Spring Scala

@daclouds_
하늘의 외계인?

Hcube
wiv
J rabbit :-)

$H^3$
public class Deadlock {
    static class Friend {
        private final String name;
        public Friend(String name) {
            this.name = name;
        }
        public String getName() {
            return this.name;
        }
        public synchronized void bow(Friend bower) {
            System.out.format("%s: %s
            + " has bowed to me!%n",
            this.name, bower.getName());
            bower.bowBack(this);
        }
        public synchronized void bowBack(Friend bower) {
            System.out.format("%s: %s
            + " has bowed back to me!%n",
            this.name, bower.getName());
        }
    }
    public static void main(String[] args) {
        final Friend alphonse =
        new Friend("Alphonse");
        final Friend gaston =
        new Friend("Gaston");
        new Thread(new Runnable() {
            public void run() { alphonse.bow(gaston);
        }).start();
        new Thread(new Runnable() {
            public void run() { gaston.bow(alphonse);
        }).start();
    }
}
http://docs.oracle.com/javase/tutorial/essential/concurrency/deadlock.html
Starvation and livelock are much less common a problem than deadlock, but are still problems that every designer of concurrent software is likely to encounter.

**Starvation**

*Starvation* describes a situation where a thread is unable to gain regular access to shared resources and is unable to make progress. This happens when shared resources are made unavailable for long periods by "greedy" threads. For example, suppose an object provides a synchronized method that often takes a long time to return. If one thread invokes this method frequently, other threads that also need frequent synchronized access to the same object will often be blocked.

**Livelock**

A thread often acts in response to the action of another thread. If the other thread's action is also a response to the action of another thread, then *livelock* may result. As with deadlock, livelocked threads are unable to make further progress. However, the threads are not blocked — they are simply too busy responding to each other to resume work. This is comparable to two people attempting to pass each other in a corridor: Alphonse moves to his left to let Gaston pass, while Gaston moves to his right to let Alphonse pass. Seeing that they are still blocking each other, Alphone moves to his right, while Gaston moves to his left. They're still blocking each other, so...

http://docs.oracle.com/javase/tutorial/essential/concurrency/starvelive.html
Concurrency
Functional Programming (Spring Java)
About Us

Korea Scala User Group
제10회 KSUG 세미나

Scalability
Scala를 통해 Scalability 높여보기
이동욱
i015005@gmail.com
http://me2day.net/nephilim
http://nephilim.tistory.com
주의!! 이 슬라이드는 간접 광고를 포함하고 있습니다.
 Scala Study (10/05/08 ~ 10/09/04)
라 스칼라 코딩단 (Scala Korea)

https://github.com/codeport/scala
https://groups.google.com/group/scala-korea
The goal of Spring Scala is to make it easier to use the Spring framework in Scala. Currently, the two main areas of focus are:

- **Wiring up Scala classes as Spring Beans**, both in traditional **XML** as well as **Scala**
- **Provide Scala-friendly wrappers for the Spring templates**

For more information, please refer to the documentation on the [wiki](#).
The goal of Spring Scala is to make it easier to use the Spring framework in Scala. Currently, the two main areas of focus are:

- Wiring up Scala classes as Spring Beans, both in traditional XML as well as Scala
- Provide Scala-friendly wrappers for the Spring templates

For more information, please refer to the documentation on the wiki.
3단계

Java 코드를 Scala로 바꿔라

Scala 문법을 익혀서 좀 더 elegant하고 typesafe 한 코드를 작성하라

이익을 얻어라

Spring in Scala Webinar - Jan Machacek
http://vimeo.com/39187629
준비!
Spring Scala uses a Gradle-based build system. In the instructions below, `./gradlew` is invoked from the root of the source tree and serves as a cross-platform, self-contained bootstrap mechanism for the build. The only prerequisites are Git and JDK 1.7+

필요해요!! Git and JDK 1.7+

소스 내려받기

```bash
git clone git://github.com/SpringSource/spring-scala.git
```

빌드하기

```bash
./gradlew build
```

생성된 spring-* jar 파일을 로컬 메이븐 저장소에 설치

```bash
./gradlew install
```

... and discover more commands with `./gradlew tasks`. See also the Gradle build and release FAQ.
Eclipse or IntelliJ IDEA, ...

if you using Eclipse...

Plugins

* Required
  - Scala IDE for Eclipse

* Optional
  - ScalaTest for ScalaIDE
  - Gradle Integration
Available Software

Check the items that you wish to install.

