NavigableMap nhưng các phần tử được sắp xếp dựa trên key.

4. Resource Bundle Controls

Chưa hiểu lắm.

5. Array copies

Class Arrays chứa rất nhiều static method để chơi với array. Trước Java 6, bạn có thể convert array sang list, sắp xếp, thực hiện binary search, kiểm tra bằng nhau, tạo hash code, hiển thị các element phân cáh bằng dấu phấy. Bây giờ, Mustang thêm một hành động mới là copy sử dụng method copyOf(),copyOfRange() . Nó giống như method System.arraycopy(), trong khi System.arraycopy() yêu cầu khai báo một array mới với kích thước cố định để làm destination trước khi thực hiện copy thì copyOf(),copyOfRange() không cần điều đó. Những method mới này cũng cho phép copy một phần hoặc toàn bộ array, array đích có thể nhỏ hơn hoặc lớn hơn array nguồn.

6. Lazy atomics

Sun giới thiếu package java.util.concurrent.atomic gồm các class AtomicBoolean, AtomicInteger, AtomicIntegerArray, AtomicIntegerFieldUpdater, AtomicLong, AtomicLongArray, AtomicLongFieldUpdater, AtomicReference, AtomicReferenceArray, and AtomicReferenceFieldUpdater.

Bản thân các class này đã chứa các tác vụ get, set, compare.

Theo tìm hiểu của tôi (nhanpt) thì có 2 lý do để sử dụng các class này:

- Thứ nhất, bạn có thể sử dụng các instance của class này để pass by reference, tức là quăng atomic vào hàm như là parameter, hàm đó thay đổi giá trị của atomic thì giá trị này ngoài hàm cũng thay đổi theo. Xem ví dụ sau, kết quả in ra sẽ là true.

```
public class Sample {
	public static void main(String args[]) {
		AtomicBoolean a = new AtomicBoolean(false);
		change(a);
		System.out.println(a);
	}
	public static void change(AtomicBoolean a) {
		a.set(true);
	}
}
```

- Thứ hai, sử dụng các class này là thread-safe, tức là nhiều thread có thể đồng thời đọc viết vào instance này, trong khi đó nếu dùng các primitive type như là boolean, int, … thì bạn phải tự tay synchronize và giải quyết xung đột.

CHAPTER 3: I/O, NETWORKING, AND SECURITY UPDATES

I/O, networking, security có gì mới? Một lần nữa, chơi với những thứ mới của I/O không có nghĩa là bạn chơi với package mới (java.nio) (mà thật ra cái này cũng cũ rích chứ đâu, từ Java 4 đã có nio rồi ahihi, mà thôi kệ dù sao so với java.io cũ thì nó nghe cũng có vẻ mới lạ kaka). OK, chúng ta sẽ tiếp tục với những thay đổi trong các package java.io, java.net, java.security

Bảng 3-1. Package java.io.*

Package	Version	Interfaces	Classes	Throwable	Total
io	5.0	12	50	16+0	78
io	6.0	12	51	16+1	80
Delta		0	1	0+1	2

Bảng 3-2. Package java.net.*

Package	Version	Interfaces	Classes	Enum	Throwable	Total
net	5.0	6	34	2	12+0	54
net	6.0	8	38	2	12+0	60
Delta		2	4	0	0+0	6

Bảng 3-3. Package java.security.*

Package	Version	Interfaces	Classes	Enum	Throwable	Total
security	5.0	12	50	1	16+0	79
security	6.0	13	52	1	16+0	82
security.acl	5.0	5	0	0	3+0	8
security.acl	6.0	5	0	0	3+0	8
security.cert	5.0	8	27	0	9+0	44
security.cert	6.0	8	27	0	9+0	44
security.interfaces	5.0	13	0	0	0+0	13
security.interfaces	6.0	13	0	0	0+0	13
security.spec	5.0	3	22	0	2+0	27
security.spec	6.0	3	22	0	2+0	27
Delta		1	2	0	0+0	3

Có thể thấy là những package này dường như rất ít thay đổi, so với cái tụi package trên chapter 2 thì còn ít ới là ít.

I. Package java.io

Ngoại trừ những gì đã thảo luận về class Console ở chapter 2, những thay đổi đáng kể của java.io là cách mình chơi với class File và toURL() bị deprecated. Cụ thể như sau:

Một trong những điểm mới là ở Java 6, bạn có thể lấy dung lượng còn trống của một file partition, điều mà trước đây bạn không thể làm, sử dụng getUsableSpace() và getTotalSpace().

```
import java.io.*;

public class Space {

	public static void main(String args[]) {

		Console console = System.console();

		File roots[] = File.listRoots();

		for (File root: roots) {

			console.printf("%s has %,d of %,d free%n",
			root.getPath(),

			root.getUsableSpace(), root.getTotalSpace());

		}

	}

}
```

```
Output:

A:\ has 30,720 of 730,112 free

C:\ has 5,825,671,680 of 39,974,860,288 free

D:\ has 51,128,320 of 100,431,872 free

E:\ has 0 of 655,429,632 free

F:\ has 0 of 0 free

G:\ has 19,628,294,144 of 40,047,280,128 free

H:\ has 347,922,432 of 523,993,088 free

I:\ has 248,061,952 of 255,827,968 free
```

Để access vào những available file system space, Mustang cũng thêm vào một số method mới để làm việc với các tác vụ read/write/execute. Trước Java 6, chỉ có canRead() và canWrite(), bây giờ có thêm canExecute() và 3 setter là setReadable(), setWritable(), and setExecutable(). Mỗi setter method này đều có 2 version, version thứ 1 chỉ cần 1 boolean parameter (quá dễ hiểu, chỉ là true false thôi mà). Version thứ 2 cần 2 parameter, để support cho những file system hỗ trợ phân quyền riêng biệt cho owner và non-owner.

