# 1. Setup

User and pass cho 20.194.10.15 để lấy code:

cscguest/8S3Pt24s~

Địa chỉ IBM để tìm hiểu lệnh:

<http://www.ibm.com/support/knowledgecenter/ssw_ibm_i_71/cl/call.htm>

Địa chỉ a.Hoang kiu tìm hiểu:

<http://www.go4as400.com/>

Download Spring Tool Suite ở trang này:

<https://spring.io/tools/sts/all>

Cài java core 7 và 8 (jdk 1.7 và jdk 1.8)

Workspace cho STS:



Mình mở STS ở directory này:  
C:\Users\tnguyen443\Desktop\Software tu down\spring-tool-suite-3.8.2.RELEASE-e4.6.1-win32-x86\_64\sts-bundle\sts-3.8.2.RELEASE

Trong file doc anh Hoang Dang co gui 1 chương trình cài đặt sau:



## - 2 tool trong STS

A Hoàng kiu setup 2 cái tool này để làm việc (hình như là 1 dạng SVN cho eclipse):



## - Tìm hiểu FileZilla

**FileZilla Client** (còn được gọi tắt **là FileZilla**) **là** [trình khách FTP] tự do, mã nguồn mở, đa nền tảng. Sẵn dùng cho Windows, Linux, và Mac OS X. Hỗ trợ FTP, SFTP, và FTPS. Tính đến 5 tháng 3 2009, **FileZilla**đứng thứ 5 trong số những phần mềm được tải về nhiều nhất từ SourceForge.net.

FTP là File Transfer Protocol: giao thức chuyển nhượng tập tin. Port mặc định là 21. Tuy Control Panel có giao thức transfer khác, nhưng người ta cũng dùng giao thức này, thường dùng để transfer data giữa user và host.

## Cài đặt BeyondCompare 4:



## Cài đặt lại jdk7 (jdk rộng hơn jre, vì trong máy có sẵn jre nên mình muốn setup lại jdk):



# 2. Các địa chỉ học Spring:

<https://www.tutorialspoint.com/spring/spring_hello_world_example.htm>

# 3. HelloSpring:

## 3.1 Giới thiệu:

Đây là project mình thực hiện mô phỏng đầy đủ các chức năng của Spring Framework, từ đây về sau, có gì thắc mắc thì lên Project HelloSpring kiếm trước.

<https://codersontrang.com/2013/01/20/gioi-thieu-ve-spring-framework/>

Spring nổi bật ở tính năng liên kết các thành phần lại với nhau theo cách riêng và dễ quản lý. Spring tạo ra 1 framework dễ xây dựng ứng dụng hơn J2EE.

## 3.2 Thư viện

Các thư viện cài đặt cho ví dụ Spring:



Lib dành cho cache:



Lib dành cho log:



## 3.3 Tìm hiểu về file Beans.xml - beanwiring

Beans.xml có tác dụng liên kết các thành phần trong Spring.

<https://codersontrang.com/2013/01/25/bean-wiring-cac-cach-de-lien-ket-cac-thanh-phan-trong-spring/>

Như đã biết, Spring dùng thẻ **<bean>** để khởi tạo một thành phần. Trong một thành phần này, có thể có nhiều thuộc tính tham chiếu đến khác thành phần khác được quản lý bởi Spring container. Bình thường, nếu như không dùng Spring, một thuộc tính sẽ được khai báo để tham chiếu đến một giá trị/đối tượng nào đó qua contructor hoặc phương thức setXXX() tương ứng với thuộc tính đó. Tương tự, Spring cũng cung cấp đầy đủ các cách khai báo để liên kết các thành phần qua Contructor hoặc các phương thức setter.

…

## 3.4 Properties trong java

Với những thông tin như jdbc driver, URL, port, schema…, thay vì viết trực tiếp như vậy, hoặc đặt thành những hằng số trong code thì ta có thể lưu chúng vào file cấu hình (chẳng hạn databaseConfig.properties).

Properties cũng có key và value tương tự như map, nhưng quan trọng nhất của properties là có thể năng save thành file text và load file text lên.

## 3.5 Tìm hiểu về file Beans.xml – autowiring

<https://codersontrang.com/2013/02/02/bean-autowiring-co-che-tu-dong-lien-ket-cac-thanh-phan-trong-spring/>

Qua bài viết “[Bean Wiring – các cách để liên kết các thành phần trong Spring](http://wp.me/p2A80w-5E)“, chúng ta đã được giới thiệu về các cách cơ bản mà các thành phần được khởi tạo và liên kết với nhau trong Spring thông qua việc khai báo ở file cấu hình (.xml) như thế nào. Như đã biết, chúng ta phải trực tiếp dùng thẻ <constructor-arg> hoặc <property> để chỉ ra thành phần nào sẽ được tham chiếu đến. Spring còn có một cơ chế để giảm bớt đi việc khai báo này mà vẫn đảm bảo các thành phần có thể liên kết với nhau, đó là cơ chế Bean Autowiring. Việc liên kết giữa các thành phần sẽ diễn ra hoàn toàn tự động. Cơ chế tự động được thực hiện bởi Spring container dựa vào một trong 4 cách thức sau:

* **by Name**: Tự động liên kết một thuộc tính đến một thành phần có tên trùng với tên của thuộc tính đó
* **by Type**: Tự động liên kết một thuộc tính đến một thành phần có kiểu khớp với kiểu của thuộc tính đó
* **by Constructor**: Giống như byType, nhưng thay vì tìm đến một thành phần có kiểu khớp với kiểu của tham số trong các phương thức setter thì ở đây sẽ tìm đến một thành phần có kiểu khớp với kiểu của tham số trong constructor.
* **by AutoDetect**: Tự động thực hiện cơ chế tự động liên kết *by Constructor* trước, nếu không có sự liên kết thỏa mãn, sẽ thực hiện tiếp cơ chế tự động liên kết *by Type*

## 3.6 Cơ chế Cache trong Spring Framework:

### Spring 3

<https://codersontrang.com/2013/03/04/co-che-cache-trong-spring-framework/>

Lưu ý cách này không dùng được spring 4, chỉ có thể áp dụng cho spring 3.

Các thư viện sử dụng:



File ehcache.xml – file này éo biết làm mẹ gì, nhưng có nó sẽ gây ra lỗi:

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<ehcache xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:noNamespaceSchemaLocation=*"http://ehcache.sf.net/ehcache.xsd"*>

<diskStore path=*"java.io.tmpdir"* />

<cache name=*"products"*

maxEntriesLocalHeap=*"100"*

maxEntriesLocalDisk=*"1000"*

eternal=*"false"*

timeToIdleSeconds=*"300"*

timeToLiveSeconds=*"600"*

memoryStoreEvictionPolicy=*"LFU"*

transactionalMode=*"off"*>

<persistence strategy=*"localTempSwap"* />

</cache>

</ehcache>

Lỗi mà cái file ngu ehcache.xml gây ra:



### Spring 4: chuyển qua sử dụng annotation

<http://websystique.com/spring/spring-4-cache-tutorial-with-ehcache/>

Spring4CachingExample2.zip: file này nằm trong doc của project HelloSpring, đây là ví dụ của trang web bên trên, có sử dụng down để load thư viện.