Work with:

'Maven for Scala' – http://alchim31.free.fr/m2e-scala/update-site

Find more software by working with the "Available Software Sites" preferences.

type filter text

Name

- Maven Integration for Eclipse
- m2e – Maven Integration for Eclipse
- m2e – slf4j over logback logging (Optional)
- Maven Integration for Scala IDE

Select All  Deselect All
<table>
<thead>
<tr>
<th>Name</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maven Integration for Scala IDE</td>
<td></td>
</tr>
<tr>
<td>Maven Integration for WTP</td>
<td></td>
</tr>
<tr>
<td>Scala IDE for Eclipse</td>
<td></td>
</tr>
<tr>
<td>Scala Worksheet</td>
<td></td>
</tr>
<tr>
<td>SpringSource Tool Suite</td>
<td></td>
</tr>
<tr>
<td>SpringSource Tool Suite Gradle Integration</td>
<td></td>
</tr>
</tbody>
</table>
단순함, 간결함
Scala

class Person(
    val firstName: String,
    val lastName: String) {
    var middleName: String = _
}

Java

public class JavaPerson {
    private final String firstName;
    private String middleName;
    private final String lastName;

    public JavaPerson(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public String firstName() {
        return firstName;
    }

    public String lastName() {
        return lastName;
    }

    public String middleName() {
        return middleName;
    }

    public void middleName_$eq(String middleName) {
        this.middleName = middleName;
    }
}
잠깐!! val, var 가 뭐에요?
class Person{
    val firstName: String,
    val lastName: String{
        var middleName: String = _
    }
}
`val` 은 변경 불가

```scala
scala> val x = 1
 ; x: Int = 1

scala> x = 2
error: reassignment to val
 x = 2

  ^
```
variable 는 변경 가능

```
scala> var x = 1
; x: Int = 1
scala> x = 2
; x: Int = 2
```
Scala

foo(),
foo_$eq(String)

Java

String getFoo(),
void setFoo()
Getter / Setter?
@scala.reflect.BeanProperty

class Person(
  val firstName: String,
  val lastName: String)
{
  @BeanProperty
  var middleName: String = _
}

Java

class JavaPerson {
  ...

  public void middleName_$eq(String middleName) {
    this.middleName = middleName;
  }

  public String getMiddleName() {
    return middleName;
  }

  public void setMiddleName(String middleName) {
    this.middleName = middleName;
  }
}
빈 (Bean) 등록하고 읽어오기
XML 로 빈(Bean) 등록

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans ...>
  <bean id="kingori" class="...Person">
    <property name="firstName" value="세원"/>
    <property name="lastName" value="안"/>
  </bean>
</beans>
```
XML 읽어오기

```kotlin
val reader = new XmlBeanDefinitionReader(beanFactory)
reader.loadBeanDefinitions(
    new ClassPathResource("test-context.xml", getClass)
)

val bean = beanFactory.getBean("kingori", classOf[Person])
```
잠깐!! classOf[Person] ???

Scala

classOf[Person]

Java

Person.class
ScalaConfig 빈(Bean) 등록

class PersonConfiguration extends FunctionalConfiguration {

  bean() {
    new Person("성철", "박")
  }

}
ScalaConfig 빈(Bean) 등록

```scala
class PersonConfiguration extends FunctionalConfiguration {
  bean("geekinside", aliases = Seq("fupfin"),
       scope = BeanDefinition.SCOPE_PROTOTYPE)
  {
    new Person("성철", "박")
  }
}
```
<bean id="geekinside" class="....Person" />

<alias name="geekinside" alias="fupfin" />
<alias name="geekinside" alias="gyumee" />
@Bean(aliases="fupfin", "gyumee")
public Person geekinside() {
    return new Person("성철", "박");
}
Scala