Thay đổi còn lại của class File chính là method toURL() bị deprecated. Vậy giờ làm sao để lấy url đây? Đơn giản là lấy qua URI:

```
URL url = aFile.toURL();
```

Sẽ thành:

```
URL url = aFile.toURI().toURL();
```

II. Package java.nio

Package này hầu như không có thay đổi đáng chú ý.

III. Package java.net

Sự thay đổi của package java.net trong Java 6 hướng đến việc xử lý cookie, Java 5 có CookieHandler, Java 6 thêm interface CookiePolicy và CookieStore với 2 class CookieManager và HttpCookie.

Đã bàn về cookie thì phải bàn về HTTP, chúng lưu dữ liệu lên hệ thống của bạn. Những dữ liệu này giúp remote system ghi nhớ một vài thông tin của bạn khi truy cập lại.

J2SE 5.0 CookieHandler là một abstract class, không có implementation, không có cơ chế hay chính sách lưu trữ, và không có gì để lưu luôn (nói chung là chẳng có gì ngoài cái xác khô abstract class). OK now, Java 6 comes in, class CookieManager chính là implementation của CookieHandler, CookieStore là cơ chế lưu trữ, CookiePolicy là chính sách accept hay reject cookies, và cuối cùng là HttpCookie chính là thứ mà được lưu trữ. Mọi thứ có vẻ được hỗ trợ tốt hơn trước, bạn không phải tự mình gồng gánh tất cả để lưu cookies. Tuyệt vời!

Một sự khác biệt nữa là ở Java 6, cache(CookieStore) sẽ chơi với việc cookies hết hạn hay không. Trước đây đó là trách nhiệm của handler (CookieManager), bây giờ thì handler chỉ việc bảo cache lưu cái gì đó, còn không chơi với hết hạn hay không hết hạn nữa.

Với CookiePolicy, bạn có thể define một custom policy khi chơi với cookie hoặc có thể bảo CookieManager sử dụng một trong những policy có sẵn là ACCEPT_ALL, ACCEPT_NONE, hoặc ACCEPT_ORIGINAL_SERVER, cái thứ 3 sẽ từ chối những cookie của third-party, chỉ chấp nhận những cookie đến từ original server – cùng server với response.

Tạm thời ngưng ở đây, ai muốn tìm hiểu thêm thì tự thân vận động nhé.

IV. Package java.security

Bảng 3.3 đã cho thấy package này không thay đổi lắm.

Class Policy có thêm 2 marker interface là Policy.Parameters và Configuration.

Parameters, những marker này được implement bởi class URIParameter để gom URI với Policy và Configuration. Những cái thứ này được sử dụng nội bộ bởi class PolicySpi và ConfigurationSpi.

CHAPTER 4: AWT AND SWING UPDATES

Cái này là làm về giao diện nên chắc không dùng tới, khỏi làm luôn.

CHAPTER 5: JDBC 4.0

Cái này là về database. Phiên bản JDBC đầu tiên được giới thiệu từ JDK 1.1, hỗ trợ việc kết nối chương trình Java với database SQL. Một cái dở của JDBC là loading database driver, Java 6 khắc phục điều này bằng cách cho phép access kết nối JDBC mà không cần load driver.

Bảng 5-1. Package java.sql.*

Package	Ver	Interface	Class	Enum	Throwable	Annotation	Total
sql	5.0	18	7	0	0+4	0	29
sql	6.0	27	8	3	0+19	4	61
Delta		9	1	3	0+15	4	32

Bảng 5-2. Package javax.sql.*

Package	Ver	Interface	Class	Throwable	Total
sql	5.0	12	2	0+0	14
sql	6.0	14	3	0+0	17
sql.rowset	5.0	7	2	0+1	10
sql.rowset	6.0	7	2	0+1	10
sql.rowset.serial	5.0	0	9	0+1	10
sql.rowset.serial	6.0	0	9	0+1	10
sql.rowset.spi	5.0	4	2	0+2	8
sql.rowset.spi	6.0	4	2	0+2	8
Delta		2	1	0+0	3

Có nhiều thay đổi khi nói về JDBC 4.0

I. The java.sql and javax.sql Packages

java.sql là package chính của JDBC, cung cấp những class để tương tác với data source. Vì những thay đổi của package javax.sql là rất nhỏ nên chúng ta gom nguyên 2 package này vào chung một phần. OK lego, có các điểm chính như sau:

- Database driver loading
- Exception handling improvements
- Enhanced BLOB/CLOB functionality
- Connection and statement interface enhancements
- National character set support
- SQL ROWID access
- SQL 2003 XML data type support
- Annotations

1. Database driver loading

Mustang thay đổi cách load/register database driver, tạo một subdirectory tên là services trong META-INF cho file jar, đặt vô đó một text file để service provider có thể tìm thấy, Java runtime environment sẽ tự tìm đến khi được yêu cầu. Đây chính là điểm mới của Java 6 nhằm hỗ trợ cho việc load JDBC driver, bạn không cần phải tự mình load driver nữa.