There are many caching solutions available in market today namely EhCahe, Guava Cache, Caffeine Cache, OScache, JBoss Cache,etc.

Caching Annotations:

* @Cacheable : triggers cache population
* @CacheEvict : triggers cache eviction
* @CachePut : updates the cache without interfering with the method execution
* @Caching : regroups multiple cache operations to be applied on a method
* @CacheConfig : shares some common cache-related settings at class-level
* @EnableCaching : Configuration level annotation which enables Caching

#### Các package liên quan trong ví dụ này:

com.websystique.spring.model

com.websystique.spring.service

com.websystique.spring.configuration

**File ProductServiceImpl khi không sử dụng cache:**

**package** com.websystique.spring.service;

**import** org.springframework.stereotype.Service;

**import** com.websystique.spring.model.Product;

@Service("productService")

**public** **class** ProductServiceImpl **implements** ProductService{

@Override

**public** Product getByName(String name) {

showLookupOperation();

**return** **new** Product(name, 100);

}

**public** **void** showLookupOperation(){

**try** {

**long** time = 5000L;

Thread.*sleep*(time);

} **catch** (InterruptedException e) {

**throw** **new** IllegalStateException();

}

}

}

**AppConfig khi chưa xài cache của Spring:**

**package** com.websystique.spring.configuration;

**import** java.util.Arrays;

**import** org.springframework.cache.CacheManager;

**import** org.springframework.cache.concurrent.ConcurrentMapCache;

**import** org.springframework.cache.support.SimpleCacheManager;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.ComponentScan;

**import** org.springframework.context.annotation.Configuration;

@Configuration

@ComponentScan(basePackages = "com.websystique.spring")

**public** **class** AppConfig {

@Bean

**public** CacheManager cacheManager() {

// configure and return an implementation of Spring's CacheManager SPI

SimpleCacheManager cacheManager = **new** SimpleCacheManager();

cacheManager.setCaches(Arrays.*asList*(**new** ConcurrentMapCache("products")));

**return** cacheManager;

}

}

**SampleApplication khi chưa xài cache của Spring:**

**package** com.websystique.spring.configuration;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**import** org.springframework.context.annotation.AnnotationConfigApplicationContext;

**import** org.springframework.context.support.AbstractApplicationContext;

**import** com.websystique.spring.service.ProductService;

**public** **class** SampleApplication {

**private** **static** **final** Logger ***logger*** = LoggerFactory.*getLogger*(SampleApplication.**class**);

**public** **static** **void** main(String[] args) {

AbstractApplicationContext context = **new** AnnotationConfigApplicationContext(AppConfig.**class**);

ProductService service = (ProductService) context.getBean("productService");

***logger***.info("Iphone ->"+service.getByName("Iphone"));

***logger***.info("Iphone ->"+service.getByName("Iphone"));

***logger***.info("Iphone ->"+service.getByName("Iphone"));

((AbstractApplicationContext) context).close();

}

}

Nếu không có cache, phải 5 giây thì chương trình mới lấy được data:



Lib dành cho log:



Lib dành cho cache:



Không hiểu sao cache dùng cho ví dụ này vẫn k xài được.

Trong lúc sử dụng trên trang chủ Spring có nói đến Gradle, cải tiến từ Ant và Maven, khuyên mình đọc sách này: Gradle in Action

### Spring 4: Ví dụ trên trang chủ Spring:

<https://spring.io/guides/gs/caching>

Giới thiệu khái niệm cache trong spring:

<http://dangtritue.blogspot.com/2008/09/xy-dng-h-thng-cache-vi-spring-aop.html>

Thư viện mình dùng maven để down được, chạy được luôn:



Tới đây thì mình đã hoàn thành được thêm và xóa cache trong spring cơ bản. Lưu ý là mình cần xóa cái ehcache.xml đi vì nó cấu hình cái cache. Có lẽ nên tìm hiểu file này sau.

Tới đây thì có các thư viện sau:



## 3.7 autowired sử dụng annotation @autowired:

<https://huongdanjava.com/bean-autowiring-su-dung-autowired-annotation.html>

Mẹ cái địa chỉ ở trên vô dụng vkl, tiếng Việt viết như cờ cờ

Địa chỉ ở đây mới đúng nè:

<https://www.tutorialspoint.com/spring/spring_autowired_annotation.htm>

Sử dụng package com.autowired.annotation

Cái annotation @autowired theo mình thấy nó autowired thành phần nào đó trong 1 class, chứ k phải autowired nguyên cả cái class.

Bonus thêm 1 số annotation:

<http://dodangquan.blogspot.com/2016/01/mot-so-annotation-trong-spring.html>

## 3.8 Bên lề: Slf4j

Đây là cải tiến của log4j. Mình làm ví dụ này trong package slf4j.log

<https://nmhblog.wordpress.com/2010/08/21/java-logging/>

Mặc định là logger.info hay logger.debug sẽ print ra màn hình console. Để có file log ta phải chỉnh cấu hình logback.xml như sau:

<configuration>

<appender name=*"FILE"* class=*"ch.qos.logback.core.FileAppender"*>

<file>slf4j-example.log</file>

<encoder>

<pattern>%date %level [%thread] %logger{10} [%file:%line] %msg%n</pattern>

</encoder>

</appender>

<root level=*"debug"*>

<appender-ref ref=*"FILE"* />

</root>

</configuration>

Sau đó file slf4j-example.log sẽ nằm trong project HelloSpring

Result:



Vì root level mình để debug nên nó sẽ in tất cả mọi thứ ra (debug+info+…)

Nếu mình để root level là info thì nó chỉ log cái logger.info, cái logger.debug thì k được log ra.

Nếu root level là warn thì chương trình hiện tại không print gì cả, vì không có error hay warning gì ở chương trình HelloWorld này cả.

Cách sử dụng file log trong Daito:

<configuration>

<property name=*"thuat.log.file"* value=*"thuat-test.log"*/>

<property name=*"message.pattern"* value=*"%d [%thread] %level %mdc %logger{0} %C %M %ex - %msg%n"*/>

<appender name=*"FILE"*

class=*"ch.qos.logback.core.rolling.RollingFileAppender"*>

<File>${thuat.log.file}</File>

<encoder>

<Pattern>${message.pattern}</Pattern>

</encoder>

<rollingPolicy class=*"ch.qos.logback.core.rolling.TimeBasedRollingPolicy"*>

<FileNamePattern>${thuat.log.file}.%d{yyyy-MM-dd-HH}

</FileNamePattern>

</rollingPolicy>

</appender>

<appender name=*"CONSOLE"* class=*"ch.qos.logback.core.ConsoleAppender"*>

<encoder>

<pattern>${message.pattern}</pattern>

</encoder>

</appender>

<!-- Ví dụ này là version 1, cách làm thông thường -->

<!-- <appender name="FILE" class="ch.qos.logback.core.FileAppender">

<file>slf4j-example.log</file>

<encoder>

<pattern>%date %level [%thread] %logger{10} [%file:%line] %msg%n</pattern>

</encoder>

</appender> -->

<root level=*"debug"*>

<appender-ref ref=*"FILE"* />

</root>

</configuration>

Với TimeBasedRollingPolicy quy định ở đây tối đa cho 1 file log là 1 tiếng.