```scala
bean("geekinside", aliases = Seq("fupfin","gyumee") { 
    new Person("성철", "박")
}
```
빈(Bean) 관계

class PersonConfiguration extends FunctionalConfiguration {
    val arawn = bean() {
        new Person("용권", "박")
    }

    val outsider = bean() {
        new Person("정훈", "변")
    }

    val arawn = bean() {
        val arawn = new Person("용권", "박")
        arawn.friend = outsider()
        arawn
    }
}
잠깐!! 리턴 값이 없다???

val arawn = bean() {
    val arawn = new Person("용권", "박")
    arawn.friend = outsider
    arawn
}

스칼라는 명시적인 리턴 값이 없을 경우, 마지막으로 평가된 값이 리턴 됩니다.
설정 조합 (Configuration composition)

abstract class PersonConfiguration extends FunctionalConfiguration {
    val firstName: String
    val lastName: String
    bean() {
        new Person(firstName, lastName)
    }
}

class JohnDoeConfiguration extends PersonConfiguration {
    val firstName = singleton() { "동욱" }
    val lastName = singleton() { "이" }
}
trait FirstNameConfig extends FunctionalConfiguration {
  lazy val firstName = bean() {
    "태환"
  }
}

trait LastNameConfig extends FunctionalConfiguration {
  lazy val lastName = bean() {
    "이"
  }
}
why trait ???

trait PasswordCodec {
  def encode(plainText: String): String
}

trait Sha1PasswordCodec extends PasswordCodec {
  def encode(plainText: String) = ...
}

trait PlainPasswordCodec extends PasswordCodec {
  def encode(plainText: String) = ...
}
Mix-in composition

class UserService (private val repository: JpaRepository[User, Long]) {
   this: PasswordCodec with Notifier =>

   @Transactional
def register(user: User) {
      user.password = encode(user.password)
      registered(user)
      repository.save(user)
   }
}

new UserService(repository) extends PlainPasswordCodec with EmailNotifier
new UserService(repository) extends Sha1PasswordCodec with SmsNotifier
class MixInConfiguration {
    @Autowired var repository: UserRepository = _
    profile("dev") {
        val user = bean() {
            new UserService(repository)
            with PlainPasswordCodec with EmailNotifier
        }
    }
    profile("prod") {
        val user = bean() {
            new UserService(repository)
            with Sha1PasswordCodec with SmsNotifier
        }
    }
}
설정 가져오기
<beans>
  <bean id="firstName" class="java.lang.String">
    <constructor-arg value="응준"/>
  </bean>
  <bean id="lastName" class="java.lang.String">
    <constructor-arg value="이"/>
  </bean>
</beans>
class PersonConfiguration extends FunctionalConfiguration {

    importXml("classpath:/names.xml")

    val john = bean() {
        new Person(
            getBean("firstName"), getBean("lastName"))
    }

}
@Configuration

public class NameConfiguration {
    @Bean
    public String firstName() {
        return "대원";
    }

    @Bean
    public String lastName() {
        return "정";
    }
}
class PersonConfiguration extends FunctionalConfiguration {

    importClass(classOf[NameConfiguration])

    val john = bean() {
        new Person(
            getBean("firstName"), getBean("lastName")
        )
    }
}
class DataSourceConfiguration extends FunctionalConfiguration {
    val dataSource = bean("dataSource") {
        val dataSource = ...
        dataSource
    }
    destroy {
        _.close()
    }
}
SimpleJdbcTemplate (단항)

template.queryForObject(
   "SELECT * FROM USERS WHERE ID = :id", Map("id" -> 1) ) {
   (set, i) => {
      set.getString("FIRST_NAME")
   }
}

template.queryForObject(
   "SELECT * FROM USERS WHERE ID = ?", 1 ) {
   (set, i) => {
      set.getString("FIRST_NAME")
   }
}
다항

```
template.query(
    "SELECT * FROM USERS WHERE LAST_NAME = :last_name",
    Map("last_name" -> "박")
) {
    (set, i) => {
        set.getString("FIRST_NAME")
    }
}
```

```
template.query(
    "SELECT * FROM USERS WHERE LAST_NAME = ?", "훈"
) {
    (set, i) => {
        set.getString("FIRST_NAME")
    }
}
```
결과를 Map 으로 받기

template.queryForMap(
   "SELECT * FROM USERS WHERE ID = :id"
   , Map("id" -> 1)
)

template.queryForMap(
   "SELECT * FROM USERS WHERE ID = ?"
   , 1
)
결과를 Seq 로 받기

template.queryForSeq(
   """SELECT * FROM USERS
       WHERE LAST_NAME = :last_name""
   , Map("last_name" -> "Odersky")
)

template.queryForSeq(
   """SELECT * FROM USERS
       WHERE LAST_NAME = ?""", "Odersky"
)
template.update(
    """"INSERT INTO USERS(ID, FIRST_NAME, LAST_NAME)
       VALUES (:id, :first_name, :last_name)"
    , Map(  
      "id" -> 3,
      "first_name" -> "창우",
      "last_name" -> "박"
    )
)

template.update(  
    """"INSERT INTO USERS(ID, FIRST_NAME, LAST_NAME)
       VALUES (?, ?, ?)"
    , 4, "창우", "박"
)
Slick is a modern database query and access library for Scala.
http://slick.typesafe.com/

Scala Language-Integrated Connection Kit

**Scala**
Seamless data access for your Scala application — Write Scala code to query your database.

**Type Safe**
All database entities and queries are statically checked at compile-time.

**Composable**
Compose query operations just as if you were using Scala's collections.
The *lifted embedding* is Slick's stable query API which is based on ScalaQuery. This example shows how to insert data and perform a simple query:

```scala
object Coffees extends Table[(String, Int, Double)]("COFFEES") {
  def name = column[String]("COF_NAME", O.PrimaryKey)
  def supID = column[Int]("SUP_ID")
  def price = column[Double]("PRICE")
  def * = name ~ supID ~ price
}