Trước Java 6, để load JDBC driver:

```
Class.forName("oracle.jdbc.driver.OracleDriver");

// or DriverManager.registerDriver(new
oracle.jdbc.driver.OracleDriver());

Connection con = DriverManager.getConnection(url, username, password);
```

Bây giờ bạn có thể bỏ dòng đầu tiên và dùng trực tiếp driver

```
Connection con = DriverManager.getConnection(url, username, password);
```

Một câu hỏi được đặt ra, điều gì sẽ xảy ra trong trường hợp có nhiều driver cho một loại kết nối database? Việc này sẽ được xử lý ở folder CLASSPATH. FIle JAR đầu tiên sẽ match connection từ DriverManager, giống như lúc gọi Class.forName()

2. Exception handling improvements

JDBC 4.0, xử lý exception được improve ở 3 điểm chính.

a. Có thể sử dụng for loop để lặp cho exception

Để hỗ trợ cho for loop, class SQLException bây giờ hiện thực Iterable<T>, với T là một Throwable.

```
try {
	...
} catch (SQLException sqle) {
	for(Throwable t : sqle) {
		System.out.println("Throwable: " + t);
	}
}
```

b. SQLException có thêm constructor mới truyền vào nguyên nhân exception

- `SQLException(Throwable cause).`
- `SQLException(String reason, Throwable cause).`
- `SQLException(String reason, String sqlState, Throwable cause).`
- `SQLException(String reason, String sqlState, int vendorCode, Throwable cause).`

c. Có thêm subclass cho SQLException

Transient exception (Exception tạm thời) là những exception mà khi retry có thể thành công mặc dù không có gì thay đổi:

- SQLTimeoutException
- SQLTransactionRollbackException
- SQLTransientConnectionException

Nontransient exception là những exception mà khi retry thì sẽ tiếp tục fail nếu nguyên nhân chưa được khắc phục:

- SQLDataException
- SQLFeatureNotSupportedException
- SQLIntegrityConstraintViolationException
- SQLInvalidAuthorizationSpecException
- SQLNonTransientConnectionException
- SQLSyntaxErrorException

3. Enhanced BLOB/CLOB functionality

BLOBs and CLOBs nghe vui tai thiệt, chúng đại diện cho những object binary/character lớn được lưu trong database. Trước JDBC 4.0, một số chỗ trong API vẫn còn mơ hồ. Mustang khắc phục điều này, ví dụ như, thay vì gọi setCharacterStream(int parameterIndex, Reader reader, int length) trên một object PreparedStatement và để **system tự quyết định** column là LONGVARCHAR hay CLOB, bây giờ bạn có thể gọi trực tiếp setClob(int parameterIndex, Reader reader, long length).

Một số thay đổi khác như thêm method để tạo empty object trong class Connection: createBlob(), createClob(), createNClob().

4. Connection and statement interface enhancements

Connection và Statement là 2 interface quan trọng của JDBC, Connection mô tả một database session, Statement là một SQL statement để lấy về ResultSet. Hai interface này có thêm một chút thay đổi như sau

a. Connection

Interface này có 2 thay đổi chính, cover bằng 5 method:

- Thứ nhất: Kiểm tra xem một connection đã close hay chưa
  public boolean isValid(int timeout): timeout là số giây để chờ reply, nếu không có reply thì return false, nếu truyền 0 thì sẽ chờ mãi mãi.
- Thứ hai: Có thể get và set connection's client info properties
  public Properties getClientInfo()
  public String getClientInfo(String name)
  public void setClientInfo(Properties props)
  public void setClientInfo(String name, String value)

b. Statement

Statement class nay có thêm method isClosed() để kiểm tra xem statement đã close chưa.

PreparedStatement cho phép lưu giữ những SQL statement ở dạng pre-compile, do đó nâng cao hiệu suất khi statement được chạy nhiều lần. Interface PreparedStatement có thêm 2 method isPoolable() và setPoolable(), bạn có thể yêu cầu một PreparedStatement trở thành pooled hoặc không, pooled statement là statement có thể được share cho nhiều connection bởi statement pool manager.

Ở package javax.sql có một interface là PooledConnection, thằng cu này bây giờ đã cho register một StatementEventListener vào connection, sẽ được notify khi prepared statement close hoặc invalid.

5. National character set support

JDBC 4.0 hỗ trợ một số set type mới: NCHAR, NVARCHAR, LONGNVARCHAR, and NCLOB, ký tự N đại diện cho national character set.

Một số interface cũng được thay đổi để chơi với t national character set, PreparedStatement và CallableStatement có thêm setNString(), setNCharacterStream(), và setNClob(). ResultSet có getNString(), getNCharacterStream(), và getNClob(), updateNString(), updateNCharacterStream(), và updateNClob().

6. SQL ROWID access

Một feature mới của JDBC 4.0 là support cho access SQL built-in type ROWID, phải chú ý một điều rằng cái này còn phụ thuộc vào database ở dưới có cho phép lấy rowid hay không nữa, để làm được điều này, phải đi hỏi DatabaseMetaData, method getRowIdLifetime() của nó trả về một getRowIdLifetime() chứa 1 trong những enum:

- ROWID_UNSUPPORTED: data source không hỗ trợ feature rowid.
- ROWID_VALID_FOREVER: forever.
- ROWID_VALID_SESSION: at least the session.
- ROWID_VALID_TRANSACTION: transaction.
- ROWID_VALID_OTHER: có thể lấy rowid nhưng không chắc bao giờ nó die.

7. SQL 2003 XML data type support

Một feature bự được add vào SQL 2003 là support cho XML như là native type. Ở Java, bạn cũng không cần sử dụng CLOBs để access vào data nữa, JDBC 4.0 mapping trực tiếp vào SQL XML type.

Khi query vào một XML column, ResultSet sẽ trả về kiểu SQLXML, interface SQLXML gồm 9 method:

- `public void free()`
- `public InputStream getBinaryStream()`
- `public Reader getCharacterStream()`

- public <T extends Source> T getSource(Class<T> sourceClass)
- public String getString()
- public OutputStream setBinaryStream()
- public Writer setCharacterStream()
- public <T extends Result> T setResult(Class<T> resultClass)
- public void setString(String value)

8. Annotations

Chapter 10 sẽ thảo luận về Annotation, nhưng một số annotation về JDBC sẽ được đề cập ở đây, bao gồm 4 annotation mới là Select, Update, ResultColumn và AutoGeneratedKeys.