ConsoleAppender thì để in ra console, mà hình như ở đay chỉ sử dụng command line mới thấy được.

Kết quả:



Trang web rất hay để tìm hiểu về slf4f:

<http://logback.qos.ch/manual/appenders.html>

Trong ví dụ trên ở Daito có xài cái ${project.build.directory}, cái này được define trong file maven pom.xml. Nhưng chưa hiểu lắm cách define

## 3.9 @Component

Ta có thể sử dụng @Component thay thế cho 3 annotation còn lại là @Repository, @Service và @Controller. Spring sẽ cho phép scan tất cả với @Component, nó hoạt động tốt nhưng sẽ không tối ưu, nó sẽ không catch exception liên quan tới DAO thay vì Repository

## 3.10 Assert trong Spring

Cái này dùng để kiểm tra nhanh dữ liệu đầu vào, ví dụ như notNull, hasText…

**public** **class** AssertUtils **extends** Assert {

**public** **static** **void** notNull(**final** Object o, **final** String message, **final** Object... args) {

**if** (o == **null**) {

**throw** **new** IllegalArgumentException(String.*format*(message, args));

}

}

}

## 3.11 Singleton Pattern in Java

**package** com.test;

**public** **class** Database {

// private constructor của Database để không thể khởi tạo nó ở ngoài

**private** Database() {

}

// tạo ra 1 instance duy nhất, phải để static để method này thành method của lớp.

**private** **static** Database *INSTANCE*;

**public** **static** Database getInstance(){

**if** (*INSTANCE* == **null**) {

*INSTANCE* = **new** Database();

}

**return** *INSTANCE*;

}

**private** **boolean** offSet = **false**;

**public** **static** **boolean** isOffset(){

**return** *INSTANCE*.offSet;

}

}

Chạy hàm main Test:

**package** com.test;

**public** **class** Test {

**public** **static** **void** main(String[] args) {

System.***out***.println(Database.*getInstance*().*isOffset*());

}

}

Kết quả:



## 3.12 Sử dụng InitializingBean và DisposableBean

2 interface này bắt buộc user phải overide lại 2 method là:

1. For bean implemented InitializingBean, it will run afterPropertiesSet() after all bean properties have been set.
2. For bean implemented DisposableBean, it will run destroy() after Spring container is released the bean.

LegacyDataConfig.java:

**package** com.initializingbean;

**import** org.springframework.beans.factory.DisposableBean;

**import** org.springframework.beans.factory.InitializingBean;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Component;

@Component

**public** **class** LegacyDataConfig **implements** InitializingBean, DisposableBean {

String message;

**public** String getMessage() {

**return** message;

}

@Autowired

**public** **void** setMessage(String message) {

**this**.message = message;

}

@Override

**public** **void** afterPropertiesSet() **throws** Exception {

System.***out***.println("Init method after properties are set");

}

@Override

**public** **void** destroy() **throws** Exception {

System.***out***.println("Spring Container is destroy! Customer clean up");

}

}

Spring-Customer.xml:

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-4.0.xsd"*>

<bean id=*"legacyDataConfig"* class=*"com.initializingbean.LegacyDataConfig"*>

<property name=*"message"* value=*"i'm property message"* />

</bean>

</beans>

App.java:

**package** com.initializingbean;

**import** org.springframework.context.ConfigurableApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**public** **class** App {

**public** **static** **void** main( String[] args )

{

ConfigurableApplicationContext context =

**new** ClassPathXmlApplicationContext("Spring-Customer.xml");

LegacyDataConfig legacy = (LegacyDataConfig)context.getBean("legacyDataConfig");

System.***out***.println(legacy.message);

context.close();

}

}

## 3.13 Khác biệt giữa ApplicationContext và BeanFactory:

<https://huongdanjava.com/beanfactory-va-applicationcontext-trong-sping.html>

Cơ bản nhất là ApplicationContext extends từ interface Beanfactory nhưng ApplicationContext sẽ khởi tạo mọi đối tượng trong khung chứa bean ngay khi gọi đến, trong khi BeanFactory chỉ tạo ra đối tượng cần khi gọi getBean().

## 3.14 Sử dụng ApplicationContextAware và BeanNameAware

Cái ApplicationContextAware này trong project Daito nó có ghi một câu:

The class is created to reference Spring ApplicatinContext in a static way. This should be used in those classes that can't be configured as Spring beans, but require accessing some Spring beans.

Thấy câu này không đúng lắm. Đại khái cái thằng ApplicationContextAware này dùng để tạo ra ApplicationContext, tạo một hay nhiều cái đều được, tạo bất cứ đâu. Nếu có một cái class nào đó k được cấu hình trong khung Bean, nhưng lại cần sử dụng mấy cái instance trong Bean thì cứ lôi thằng này ra sử dụng.

Ví dụ này nằm trong project org.arpit.javapostsforlearning

Ví dụ này mình lấy trên trang này:

<http://www.java2blog.com/2012/08/spring-applicationcontext.html>

# 4. Maven Project

<http://o7planning.org/vi/10127/huong-dan-lap-trinh-spring-cho-nguoi-moi-bat-dau#a720908>

Dùng cái hướng dẫn này mình sử dụng được file pom để down các thư viện về

# 5. Các lưu ý cho Project Daito:

\_ Column nếu là String thì k được NULL, nếu NULL thì thêm ký tự single space vào.

\_ Column nếu là numberic thì k được NULL, nếu NULL thì thêm số 0 vào.