Coffees.insertAll(
  ("Colombian", 101, 7.99),
  ("Colombian_Decaf", 101, 8.99),
  ("French_Roast_Decaf", 49, 9.99)
)

val q = for {
  c <- Coffees if c.supID === 101
  // ^ comparing Rep[Int] to Rep[Int]!
} yield (c.name, c.price)
println(q.selectStatement)
q.foreach { case (n, p) => println(n + ": " + p) }
```
The *direct embedding* is a new experimental API for Slick that uses *macros* to allow expressions operating on standard Scala types to be used for database queries. This example uses a Scala *case class* to perform a simple query:

```scala
@table("COFFEES") case class Coffee(
  @column("COF_NAME") name: String,
  @column("SUP_ID") supID: Int,
  @column("PRICE") price: Double
)
val coffees = Queryable[Coffee]

// for inserts use lifted embedding or SQL

val l = for {
  c <- coffees if c.supID == 101
  // ^ comparing Int to Int!
} yield (c.name, c.price)
backend.result( l, session )
.foreach { case (n, p) => println(n + " : " + p) }
```
Slick also allows you to write your own SQL queries and execute them with an API which is optimized for Scala, much easier to use and more concise than JDBC. This example uses plain SQL code to insert some data and perform a query, mapping its results to a case class:

```
1. case class Coffee(name: String, supID: Int, price: Double)
2.
3. implicit val getCoffeeResult = GetResult(r => Coffee(r.<>, r.<>, r.<>))
4.
5. Database.forURL("...") withSession {
6.   Seq(
7.     Coffee("Colombian", 101, 7.99),
8.     Coffee("Colombian_Decaf", 101, 8.99),
10.   ).foreach(c => sqlu"
11.     insert into coffees values (${c.name}, ${c.supID}, ${c.price})
12.   
13.   val sup = 101
14.   val q = sql"select * from coffees where sup_id = $sup".as[Coffee]
15.   // A bind variable to prevent SQL injection ^
16.   q.foreach(println)
17. }
```
def someServiceMethod = {
    transactionTemplate.doInTransaction(status => {
        updateOperation1
        resultOfUpdateOperation2
    })
}

public Object someServiceMethod() {
    return transactionTemplate.execute(new TransactionCallback() {
        // the code in this method executes in a transactional context
        public Object doInTransaction(TransactionStatus status) {
            updateOperation1();
            return resultOfUpdateOperation2();
        }
    });
}
TransactionManagement trait

class TransactionalExample extends TransactionManagement {
    val dataSource: DataSource = ...
    val jdbcTemplate = new SimpleJdbcTemplate(dataSource)
    val transactionManager = new DataSourceTransactionManager(dataSource)

    val result: String = transactional() {
        status => {
            jdbcTemplate.queryForObject[String](
                "SELECT FIRST_NAME FROM USERS WHERE ID = 1"
            )
        }
    }
}
RestTemplate

```java
val template = new RestTemplate()
val result = template.getForAny
[String](
   "http://example.com"
)

RestTemplate restTemplate = new RestTemplate();
String result = restTemplate.getForObject(
   "http://example.com",
   String.class
);
```
ConnectionFactory connectionFactory = ...;

JmsTemplate template = new JmsTemplate(connectionFactory);
template.send("queue", new MessageCreator()
    {
        public Message createMessage(Session session) throws JMSException {
            return session.createTextMessage("Hello World");
        }
    });

Message message = template.receive("queue");

if (message != null && message instanceof TextMessage) {
    System.out.println(((TextMessage) message).getText());
} else {
    System.out.println("No text message received");
}
JmsTemplate (Scala)

```scala
val connectionFactory : ConnectionFactory = ...
val template = new JmsTemplate(connectionFactory)

template.send("queue") {
  session: Session => session.createTextMessage("Hello World")
}

template.receive("queue") match {
  case Some(textMessage: TextMessage) => println(textMessage.getText)
  case None => println("No text message received")
}
```
Function

template.send("queue", new MessageCreator() {
    public Message createMessage(Session session)
        throws JMSException {
            return session.createTextMessage("Hello World");
        }
});

template.send("queue") {
    session: Session => session.createTextMessage("Hello World")
}
Pattern Matching

Message message = template.receive("queue");
if (message != null && message instanceof TextMessage) {
    System.out.println(((TextMessage) message).getText());
} else {
    System.out.println("No text message received");
}

template.receive("queue") match {
    case Some(textMessage: TextMessage) =>
        println(textMessage.getText)
    case None => println("No text message received")
}
DEMO

https://github.com/rstoyanchev/spring-mvc-chat