Ví dụ, define class Student như sau để đổ data từ database vào

```
public class Student {
	public int id;
	public String first;
	public String last;
}
```

Dưới database sẽ có các cột cùng tên là id, first, last. Nhưng nếu bạn muốn đổ cột last vào class Student với tên là lastName thì hãy dùng annotation như sau

```
@ResultColumn("last") String lastName
```

Một ví dụ khác về sử dụng Select, Update

```
interface MyQueries extends BaseQuery {
	@Select("select id, first, last from students")
	DataSet<Student> getAllStudents();
	@Update("delete * from students")
	int deleteAllStudents();
}
```

CHAPTER 6: EXTENSIBLE MARKUP LANGUAGE (XML)

Cái này chắc không cần phải làm.

CHAPTER 7: WEB SERVICES

Là những thành phần ứng dụng dùng để chuyển đổi một ứng dụng thông thường sang một ứng dụng web. Đồng thời nó cũng xuất bản các chức năng của mình để mọi người dùng internet trên thế giới đều có thể sử dụng thông qua nền tảng web. Những package liên quan đến web service trong Java 6 đều là mới nên không cần phải làm bảng so sánh giữa 2 version 5.0 và 6.0. Có tất cả 3 package, javax.xml.ws chứa JAX-WS API, javax.xml.soap chứa SAAJ (SOAP with Attachments API for Java) và Web Services Metadata nằm trong javax.jws. Chapter này tui chỉ nói sơ qua thôi.

I. Package javax.jws (Web Services Metadata)

Hai package liên quan là javax.jws và javax.jws.soap, cả 2 đều define enumeration và annotation, không có class và interface nào. Import những package này xong thì bạn có thể annotate những class represent service cũng như method của chúng.

```
package net.zukowski.revealed;
import javax.jws.WebService;
import javax.jws.WebMethod;
@WebService
public class HelloService {
	@WebMethod
	public String helloWorld() {
		return "Hello, World";
	}
}
```

II. Packages javax.xml.ws and javax.xml.soap

Nhắc lại: javax.xml.ws chứa JAX-WS API, javax.xml.soap chứa SAAJ

JAX-WS API khá là giống với JAXB. Trong khi JAXB là Java-to-XML mapping (và ngược lại: XML to Java) thì JAX-WS mapping Java object từ/sang Web Services Description Language (WSDL), cùng với SAAJ, bộ ba tạo ra API cho web service.

1. SOAP message

SOAP là một giao thức giao tiếp có cấu trúc như XML và mã hóa thành định dạng chung cho các ứng dụng trao đổi với nhau

javax.xml.soap cung cấp MessageFactory, bạn có thể get instance và tạo message

```
SOAPMessage soapMessage = MessageFactory.newInstance().createMessage();
```

Cái này sẽ tạo SOAP 1.2 message, để tạo SOAP 1.1 message:

```
SOAPMessage soapMessage =

MessageFactory.newInstance().createMessage(SOAPConstants.SOAP_1_1_PROTOCO
L);
```

SOAP message bao gồm SOAPPart, SOAPEnvelope, SOAPBody và SOAPHeader, mấy đứa này đều dạng XML. Nếu muốn include non-XML data vô thì phải dùng AttachmentPart

```
AttachmentPart attachment = soapMessage.createAttachmentPart();

attachment.setContent(textContent, "text/plain");

soapMessage.addAttachmentPart(attachment);
```

Những gì bạn có thể bỏ vô SOAP message còn tùy thuộc vào service đang kết nối vào, cứ tạo các thành phần nhỏ riêng biệt, sau đó put them together. Theo javadoc thì đây là những điều mà javax.xml.soap có thể làm:

- Create a point-to-point connection to a specified endpoint
- Create a SOAP message
- Create an XML fragment
- Add content to the header of a SOAP message
- Add content to the body of a SOAP message
- Create attachment parts and add content to them
- Access/add/modify parts of a SOAP message
- Create/add/modify SOAP fault information
- Extract content from a SOAP message
- Send a SOAP request-response message

2. The JAX-WS API

JAX-WS = Java API for XML Web Services

CHAPTER 8: THE JAVA COMPILER API

Bạn không cần phải gọi class Main của package com.sun.tools.javac để compile source nữa, bây giờ bạn có thể access compiler thông qua package javax.tools mà không cần thêm bất cứ file JAR nào vào classpath

Package	Ver	Interface	Class	Enum	Total
tools	6.0	11	6	3	20

I. Compiling Source, Take 1

API này cung cấp một số option để compile source. Đầu tiên, hãy xem cái cách vừa nhanh vừa bẩn sau đây (quick and dirty). Với cách này, error sẽ được send ra standard error (stderr).

Để invoke Java compiler từ Java program, bạn cần phải access vào interface JavaCompilerTool. Bằng cách này bạn có thể set source path, class path, destination directory.

Để access vào JavaCompilerTool mặc định, bạn cần gọi ToolProvider.getSystemJavaCompilerTool(), method này return về một class implements interface JavaCompilerTool.

Cách đơn giản để compile source với JavaCompilerTool là gọi hàm run()

```
int run(InputStream in, OutputStream out, OutputStream err, String...
arguments)
```

Ba parameter đầu có thể truyền null, lúc này sẽ dùng System.in, System.out và System.err. Parameter cuối là tên file truyền vô

```
int results = tool.run(null, null, null, "Foo.java");
```

Có 2 cách để xem kết quả của compile

- Kiểm tra file .class cùng tên
- Kiểm tra giá trị int trả về, nếu success thì trả về 0, khác 0 là có lỗi, tìm hiểu javadoc để hiểu thêm các giá trị khác 0 nghĩa là gì.

```
import java.io.*;
import javax.tools.*;
public class FirstCompile {
public static void main(String args[]) throws IOException {
        JavaCompilerTool compiler =
        ToolProvider.getSystemJavaCompilerTool();
        int results = compiler.run(null, null, null, "Foo.java");
                System.out.println("Success: " + (results == 0));
        }
}
```

Chạy chương trình, trường hợp không có file Foo.java sẽ có lỗi như sau

```
error: cannot read: Foo.java
1 error
Success: false
```

II. Compiling Source, Take 2

Ví dụ trên compile source để generate file .class, sử dụng các tùy chỉnh mắc định, tương tự như chạy lệnh javac Foo.java bằng command line. Bạn có thể tùy chỉnh để có kết quả tốt hơn.