## EBCDIC:

(Extended Binary Coded Decimal Interchange Code). Nó là mã có 8 cột, không có cột kiểm tra (như bảng 1.4).

|  |  |  |
| --- | --- | --- |
| **Table of EBCDIC character set Document Produced by Jim Plaxco** | | |
| decimal value | hex value | Character |
| 000 | 00 | NUL |
| 001 | 01 | SOH |
| 002 | 02 | STX |
| 003 | 03 | ETX |
| 004 | 04 | PF |
| 005 | 05 | HT |
| 006 | 06 | LC |
| 007 | 07 | DEL |
| 008 | 08 | GE |
| 009 | 09 | RLF |
| 010 | 0A | SMM |
| 011 | 0B | VT |
| 012 | 0C | FF |
| 013 | 0D | CR |
| 014 | 0E | SO |
| 015 | 0F | SI |
| 016 | 10 | DLE |
| 017 | 11 | DC1 |
| 018 | 12 | DC2 |
| 019 | 13 | TM |
| 020 | 14 | RES |
| 021 | 15 | NL |
| 022 | 16 | BS |
| 023 | 17 | IL |
| 024 | 18 | CAN |
| 025 | 19 | EM |
| 026 | 1A | CC |
| 027 | 1B | CU1 |
| 028 | 1C | IFS |
| 029 | 1D | IGS |
| 030 | 1E | IRS |
| 031 | 1F | IUS |
| 032 | 20 | DS |
| 033 | 21 | SOS |
| 034 | 22 | FS |
| 035 | 23 |  |
| 036 | 24 | BYP |
| 037 | 25 | LF |
| 038 | 26 | ETB |
| 039 | 27 | ESC |
| 040 | 28 |  |
| 041 | 29 |  |
| 042 | 2A | SM |
| 043 | 2B | CU2 |
| 044 | 2C |  |
| 045 | 2D | ENQ |
| 046 | 2E | ACK |
| 047 | 2F | BEL |
| 048 | 30 |  |
| 049 | 31 |  |
| 050 | 32 | SYN |
| 051 | 33 |  |
| 052 | 34 | PN |
| 053 | 35 | RS |
| 054 | 36 | UC |
| 055 | 37 | EOT |
| 056 | 38 |  |
| 057 | 39 |  |
| 058 | 3A |  |
| 059 | 3B | CUB |
| 060 | 3C | DC4 |
| 061 | 3D | NAK |
| 062 | 3E |  |
| 063 | 3F | SUB |
| 064 | 40 | BLANK |
| 065 | 41 |  |
| 066 | 42 |  |
| 067 | 43 |  |
| 068 | 44 |  |
| 069 | 45 |  |
| 070 | 46 |  |
| 071 | 47 |  |
| 072 | 48 |  |
| 073 | 49 |  |
| 074 | 4A | ¢ |
| 075 | 4B | . |
| 076 | 4C | < |
| 077 | 4D | ( |
| 078 | 4E | + |
| 079 | 4F | | |
| 080 | 50 |  |
| 081 | 51 |  |
| 082 | 52 |  |
| 083 | 53 |  |
| 084 | 54 |  |
| 085 | 55 |  |
| 086 | 56 |  |
| 087 | 57 |  |
| 088 | 58 |  |
| 089 | 59 |  |
| 090 | 5A | ! |
| 091 | 5B | $ |
| 092 | 5C | \* |
| 093 | 5D | ) |
| 094 | 5E | ; |
| 095 | 5F | ¬ |
| 096 | 60 | - |
| 097 | 61 | / |
| 098 | 62 |  |
| 099 | 63 |  |
| 100 | 64 |  |
| 101 | 65 |  |
| 102 | 66 |  |
| 103 | 67 |  |
| 104 | 68 |  |
| 105 | 69 |  |
| 106 | 6A | ¦ |
| 107 | 6B | , |
| 108 | 6C | % |
| 109 | 6D | \_ |
| 110 | 6E | > |
| 111 | 6F | ? |
| 112 | 70 |  |
| 113 | 71 |  |
| 114 | 72 |  |
| 115 | 73 |  |
| 116 | 74 |  |
| 117 | 75 |  |
| 118 | 76 |  |
| 119 | 77 |  |
| 120 | 78 |  |
| 121 | 79 | ` |
| 122 | 7A | : |
| 123 | 7B | # |
| 124 | 7C | @ |
| 125 | 7D | ' |
| 126 | 7E | = |
| 127 | 7F | " |
| 128 | 80 |  |
| 129 | 81 | a |
| 130 | 82 | b |
| 131 | 83 | c |
| 132 | 84 | d |
| 133 | 85 | e |
| 134 | 86 | f |
| 135 | 87 | g |
| 136 | 88 | h |
| 137 | 89 | i |
| 138 | 8A |  |
| 139 | 8B |  |
| 140 | 8C |  |
| 141 | 8D |  |
| 142 | 8E |  |
| 143 | 8F |  |
| 144 | 90 |  |
| 145 | 91 | j |
| 146 | 92 | k |
| 147 | 93 | l |
| 148 | 94 | m |
| 149 | 95 | n |
| 150 | 96 | o |
| 151 | 97 | p |
| 152 | 98 | q |
| 153 | 99 | r |
| 154 | 9A |  |
| 155 | 9B |  |
| 156 | 9C |  |
| 157 | 9D |  |
| 158 | 9E |  |
| 159 | 9F |  |
| 160 | A0 |  |
| 161 | A1 | ~ |
| 162 | A2 | s |
| 163 | A3 | t |
| 164 | A4 | u |
| 165 | A5 | v |
| 166 | A6 | w |
| 167 | A7 | x |
| 168 | A8 | y |
| 169 | A9 | z |
| 170 | AA |  |
| 171 | AB |  |
| 172 | AC |  |
| 173 | AD |  |
| 174 | AE |  |
| 175 | AF |  |
| 176 | B0 |  |
| 177 | B1 |  |
| 178 | B2 |  |
| 179 | B3 |  |
| 180 | B4 |  |
| 181 | B5 |  |
| 182 | B6 |  |
| 183 | B7 |  |
| 184 | B8 |  |
| 185 | B9 |  |
| 186 | BA |  |
| 187 | BB |  |
| 188 | BC |  |
| 189 | BD |  |
| 190 | BE |  |
| 191 | BF |  |
| 192 | C0 | { |
| 193 | C1 | A |
| 194 | C2 | B |
| 195 | C3 | C |
| 196 | C4 | D |
| 197 | C5 | E |
| 198 | C6 | F |
| 199 | C7 | G |
| 200 | C8 | H |
| 201 | C9 | I |
| 202 | CA |  |
| 203 | CB |  |
| 204 | CC | non-displayable |
| 205 | CD |  |
| 206 | CE | non-displayable |
| 207 | CF |  |
| 208 | D0 | } |
| 209 | D1 | J |
| 210 | D2 | K |
| 211 | D3 | L |
| 212 | D4 | M |
| 213 | D5 | N |
| 214 | D6 | O |
| 215 | D7 | P |
| 216 | D8 | Q |
| 217 | D9 | R |
| 218 | DA |  |
| 219 | DB |  |
| 220 | DC |  |
| 221 | DD |  |
| 222 | DE |  |
| 223 | DF |  |
| 224 | E0 | \ |
| 225 | E1 |  |
| 226 | E2 | S |
| 227 | E3 | T |
| 228 | E4 | U |
| 229 | E5 | V |
| 230 | E6 | W |
| 231 | E7 | X |
| 232 | E8 | Y |
| 233 | E9 | Z |
| 234 | EA |  |
| 235 | EB |  |
| 236 | EC | non-displayable |
| 237 | ED |  |
| 238 | EE |  |
| 239 | EF |  |
| 240 | F0 | 0 |
| 241 | F1 | 1 |
| 242 | F2 | 2 |
| 243 | F3 | 3 |
| 244 | F4 | 4 |
| 245 | F5 | 5 |
| 246 | F6 | 6 |
| 247 | F7 | 7 |
| 248 | F8 | 8 |
| 249 | F9 | 9 |
| 250 | FA | non-displayable |
| 251 | FB |  |
| 252 | FC |  |
| 253 | FD |  |
| 254 | FE |  |
| 255 | FF | EO |

## HSQL:

**HSQLDB là** một database đơn giản, viết bằng Java. **Là** loại database quan hệ giống MySQL, Oracle, SQL Server.

Web hướng dẫn HSQLDB:

<http://o7planning.org/vi/10287/huong-dan-cai-dat-va-cau-hinh-database-hsqldb>

<http://o7planning.org/vi/10203/cau-hinh-hsqldb-datasource-su-dung-data-source-explorer>

### Lệnh tạo table:

**CREATE** MEMORY **TABLE** PUBLIC.DEPARTMENT(DEPT\_ID **INTEGER** **NOT** **NULL** **PRIMARY** **KEY**,DEPT\_NAME **VARCHAR**(255) **NOT** **NULL**,DEPT\_NO **VARCHAR**(20) **NOT** **NULL**,LOCATION **VARCHAR**(255),**UNIQUE**(DEPT\_NO))

### Lệnh insert:

**INSERT** **INTO** DEPARTMENT **VALUES**(10,'ACCOUNTING','D10','NEW YORK')

### Table Employee với Constraint:

**CREATE** MEMORY **TABLE** PUBLIC.EMPLOYEE(EMP\_ID BIGINT **NOT** **NULL** **PRIMARY** **KEY**,EMP\_NAME **VARCHAR**(50) **NOT** **NULL**,EMP\_NO **VARCHAR**(20) **NOT** **NULL**,HIRE\_DATE **DATE** **NOT** **NULL**,IMAGE BLOB(1G),JOB **VARCHAR**(30) **NOT** **NULL**,SALARY **DOUBLE** **NOT** **NULL**,DEPT\_ID **INTEGER** **NOT** **NULL**,MNG\_ID BIGINT,**UNIQUE**(EMP\_NO),**CONSTRAINT** FK75C8D6AE269A3C9 **FOREIGN** **KEY**(DEPT\_ID) **REFERENCES** PUBLIC.DEPARTMENT(DEPT\_ID),**CONSTRAINT** FK75C8D6AE6106A42 **FOREIGN** **KEY**(EMP\_ID) **REFERENCES** PUBLIC.EMPLOYEE(EMP\_ID),**CONSTRAINT** FK75C8D6AE13C12F64 **FOREIGN** **KEY**(MNG\_ID) **REFERENCES** PUBLIC.EMPLOYEE(EMP\_ID))

Không hiểu sao không thêm data được cho table này:

### Sequence:

**CREATE** SEQUENCE AJE1FP\_SEQ START **WITH** 1;

Tạo ra 1 cái rule cho mấy loại:

SMALLINT, INTEGER, BIGINT, DECIMAL and NUMERIC

sequence generator definition

<sequence generator definition> ::= CREATE SEQUENCE [ IF NOT EXISTS ] <sequence generator name> [ <sequence generator options> ]

<sequence generator options> ::= <sequence generator option> ...

<sequence generator option> ::= <sequence generator data type option> | <common sequence generator options>

<common sequence generator options> ::= <common sequence generator option> ...

<common sequence generator option> ::= <sequence generator start with option> | <basic sequence generator option>

<basic sequence generator option> ::= <sequence generator increment by option> | <sequence generator maxvalue option> | <sequence generator minvalue option> | <sequence generator cycle option>

<sequence generator data type option> ::= AS <data type>

<sequence generator start with option> ::= START WITH <sequence generator start value>

<sequence generator start value> ::= <signed numeric literal>

<sequence generator increment by option> ::= INCREMENT BY <sequence generator increment>

<sequence generator increment> ::= <signed numeric literal>

<sequence generator maxvalue option> ::= MAXVALUE <sequence generator max value> | NO MAXVALUE

<sequence generator max value> ::= <signed numeric literal>

<sequence generator minvalue option> ::= MINVALUE <sequence generator min value> | NO MINVALUE

<sequence generator min value> ::= <signed numeric literal>

<sequence generator cycle option> ::= CYCLE | NO CYCLE

## Khác nhau giữa Decimal và BigDecimal



Decimal là int, double, long bình thường. Còn BigDecimal là 1 object, tính toán số lớn và tiền tệ. Tuy nhiên BigDecimal chậm.

# 6 Add code cho etes

Làm nhanh quá k kịp nhìn nhưng tạm thời là vầy:

Chọn regular time

Project nhập mã code vào

Show project

Show Billing, chọn số nào đó (lúc anh Hoàng đưa thì chọn 000)

# 7. Tìm hiểu Project Daito:

Cảm thấy đắm đuối với số lượng project và file trong cái project này.

Trắc nghiệm nhanh nhé:

**Số lượng file**: quá nhiều

**Cảm thấy**: Hào hứng tìm hiểu

**Có lo và sợ**: tất nhiên là có

**Công việc tiếp theo**: cố gắng tìm hiểu

Giờ không tìm hiểu thì biết làm gì có ích cho đời giờ.

# 7.1 Project qre2-core:

Project nằm ở địa chỉ svn: 20.194.10.15/svn/java-csc-qre-core/trunk

## com.csc.qre.core.datatype

### DataStructureSupport.java:

**public** **interface** DataStructureSupport {…}

The interface indicating if a legacy data type support data structure.

In general, most of legacy data types may support data structure and will be the implementation of this interface.

Đây là interface bự nhất, quy định bản thiết kế cho mọi data type. Tất cả các data type đều phải implement từ thằng này. Có tổng cộng 8 methods:

**1. void** setInStrucutre(**boolean** isInStrucuture, **boolean** retainValue); Set flag to indicate the implementation is in structure or not. Không hiểu muốn làm gì :D. Set cái cờ chỉ có structure là sao?

**2. void** setParentNull(); Set cho thằng cha là null, phá vỡ mẹ cấu trúc rồi @@

**3.** WrappedData getStructureData(); Get reference to the parent, which is the root parent instead of actual parent defined in legacy. Chạy đến root parent, để làm gì vậy?