1. Introducing StandardJavaFileManager

Như đã đề cập ở trên, cách dùng ở phần I. không được recommend. Một cách khác tốt hơn là dùng class StandardJavaFileManager, class này hỗ trợ làm việc với input và output file, cũng như get report message bằng interface DiagnosticListener, class DiagnosticCollector chính là một hiện thực của interface đó.

```
DiagnosticCollector<JavaFileObject> diagnostics =
new DiagnosticCollector<JavaFileObject>();
StandardJavaFileManager fileManager =
compiler.getStandardFileManager(diagnostics);
```

Quăng null parameter cho listener vẫn sẽ ok, nhưng nó sẽ quay về Nơi tình yêu bắt đầu (cái cách dùng ở I.)

Trước khi đi sâu vào class StandardJavaFileManager, hãy ghé lại uống miếng nước rồi xem cái này trước, method JavaCompilerTool. getTask() - một bước quan trọng trong compilation process. Nó nhận 6 argument và trả về 1 instance của innner class của nó (tên là CompilationTask).

```
JavaCompilerTool.CompilationTask getTask(
      Writer out,
      JavaFileManager fileManager,
      DiagnosticListener<? super JavaFileObject> diagnosticListener,
      Iterable<String> options,
      Iterable<String> classes,
      Iterable<? extends JavaFileObject> compilationUnits)
```

Hầu hết argument đều có thể null (thì sẽ dùng mặc định):

- out: System.err
- fileManager: The compiler's standard file manager
- diagnosticListener: The compiler's default behavior
- options: No command-line options given to the compiler
- classes: No class names provided for annotation processing

Argument cuối compilationUnits không nên là null, nó chính là file mà bạn muốn compile. Và nó mang chúng ta trở lại với StandardJavaFileManager, để ý vào Iterable<? extends JavaFileObject>, class StandardJavaFileManager có 2 method trả về kiểu này, nhưng bạn cũng có thể truyền vào List<File> hoặc List<String>

```
Iterable<? extends JavaFileObject> getJavaFileObjectsFromFiles(
Iterable<? extends File> files)
Iterable<? extends JavaFileObject> getJavaFileObjectsFromStrings(
Iterable<String> names)
```

OK bây giờ bạn có thể chạy compile được rồi. Method getTask() lúc nãy return 1 instance của JavaCompilerTool.CompilationTask, instance này implement Runnable.

```
JavaCompilerTool.CompilationTask task =
compiler.getTask(null, fileManager, null, null, null, compilationUnits);
task.run();
```

Gọi task.getResult() để lấy kết quả, method này return true nếu không có error

Cuối cùng, gọi fileManager.close() để release resource.

Đoạn code hoàn chỉnh sẽ như sau

```
import java.io.*;
import java.util.*;
import javax.tools.*;
public class SecondCompile {
	public static void main(String args[]) throws IOException {
		JavaCompilerTool compiler =
		ToolProvider.getSystemJavaCompilerTool();
		DiagnosticCollector<JavaFileObject> diagnostics =
		new DiagnosticCollector<JavaFileObject>();
		StandardJavaFileManager fileManager =
		compiler.getStandardFileManager(diagnostics);
		Iterable<? extends JavaFileObject> compilationUnits =
		fileManager.getJavaFileObjectsFromStrings(Arrays.asList("Foo.j
		ava"));
		JavaCompilerTool.CompilationTask task = compiler.getTask(
		null, fileManager, diagnostic, null, null, compilationUnits);
		task.run();
		boolean success = task.getResult();
		APIfileManager.close();
		System.out.println("Success: " + success);
	}
}
```

2. Working with DiagnosticListener

Compilation errors sẽ được report vào DiagnosticListener đã đăng ký trước đó, interface này chỉ có 1 method mà bạn phải implement là

```
public void report(Diagnostic<? extends S> diagnostic)
```

Thực tế thì bạn cũng không cần phải tự implement nó, thư viện đã cung cấp sẵn một implementation là class DiagnosticCollector, việc còn lại chỉ là loop

```
for (Diagnostic diagnostic : diagnostics.getDiagnostics())
        System.console().printf(
        "Code: %s%n" +
        "Kind: %s%n" +
        "Position: %s%n" +
        "Start Position: %s%n" +
        "End Position: %s%n" +
        "Source: %s%n" +
        "Message: %s%n",
        diagnostic.getCode(), diagnostic.getKind(),
        diagnostic.getPosition(), diagnostic.getStartPosition(),
        diagnostic.getEndPosition(), diagnostic.getSource(),
        diagnostic.getMessage(null));
```

3. Changing the Output Directory

Để thay đổi thư mục xuất của file compile (ví dụ như file .class), bạn thêm parameter vào method getTask(), tương tự như khi dùng **javac –d classes** ở command line

```
Iterable<String> options = Arrays.asList("-d", "classes");
JavaCompilerTool.CompilationTask task = compiler.getTask(
null, fileManager, diagnostics, options, null, compilationUnits)
```

Cách này sẽ xuất file compile vào thư mục classes.

4. Changing the Input Directory

Class JavaFileObject dùng để xác định source để compile, sử dụng method getJavaFileObjectsFromStrings() hoặc getJavaFileObjectsFromFiles() của class StandardJavaFileManager truyền vào list (String hoặc File) sẽ trả về list JavaFileObject, quăng vào hàm getTask() ở trên để compile.

Trường hợp 1 class cần class khác để compile thì system sẽ tìm nó trong cùng thư mục (or at least relative to the top-level package directory).

Nếu dùng command line để compile, bạn có thể cung cấp thêm 1 vị trí cho compiler để tìm các file cần thiết khi compile sử dụng option –sourcepath. Với class JavaCompilerTool, bạn chỉ cần quăng thêm argument cho method getTask()

```
Iterable<String> options = Arrays.asList("-d", "classes", "-sourcepath",
"src");
```

III. Compiling from Memory

JavaCompilerTool không chỉ cho phép compile source trên ổ đĩa, class này còn cho phép bạn tạo file trên memory, compile them, run them. Java 6 cung cấp class JavaSourceFromString để hỗ trợ cho việc này. For more info, Google it yourself please.

CHAPTER 9: SCRIPTING AND JSR 223

Java 6 đã hỗ trợ scripting, lần đầu nghe thấy điều này tui cứ nghĩ rằng Rhino JavaScript interpreter đã được embeded vào Java platform.

Rhino is an open-source implementation of JavaScript written entirely in Java. It is typically embedded into Java applications to provide scripting to end users.

Nhưng thật ra không phải như vậy, Mustang thêm một interface để tích hợp scripting language (PHP, Ruby, Javascript), một framework để các ngôn ngữ script đó có thể access Java platform, và một command-line scripting shell program, jrunscript.

Bảng 9-1. Package javax.script.*

Package	Ver	Interface	Class	Throwable	Total
script	6.0	6	5	0+1	12

I. Scripting Engines

Package script được thêm vào Java 6 tương đối nhỏ, gồm 6 interface, 5 class, 1 exception. Class chính của package này là ScriptEngineManager, nó cung cấp cơ chế để thiết lập ScriptEngineFactory object, object này cho phép access và ScriptEngine. Ví dụ bên dưới sẽ cho thấy mối quan hệ giữa ScriptEngineManager, ScriptEngineFactory và ScriptEngine.

```
import javax.script.*;
import java.io.*;
import java.util.*;
public class ListEngines {
	public static void main(String args[]) {
		ScriptEngineManager manager = new ScriptEngineManager();
		List<ScriptEngineFactory> factories =
		manager.getEngineFactories();
		for (ScriptEngineFactory factory: factories) {
			Console console = System.console();
			console.printf("Name: %s%n" +
			"Version: %s%n" +
			"Language name: %s%n" +
			"Language version: %s%n" +
			"Extensions: %s%n" +
			"Mime types: %s%n" +
			"Names: %s%n",
			factory.getEngineName(),
			factory.getEngineVersion(),
			factory.getLanguageName(),
			factory.getLanguageVersion(),
			factory.getExtensions(),
			factory.getMimeTypes(),
			factory.getNames());
			ScriptEngine engine = factory.getScriptEngine();
		}
	}
}
```

```
Output:

Name: Mozilla Rhino

Version: 1.6 release 2

Language name: ECMAScript

Language version: 1.6

Extensions: [js]

Mime types: [application/javascript, application/ecmascript,
text/javascript,

text/ecmascript]

Names: [js, rhino, JavaScript, javascript, ECMAScript, ecmascript]
```

Hàng cuối cùng là những cái tên dùng để xác định engine từ manager.

Với mỗi factory lấy từ manager, bạn có thể gọi method getScriptEngine() để lấy ScriptEngine. Nhưng cũng có thể lấy ScriptEngine trực tiếp từ manager

```
ScriptEngine engine1 = manager.getEngineByExtension("js");

ScriptEngine engine2 = manager.getEngineByMimeType("text/javascript");

ScriptEngine engine3 = manager.getEngineByName("javascript");
```

Để ScriptEngine chạy expression, bạn sẽ phải dùng 1 trong 6 version của method eval(), những method này có thể quăng ScriptException nếu có error.

```
public Object eval(String script)

public Object eval(Reader reader)

public Object eval(String script, ScriptContext context)

public Object eval(Reader reader, ScriptContext context)

public Object eval(String script, Bindings bindings)

public Object eval(Reader reader, Bindings bindings)
```

Ở trên, script có thể là String hoặc Reader stream.

ScriptContext cho phép bạn xác định scope cho các Binding object, có 2 scope được tạo sẵn là ScriptContext.GLOBAL_SCOPE và ScriptContext.ENGINE_SCOPE.

Binding object giúp mang Java object sang scripting world và ngược lại.

Sau đây là ví dụ chạy javascript từ Java:

```
import javax.script.*;
import java.io.*;
public class RunJavaScript {
	public static void main(String args[]) {
		ScriptEngineManager manager = new ScriptEngineManager();
		ScriptEngine engine = manager.getEngineByName("javascript");
		try {
		Double hour = (Double)engine.eval(
		"var date = new Date();" +
		"date.getHours();");
		String msg;
		if (hour < 10) { msg = "Good morning";}
		else if (hour < 16) { msg = "Good afternoon";}
		else if (hour < 20) { msg = "Good evening";}
		else { msg = "Good night";}
		Console console = System.console();
		console.printf("Hour %s: %s%n", hour, msg);
		} catch (ScriptException e) {
				System.err.println(e);
		}
	}
}
```