**4. int** lengthInBytes(); Chắc là trả về giá trị byte của data type.

**5. boolean** isInStructure();

**6. boolean** hasParent();

**7. void** setHasChildren(); Set cho bắt buộc phải có ít nhất 1 children

**8. boolean** hasChildren(); Indicate if the data has children. This method only make sense when {@link #isInStructure()

### MathSupport.java

**public** **interface** MathSupport {…}

Đây là gốc của Math, cung cấp các phương thức cộng trừ nhân chia cho various type: integer, long, double, BigDecimal, LegacyNumber

Có tổng cộng 49 method:

* 1. **void** add(**int** value);
  2. **void** add(**long** value);
  3. **void** add(**double** value);
  4. **void** add(BigDecimal value);
  5. **void** add(LegacyNumber value);
  6. **void** add(PredefinedConstant value);
  7. **void** subtract(**int** value);
  8. **void** subtract(**long** value);
  9. **void** subtract(**double** value);
  10. **void** subtract(BigDecimal value);
  11. **void** subtract(LegacyNumber value);
  12. **void** multiply(**int** value);
  13. **void** multiply(**long** value);
  14. **void** multiply(**double** value);
  15. **void** multiply(BigDecimal value);
  16. **void** multiply(LegacyNumber value);
  17. **int** divide(**int** value);
  18. **int** divide(**long** value);
  19. **int** divide(**double** value);
  20. **int** divide(BigDecimal value);
  21. **int** divide(LegacyNumber value);
  22. **void** add(**double** value, **boolean** rounded);
  23. **void** add(BigDecimal value, **boolean** rounded);
  24. **void** add(LegacyNumber value, **boolean** rounded);
  25. **void** addRounded(**double** value);
  26. **void** addRounded(BigDecimal value);
  27. **void** addRounded(LegacyNumber value);
  28. **void** subtract(**double** value, **boolean** rounded);
  29. **void** subtractRounded(**double** value);
  30. **void** subtract(BigDecimal value, **boolean** rounded);
  31. **void** subtractRounded(BigDecimal value);
  32. **void** subtract(LegacyNumber value, **boolean** rounded);
  33. **void** subtractRounded(LegacyNumber value);
  34. **void** multiply(**double** value, **boolean** rounded);
  35. **void** multiply(BigDecimal value, **boolean** rounded);
  36. **void** multiply(LegacyNumber value, **boolean** rounded);
  37. **void** multiplyRounded(**double** value);
  38. **void** multiplyRounded(BigDecimal value);
  39. **void** multiplyRounded(LegacyNumber value);
  40. **int** divide(**int** value, **boolean** rounded);
  41. **int** divide(**long** value, **boolean** rounded);
  42. **int** divide(**double** value, **boolean** rounded);
  43. **int** divide(BigDecimal value, **boolean** rounded);
  44. **int** divide(LegacyNumber value, **boolean** rounded);
  45. **int** divideRounded(**int** value);
  46. **int** divideRounded(**long** value);
  47. **int** divideRounded(**double** value);
  48. **int** divideRounded(BigDecimal value);
  49. **int** divideRounded(LegacyNumber value);

### Binary.java:

**public** **interface** Binary **extends** LegacyNumber, NumberDeclarationSupport<Binary>

Đây là interface extends từ LegacyNumber nên nó sẽ kế thừa các method của LegacyNumber.

Đây là interface, nó quy định về số digit và số byte cho 4 loại:

Kiểu long 8 bytes nên -2^63 đến 2^63-1, nên có tối đa 20 digit:

**int** ***LONG\_DIGITS*** = 20;

**int** ***LONG\_BYTES*** = 8;

tương tự int là 4 bytes, 10 digits

short 2 bytes, 5 digits

tiny 1 bytes, 3 digits

Đặc biệt có thêm cái array này, k biết để làm gì:

**int**[] ***MAX\_DIGITS*** = { ***SHORT\_DIGITS***, ***INT\_DIGITS***, ***LONG\_DIGITS*** };

### LegacyNumber.java:

Đây là interface, nó quy định các method mà 1 number trong framework QRE đang xây dựng phải có.

Nó quy định thêm số digit của double là 15.

**int** ***LARGE\_NUMBER\_DIGITS*** = 15;

Có tổng cộng 23 cái method trong cái LegacyNumber này.

**1. int** getInteger(); Getting no of digits – trả về số digit

**2. int** getScale(); Get the number of decimal places – trả về số decimal place. Nghĩa là số phía sau dấu phẩy. Ví dụ 123.4567 thì number of decimal places là 4.

**3. int** toInt(); Return int value, which may be truncated if the value is greater than an integer can hold. Nghĩa là trả về giá trị int nhỏ hơn hoặc bằng số đang xét (không biết đúng không.

**4. long** toLong(); Tương tự toInt

**5. double** toDouble();

**6.** BigDecimal toBigDecimal();

Lưu ý, BigDecimal sử dụng trong tính toán số lớn, tính toán tiền tệ. Và khai báo thí sử dụng BigDecimal(String) chứ không nên dùng BigDecimal(double).

**7. boolean** isUnsigned(); Check if a field is unsigned. Trả về true nếu đây là số không đếm được.

**8. boolean** isBigNumber(); Indicate if a number has precision longer than {@link LegacyNumber#LARGE\_NUMBER\_DIGITS}. Trả về true nếu số digit lớn hơn 15?

**9.** String toPlainString(); Nulls are treated as zero. A 5 digit number containing 123 would come back as "00123". A 5 digit, 1 decimal place number containing -123 would come back as "-0123.0". Chưa hiểu lắm cái này.

**10.** String toUnsignedPlainDigitString(); Nulls are treated as zero. A 5 digit number containing 123 would come back as "00123". A 5 digit, 1 decimal place number containing -123 would come back as "01230". Chưa hiểu lắm cái này.

**11. boolean** isNegative(); Check if the number is holding a negative value. Trả về true nếu là số dương.

**12. boolean** isPossitive();

**13. boolean** isZero();

**14. void** negate();If the value is not unsigned, then set its value as -value.

/\*\*

\*

\*/

**15. void** abs();Set the value as absolute value, which is always non-negative. Tạo trị tuyệt đối.

@Override

**16.** Number value(); Return the value being held, which should be a subclass of Number. Chưa hiểu cái này muốn làm gì.

Lưu ý là Number là class cha của các class Integer, Long, Double, Short này nọ.

**17. void** setRounded(**final** BigDecimal value); làm tròn với BigDecimal

**18. void** setRounded(**final** **double** value);

**19. void** setRounded(**final** **int** value);

**20. void** setRounded(**final** **long** value);

**21. void** setRounded(**final** LegacyNumber value);

**22. void** fill(**char** digit); To fill a number with a given digit. Chưa hiểu hàm này muốn làm gì.

**23. void** fill(PredefinedConstant digit); To fill a number with a given digit. Chưa hiểu hàm này muốn làm gì.

### NumberDeclarationSupport.java:

Khai báo của nó như thế này:

**public** **interface** NumberDeclarationSupport<T **extends** LegacyNumber> **extends** DeclarationSupport<T> {…}

T là 1 generic, và T phải extends từ LegacyNumber.

Define các behaviours các data type có liên quan về numeric.

Gồm 6 method:

1. T init(**int** value);

2. T init(**long** value);

3. T init(**double** value);

4. T init(PredefinedConstant value);

5. T setUnsigned(**final** **boolean** unsigned);

6. T setUnsigned();

### LegacyBoolean:



**public** **class** LegacyBooleanImpl **extends** AbstractString **implements** LegacyBoolean {…}

**private** **static** **final** **int** ***FIX\_LENGTH*** = NumberConstants.***I1***; Quy định độ lớn: 1 bit.

**private** **static** **final** String ***TRUE\_STR*** = Character.*toString*(***TRUE***);

**private** **static** **final** String ***FALSE\_STR*** = Character.*toString*(***FALSE***);

Không chắc lắm nhưng hình như quy định TRUE\_STR là 1, FALSE\_STR là 0.

…

### LegacyInteger:



**public** **interface** LegacyInteger **extends** LegacyNumber, NumberDeclarationSupport<LegacyInteger> {…}

LegacyInteger tương tự như Binary, cũng quy định về digit, nhưng ở đây không quy định về byte.

**int** ***LONG\_DIGITS*** = Binary.***LONG\_DIGITS***; 20 digits

**int** ***INT\_DIGITS*** = Binary.***INT\_DIGITS***; 10 digits

**int** ***SHORT\_DIGITS*** = Binary.***SHORT\_DIGITS***; 5 digits

**int** ***TINY\_DIGITS*** = 3; 3 digits, viết code thật vl, đáng lẽ phải là Binary.***TINY\_DIGITS*** chứ, tự nhiên cái viết số 3 ra luôn là sao?

**int**[] ***MAX\_DIGITS*** = { ***TINY\_DIGITS***, ***SHORT\_DIGITS***, ***INT\_DIGITS***, ***LONG\_DIGITS*** };

**int** ***MAX\_DIGITS\_SUPPORTED*** = ***LONG\_DIGITS***;

### PackedDecimal



### LegacyData

**public** **interface** LegacyData **extends** Variable, DataStructureSupport, CompareSupport, Serializable {…}

Implements DataStructureSupport (except lengthInBytes) as it depends on the specific data type.

LegacyData extends DataStructureSupport nên nó sẽ có tính chất của interface DataStructureSupport, tùy thuộc vào từng loại data type cụ thể.

Implements compareTo and set methods:

* compareTo LegacyData, LegacyString, LegacyNumber, Number
* compareTo LegacyDate, LegacyTime, LegacyTimestamp
* compareTo LegacyConstant, LegacyPointer
* set LegacyData, Number, Object
* set LegacyDate, LegacyTime, LegacyTimestamp
* set LegacyConstant

Support Pointer Operations

* set to a LegacyPointer
* set to address of a LegacyData

Có tổng cộng 28 method:

1. **void** set(**final** Object value);
2. **void** set(**final** LegacyData value);
3. **void** set(LegacyNumber value);
4. **void** set(LegacyString value);
5. **void** set(LegacyPointer value);
6. **void** set(LegacyDate value);
7. **void** set(LegacyTime value);
8. **void** set(LegacyTimestamp value);
9. **void** set(**final** LegacyConstant value);
10. **void** set(**final** **int** value);
11. **void** set(**final** **long** value);
12. **void** set(**final** **double** value);
13. **void** set(**final** BigDecimal value);
14. **void** set(Number value);
15. **void** set(String value);
16. **void** set(**final** **boolean** value);
17. **void** set(Date value);
18. String toString();
19. **void** clear();
20. **void** setAddress(LegacyPointer pointer);
21. **void** setAddress(LegacyData aData);
22. **void** setInternal(**char**[] value);
23. **char**[] toInternal();
24. **int** length();
25. **int** lengthInBytes();
26. Object value();
27. **boolean** isTruncated();
28. LegacyConstant getSpecialValue();

### ZonedDecimalImpl

Địa chỉ a.Ngôn dùng khi giảng:

<http://www.simotime.com/datazd01.htm>

Địa chỉ hay tìm được: <http://publib.boulder.ibm.com/iseries/v5r2/ic2924/books/c0925083170.htm>

Trong java, double hay long chỉ nên dùng 15 digit để lưu. Vì thế ZonedDecimal và packedDecimal chỉ nên dùng tối đa cho số có 15 digit, nếu lơn hơn phải dùng BigDecimal (BigZonedDecimal, BigPackedDecimal).

**public** **class** ZonedDecimalImpl **extends** AbstractDecimal **implements** ZonedDecimal {…}

#### Phân tích hàm constructor của ZonedDecimalImpl:

**public** ZonedDecimalImpl(**final** **int** precision, **final** **int** scale) {

**super**(precision, scale);

}

precision: độ chính xác

scale: tỉ lệ.

Theo sự nghiên cứu trên mạng thì precision thì là number of digits, còn scale chính là decimal place:

Ví dụ 1234.567

Precision: 7

Scale: 3

Hàm constructor inherit từ hàm constructor của AbstractDecimal:

**public** AbstractDecimal(**final** **int** precision, **final** **int** scale) {

**super**(precision, scale);

**if** (precision > ***LARGE\_NUMBER\_DIGITS***) {

**throw** **new** InvalidValueException(

"Precision is too large. Please use the implementation for large numbers, which should be subclasses of "

+ getClass() + ' ' + precision);

}

}

Hàm constructor này cũng inherit từ hàm của hàm AbstractNumber, tuy nhiên kèm theo quăng ra exception là precision đã vượt LARGE\_NUMBER\_DIGITS.

Constructor của AbstractNumber:

**public** AbstractNumber(**final** **int** precision, **final** **int** scale) {

**super**(precision);

**this**.integer = precision - scale;

**this**.precisionMult = MathUtils.*powerOfTen*(precision);

**this**.integerMult = MathUtils.*powerOfTen*(integer);

**this**.scale = scale;

hasDecimal = scale > 0;

**if** (hasDecimal) {

decimalMult = MathUtils.*powerOfTen*(scale);

} **else** {

decimalMult = NumberConstants.***I1***;

}

bigNumber = precision > ***LARGE\_NUMBER\_DIGITS***;

**if** (scale < 0 || integer < 0) {

**throw** **new** InvalidValueException("precision = " + precision + " & scale=" + scale + " is invalid");

}

// By default, length in bytes is the same as total length

bytes = precision;

}

integer = precision – scale

scale = scale

…

**Ví dụ:**



Tuy nhiên hàm constructor của AbstractNumber còn inherit từ hàm constructor của AbstractLegacyData:

**public** AbstractLegacyData(**final** **int** length) {

**if** (length <= 0) {

**this**.length = 0;

} **else** {

**this**.length = length;

}

}

Hàm này đơn giản là cho cái length bằng tham số (ở đây là precision).