Output:

```
Hour 8.0: Good morning
```

Ví dụ tiếp theo sẽ mô phỏng việc sử dụng Binding, như đã đề cập ở trên, Binding giúp mang Java object sang scripting world và ngược lại, bạn có thể get Binding object của ScriptEngine và thao tác mới nó như là Map, interface ScriptEngine cũng hỗ trợ sẵn 2 phương thức get() và put() để làm việc trực tiếp với Binding object của ScriptEngine. Ví dụ sau sẽ nhận argument là một chuỗi từ command line rồi đưa vào JavaScript engine bằng Binding, sau đó JavaScript sẽ đảo ngược chuỗi và trả lại Java bằng Binding.

```
import javax.script.*;
import java.io.*;
public class FlipBindings {
	public static void main(String args[]) {
		ScriptEngineManager manager = new ScriptEngineManager();
		ScriptEngine engine = manager.getEngineByName("javascript");
		if (args.length != 1) {
			System.err.println("Please pass name on command line");
			System.exit(-1);
		}
		try {
			engine.put("name", args[0]);
			engine.eval(
			"var output = '';" +
			"for (i = 0; i <= name.length; i++) {" +
			" output = name.charAt(i) + output" +
			"}");
			String name = (String)engine.get("output");
			Console console = System.console();
			console.printf("Reversed: %s%n", name);
		} catch (ScriptException e) {
			System.err.println(e);
		}
	}
}
```

Kết quả chạy command line:

```
> java FlipBindings "Java 6 Platform Revealed"
Reversed: delaeveR mroftalP 6 avaJ
```

II. The Compilable Interface

Trước tiên nên hiểu về thông dịch và biên dịch, một chương trình viết bằng C++ trên Windows sẽ được biên dịch sang file .exe, sau này chỉ cần file exe để chạy mà không cần biên dịch nữa. Còn đối với thông dịch, mỗi lần chạy chương trình phải chạy lại thông dịch (thông dịch -> thực hiện -> thông dịch -> thực hiên …), việc này làm tăng excecution time. Scripting language sử dụng thông dịch (interpreter) nên sẽ bị dính cái dở này.

Để optimize vấn đề này, Mustang có interface Compilable để compile script. Nếu ScriptEngine implements Compilable, bạn có thể precompile script trước khi excecution.

```
import javax.script.*;
import java.io.*;
public class CompileTest {
	public static void main(String args[]) {
		ScriptEngineManager manager = new ScriptEngineManager();
		ScriptEngine engine = manager.getEngineByName("javascript");
		engine.put("counter", 0);
		if (engine instanceof Compilable) {
		Compilable compEngine = (Compilable)engine;
		try {
			CompiledScript script = compEngine.compile(
			"function count() { " +
			" counter = counter +1; " +
			" return counter; " +
			"}; count();");
			Console console = System.console();
			console.printf("Counter: %s%n", script.eval());
			console.printf("Counter: %s%n", script.eval());
			console.printf("Counter: %s%n", script.eval());
		} catch (ScriptException e) {
			System.err.println(e);
		}
		} else { System.err.println("Engine can't compile code");}
	}
}
```

Kết quả chạy:

```
> java CompileTest
Counter: 1.0
Counter: 2.0
Counter: 3.0
```

III. The Invocable Interface

Invocable là một interface khác mà ScriptEngine có thể implements, cho phép gọi function trong script language. Không những thế, nó còn cho phép bạn bind function trong script language cho interface của Java.

Invocable cung cấp method invoke() để gọi hàm trong script language, bạn cũng có thể truyền parameter khi invoke.

```
import javax.script.*;
import java.io.*;
public class InvocableTest {
	public static void main(String args[]) {
		ScriptEngineManager manager = new ScriptEngineManager();
		ScriptEngine engine = manager.getEngineByName("javascript");
		if (args.length == 0) {
			System.err.println("Please pass name(s) on command
			line");
			System.exit(-1);
		}
		try {
			engine.eval(
			"function reverse(name) {" +
			" var output = '';" +
```

```
                    " for (i = 0; i <= name.length; i++) {" +
                    " output = name.charAt(i) + output" +
                    " }" +
                    " return output;" +
                    "}");
                    Invocable invokeEngine = (Invocable)engine;
                    Console console = System.console();
                    for (Object name: args) {
                          Object o = invokeEngine.invoke("reverse", name);
                          console.printf("%s / %s%n", name, o);
                    }
              } catch (NoSuchMethodException e) {
                    System.err.println(e);
              } catch (ScriptException e) {
                    System.err.println(e);
              }
        }
}
```

Kết quả chạy:

```
> java InvocableTest one two three
one / eno
two / owt
three / eerht
```

Ví dụ trên demo cho feature rằng Java có thể gọi method của script. Sau đây sẽ là giới thiệu về implement interface sử dụng method của script, sử dụng method getInterface() của interface Invocable.

Runnable là interface chỉ có 1 method là run(), nếu scripting language của bạn đã có method run(), bạn có thể dùng nó để implement run() cho Runnable.

Đầu tiên tạo hàm run() cho script:

```
engine.eval("function run() {print('wave');}");
```

Sau đó liên kết nó với instance của interface Runnable:

```
Runnable runner = invokeEngine.getInterface(Runnable.class);
```

Bây giờ bạn có thể pass runner cho Thread constructor:

```
Thread t = new Thread(runner);
t.start();
```

Source code đầy đủ:

```
import javax.script.*;
public class InterfaceTest {
	public static void main(String args[]) {
		ScriptEngineManager manager = new ScriptEngineManager();
		ScriptEngine engine = manager.getEngineByName("javascript");
		try {
			engine.eval("function run() {print('wave');}");
			Invocable invokeEngine = (Invocable)engine;
			Runnable runner =
			invokeEngine.getInterface(Runnable.class);
			Thread t = new Thread(runner);
			t.start();
			t.join();
		} catch (InterruptedException e) {
			System.err.println(e);
		} catch (ScriptException e) {
			System.err.println(e);
		}
	}
```

IV. Jrunscript

Mustange có add thêm một vài chương trình vào thư mục bin của JDK, một trong số đó là jrunscript, nó như là một cách để access vào scripting engine bằng command line. Bạn có thể chạy những gì truyền vào hàm eval() ở phần trước.

Để biết engine nào đang được cài đặt, chạy command sau:

```
> jrunscript -q
```

Language ECMAScript 1.6 implemention "Mozilla Rhino" 1.6 release 2

Trường hợp có nhiều engine được install, để swich sang một engine thì hãy sử dụng thêm option –l truyền theo sau là tên của engine như là **js, rhino,**

JavaScript, javascript, ECMAScript, hoặc **ecmascript.** Những tên này là case sensitive:

```
> jrunscript -l javascripT
```

script engine for language javascripT can not be found

Nếu mọi thứ OK thì bạn sẽ vào giao diện script language

```
> jrunscript
```

```
js>
```

V. Get Your Pnuts Here

Javascript không phải là script engine có sẵn duy nhất, nhưng nó là script engine duy nhất được ship cùng Mustang. Pnuts cũng là 1 engine có thể làm việc với JSR233. For more information, visit https://pnuts.dev.java.net

CHAPTER 10: PLUGGABLE ANNOTATION & PROCESSING UPDATES

Annotation là khái niệm được giới thiệu ở Java 5.0 và JSR 175. Vì annotation là khá mới nên trong phần này chúng ta không chỉ tìm hiểu những cái mới, mà sẽ tìm hiểu thêm khái niệm và cách dùng chúng.

Đầu tiên, apt là viết tắt của annotation processing tool, là một command tool mới được thêm từ JDK 5.0. Bạn sử dụng annotation để annotate source code, apt để tạo annotation mới.

Trước khi tìm hiểu sâu vào annotation, cần ghé qua javadoc một chút, trong đó có ghi "Typical application programmers will never have to define an annotation type". Thật ra thì tạo mới một annotation rất khác so với sử dụng chúng, do đó chúng ta hãy xem qua cách dùng trước.

Đây là cái ruột của một annotation

```
package java.lang;
import java.lang.annotation.*;
@Documented
@Retention(RetentionPolicy.RUNTIME
public @interface Deprecated {
}
```

I. JDK 5.0 Annotations

Java 5.0 giới thiệu 3 annotation là @Deprecated, @SuppressWarnings và @Override. Hãy xem qua chúng trước khi đến với JDK 6.0 annotations

1. The @Deprecated Annotation

@Deprecated trong Java 5 khác hoàn toàn với tag @deprecated trong javadoc.
@Deprecated được đặt trước method hoặc class để chỉ rằng nó đã lỗi thời (out of date)

```
public class Dep {
	/**
	* @deprecated Don't use this method any more.
	*/
	@Deprecated
	public static void myDeprecatedMethod() {
		System.out.println("Why did you do that?");
	}
}
class DeprecatedUsage {
	public void useDeprecatedMethod() {
		Dep.myDeprecatedMethod();
	}
}
```

Class DeprecatedUsage ở trên sử dụng method myDeprecatedMethod() của class Dep đã được đánh dấu @Deprecated. Khi compile source với javac compiler thì sẽ có warning là

```
> javac Dep.java
Note: Dep.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details
```

2. The @SuppressWarnings Annotation

Có 2 loại annotation, một loại nhận argument (như @SuppressWarnings) và một loại không (như @Deprecated).

@SuppressWarnings cho phép bạn bỏ quả warning của một type xác định. Type này tùy thuộc vào nhà cung cấp compiler, đối với compiler của Sun, có 2 loại warning có thể bỏ qua là "deprecation" và "unchecked". Type "unchecked" dùng cho những warning chung, nếu bạn không muốn sửa code để tránh những warning chung chung thì hãy thêm đoạn annotation bên dưới

@SuppressWarnings({"unchecked"})

Argument của annotation là một mảng string, trường hợp bạn muốn bỏ qua warning cho method bị deprecated như ở ví dụ trên, hãy dùng

@SuppressWarnings({"deprecation"})

```
class DeprecatedUsage {
	@SuppressWarnings("deprecation")
	public void useDeprecatedMethod() {
		Dep.myDeprecatedMethod();
	}
}
```

3. The @Override Annotation

@Override thông báo cho compiler biết rằng method phía sau sẽ override method trong superclass, compiler sẽ la lên nếu superclass không có class đó để override. Cách này giúp tránh khỏi một số trường hợp lỗi đánh máy, như là hashcode() và hashCode()

II. JDK 6.0 Annotations

1. New Annotations

a. The java.beans Package

@ConstructorProperties

b. The java.lang Package

@Deprecated, @Override, @SuppressWarnings.

c. The java.lang.annotation Package

Package này chứa 4 annotation giúp tạo annotation:

Documented, Inherited, Retention, Target

d. The java.sql Package

@AutoGeneratedKeys, @ResultColumn, @Select, @Update

e. The javax.annotation Package

Generated, InjectionComplete, PostConstruct, PreDestroy, Resource, Resources

f. The javax.annotation.processing Package

SupportedAnnotationTypes, SupportedOptions, SupportedSourceVersion

g. The javax.management Package

DescriptorKey, MXBean

h. The javax.xml.bind.annotation Package

Annotation trong này dùng để customize Java element to XML schema mapping, chứa 29 annotation.

i. The javax.xml.bind.annotation.adapters Package

XmlJavaTypeAdapter, XmlJavaTypeAdapters

j. The javax.xml.ws Package

BindingType, RequestWrapper, ResponseWrapper, ServiceMode, WebEndpoint, WebFault, WebServiceClient, WebServiceProvider, WebServiceRef.

2. Annotation Processing

Tạm thời kết thúc tại đây :D