#### Phân tích hàm setPattern của ZonedDecimalImpl:



#### Phân tích hàm isAPartOf của ZonedDecimalImpl

@Override

**public** ZonedDecimal isAPartOf(**final** DataStructureSupport parent) {

asAPartOf(parent, **false**);

**return** **this**;

}

Hàm gì khó như quỷ, khó hiểu vãi nồi.

***QUYẾT TÂM TÌM HIỂU NÀO!!!***

IsAPartOf này có hàm asAPartOf của AbstractLegacyData.java:

**protected** **final** **void** asAPartOf(**final** DataStructureSupport parent, **final** **boolean** retainValue) {

**if** (LegacyDataConfig.*isAutoOffset*()) {

**int** offset = LegacyDataUtils.*getOffset*(parent);

asAPartOf(parent, offset, retainValue);

LegacyDataUtils.*setNextOffset*(parent, offset + lengthInBytes());

} **else** {

asAPartOf(parent, 0, retainValue);

}

}

Ở đây cái khó hiểu nhất là hàm LegacyDataConfig.isAutoOffset(). LegacyDataConfig là class theo kiểu singleton, isAutoOffset là method static, tuy nhiên ở đây chỉ có declaration thôi thì làm sao mà xài?

**int** offset = LegacyDataUtils.*getOffset*(parent); Cái này kệ bà nó, biết là nó trả giá trị int gì đó về từ system.

Từ việc có được thằng offset, nó sẽ gọi về cái hàm overload asAPartOf có nhiều tham số hơn.

**protected** **final** **void** asAPartOf(**final** DataStructureSupport parent, **final** **int** offset, **final** **boolean** retainValue) {

AssertUtils.*notNull*(parent, "parent array must not be null");

// Safety checking, CAN't make a data is a part of itself

**if** (**this** == parent) {

**throw** **new** InvalidValueException("Can't make a data is a part of itself");

}

**final** Object current = retainValue ? value() : **null**;

// Indicates the parent is in a data structure as well.

**if** (!parent.isInStructure()) {

parent.setInStrucutre(**true**, **true**);

}

// Indicates the data is a part of a data structure

**if** (wrappedData == **null**) {

wrappedData = WrappedDataFactory.*createWrappedData*(parent, offset);

} **else** {

// When dynamic datastructure is not supported, then once a field is made a part of structure, then it

// can't be changed as it might break linking between this field and its children

**if** (inStructure && !LegacyDataConfig.*isDynamicStructureSupported*()) {

**throw** **new** InvalidOperationException(

"Can't change parent as the variable has been in a structure and dynamic structure is not supported");

}

wrappedData.setParentData(parent.getStructureData(), offset);

}

// - indicating the parent has at least a child

parent.setHasChildren();

setInStrucutre(**true**, **false**);

hasParent = **true**;

**final** **int** bytes = lengthInBytes();

// Validate offset and bytes

**if** (!ArrayUtils.*validateOffsetAndLength*(wrappedData.getData(), offset, bytes)) {

**throw** **new** InvalidValueException("offset[" + offset + "] & length[" + bytes

+ "] are not valid to parent which has length =" + wrappedData.getData().length);

}

**if** (retainValue) {

set(current);

}

}

AssertUtils.*notNull*(parent, "parent array must not be null");

Đầu tiên dùng assert của spring kiểm tra giá trị đầu vào không được null.

**if** (**this** == parent) {

**throw** **new** InvalidValueException("Can't make a data is a part of itself");

Sau đó kiểm tra cái thằng đang xài isAPartOf này có phải cùng cấp AbstractLegacyData không? Nếu phải thì throw ra exception.

## ZonedDecimalPatternWorker

**public** **interface** ZonedDecimalPatternWorker {…}

**char**[] applyPattern(**final** **char**[] strNumber, **final** **int** scale, **final** String pattern, **final** Sign sign);

String removePattern(**final** **char**[] noDigits, **int** scale, **final** String pattern, **boolean** isUnsigned);

**boolean** isPatternIgnorable(String pattern);

PreparedPattern preparePattern(ZonedDecimal number, String pattern);

**public** **static** **class** PreparedPattern {

**private** **final** String pattern;

**private** **final** **boolean** unsigned;

**public** PreparedPattern(**final** String pattern, **final** **boolean** unsigned) {

**super**();

**this**.pattern = pattern;

**this**.unsigned = unsigned;

}

**public** String getPattern() {

**return** pattern;

}

**public** **boolean** isUnsigned() {

**return** unsigned;

}

}

Ta thấy có 3 method applyPattern, removePattern, isPatternIgnorable

Trong interface này còn có thêm 1 cái class inner, chỉ gồm có String pattern và **boolean** unsigned

## com.csc.qre.core.ulti

### MathUtils

**public** **final** **class** MathUtils {…}

Vì class này khai báo final, nên có thể lôi nó ra sử dụng ở mọi nên, chỉ cần import cái package là được, ví dụ:

**his**.precisionMult = MathUtils.*powerOfTen*(precision);

**Phân tích hàm powerOfTen:**

**public** **static** **long** powerOfTen(**final** **int** exponent) {

**if** (exponent > 0 && exponent < ***POWER\_OF\_TEN\_POSITIVE***.length) {

**return** ***POWER\_OF\_TEN\_POSITIVE***[exponent];

} **else** {

**return** (**long**) Math.*pow*(***BASE\_TEN***, exponent);

}

}

Ở đây sử dụng array static POWER\_OF\_TEN\_POSITIVE:

**private** **static** **final** **long**[] ***POWER\_OF\_TEN\_POSITIVE*** = *initPowerOfTenPositive*(***CACH\_OF\_POWER\_TEN\_POSITIVE***);

Đã vậy POWER\_OF\_TEN\_POSITIVE lại còn sử dụng hàm initPowerOfTenPositive và

***CACH\_OF\_POWER\_TEN\_POSITIVE =*** LegacyNumber.***LARGE\_NUMBER\_DIGITS*** + 1 = 16

**Hàm initPowerOfTenPositive:**

**private** **static** **long**[] initPowerOfTenPositive(**final** **int** maxExponent) {

**long**[] positives = **new** **long**[maxExponent];

**for** (**int** i = 0; i < maxExponent; i++) {

positives[i] = (**long**) Math.*pow*(***BASE\_TEN***, i);

}

**return** positives;

}

ở đây MathUtils quy định ***BASE\_TEN = 10;***

initPowerOfTenPositive sẽ trả về array có length chính bằng maxExponent: 1, 10, 100, …, 10^(maxExponent -1).

ở đây ***CACH\_OF\_POWER\_TEN\_POSITIVE = 16,*** nên hiển nhiên ***POWER\_OF\_TEN\_POSITIVE*** là 1 array 16 phần tử đã nói ở trên.



Giờ hàm powerOfTen có kiểu dũ liệu là long, return ***POWER\_OF\_TEN\_POSITIVE***[exponent];

Nghĩa là dựa vào cái exponent sẽ lấy ra phần tử nào trong ***POWER\_OF\_TEN\_POSITIVE***