0319_1546_플레이 프레임워크 **2.0** 소개

024 - 찜해주세요~
025 - 찜해주세요~
026 - 찜해주세요~ 확인필요
027 - 찜해주세요~ 표현 약간 바꾸어보기
028 - 찜해주세요~
029 - 찜해주세요~
030 - 찜해주세요~
031 - 찜해주세요~

번역본 확정작업에 참여하신 분

번역 제안관련 글을 모은 곳 - http://opentutorials.org/course/230/1474

번역본 확정방법

이번에는 번역본을 확정하거나, 향상시킵니다 :)

찜 단위에서 다음을 표시해주세요~

OK + 필명
보류, 확인필요

001 - 찜해주세요~

Introducing Play 2.0

Play 2.0 소개

Since 2007,
we have been working
on making Java web application development
easier.
우리는 2007년부터 자바 웹 어플리케이션 개발을 쉽게 만들기 위해 연구하고 있습니다.

Play started
as an internal project at Zenexity

and was heavily influenced
by our way of doing web projects: (웹 프로젝트를 수행하는 우리의 방식에..
플레이 프레임웍이 영향을 받았어요~^^)
focusing on developer productivity,
respecting web architecture,
and using a fresh approach
to packaging conventions
from the start
- breaking so-called JEE best practices
where it made sense.

플레이 프레임워크는 Zenexity 의 내부 프로젝트로 개발하기 시작했습니다. 우리가 웹 프로젝트를 수행하는 방식은 플레이 프레임워크에 많은 영향을 주었습니다. 개발자의 생산성을 중요시하고, 웹 아키텍처를 존중하며, JEE 에서 사용하는 최고의 관행을 깨뜨리는 등 패키지를 만드는 것에 대해서도 완전히 새로운 접근법을 시도합니다.

플레이 프레임워크는 Zenexity의 내부 프로젝트로 개발하기 시작했습니다.
플레이 프레임워크는 우리가 웹 프로젝트를 수행하는 방식에 영향을 많이 받았습니다.
우리의 웹프로젝트 수행방식의 특성에는 다음과 같은 것이 있습니다 :

개발자의 생산성을 중요시하고,
웹 아키텍쳐를 존중하며,
JEE에서 사용하는 최고의 관행을 깨뜨리는 등 패키징에 관해 완전히 새로운 접근법을

시도합니다. .

002 - 찜해주세요~

In 2009,
we decided to share these ideas
with the community as an open source project.

2009년에 우리는 이러한 생각들을 커뮤니티에 공유하기로 결심했고, 오픈소스 프로젝트로 진행하기 시작했어요.

The immediate feedback
was extremely positive

and the project gained
a lot of traction.

커뮤니티에서는 즉각적으로 매우 긍정적인 반응이 왔고, 프로젝트는 박차를 가하게 되었어요.

Today
- after two years of active development -
Play has several versions,
an active community of 4,000 people,
with a growing number of applications
running in production all over the globe.

플레이 프레임워크를 개발한 지 어느덧 2년이 되어가는 지금, 버전도 여러개가 있고, 4000명의 회원이 참여하는 활기찬 커뮤니티도 생겼습니다. 그리고 제작에 사용되는 어플리케이션의 수는 전세계적으로 계속 증가하고 있어요.

플레이 프레임워크를 개발한 지 어느덧 2년이 되어가는 지금, 버전도 여러 개가 있고, 4000명의 회원이 참여하는 활기찬 커뮤니티도 생겼습니다. 그리고 제작에 사용되는 어플리케이션의 수는 전 세계적으로 계속 증가하고 있어요.

003 - 찜해주세요~

Opening a project to the world

certainly means more feedback,
but it also means
discovering and learning about new use cases,
requiring features
and un-earthing(발굴하다, 찾다) bugs
that we were not specifically considered
in the original design and its assumptions.

프로젝트를 전 세계에 공개하자 매우 많은 피드백을 받았습니다.
또한 새로운 사용예에 대해서도 알게 되었습니다.
새로운 사용예에 따르면 새로운 기능이 추가로 필요하기도 하고, 몰랐던 버그도 알게 되었습니다. 이러한 기능이나 버그는 초기에 설계할 당시에는 고려하지 않았던 부분들이죠.

004 - 찜해주세요~

During the two years of work on Play
as an open source project
we have worked
to fix this kind of issues,
as well as
to integrate new features
to support a wider range of scenarios.

As the project has grown,
we have learned a lot
from our community and from our own experience
- using Play
in more and more complex and varied projects.

2년동안 플레이 프레임워크를 오픈소스 프로젝트로 작업하는 동안, 우리는 이런 종류의 이슈들을 처리해왔고, 다양한 사용 시나리오를 위해 새로운 기능들을 통합해 왔습니다.

프로젝트가 커져갈수록, 우리는 커뮤니티와 경험을 통해서 많은 것을 배워왔습니다. 플레이 프레임워크를 점점더 복잡하고 다양한 프로젝트에 사용하면서 말이죠.

프로젝트가 커지면서, 우리는 커뮤니티로부터, 그리고 그간의 경험으로부터 많은 것을 배웠습니다. 더욱 복잡하고 다양한 프로젝트에 플레이 프레임워크를 사용하면서 말이죠.

005 - 찜해주세요~

Meanwhile,
technology and the web have continued to evolve.

The web
has become the central point
of all applications.

HTML, CSS and JavaScript technologies
have evolved quickly
- making it almost impossible
for a server-side framework to keep up.

그동안에 웹 기술은 지속적으로 발달해 왔습니다. 웹은 모든 어플리케이션의 핵심이 되었어요.
HTML, CSS 와 자바스크립트 기술은 빠르게 발전했습니다.

서버사이드 프레임워크가 따라잡기에 거의 불가능할 정도였죠.

006 - 찜해주세요~

The whole web architecture
is fast moving
towards real-time processing,

and the emerging **requirements** of today's project profiles
mean
SQL no longer works
as the exclusive datastore technology.

모든 웹 아키텍처는 실시간 처리를 하는 방향으로로 빠르게 바뀌고 있습니다.
(처리 기술로.. 라고 하면, 처리 기술로 인하여.. 라고 이해할 소지도 있어요 :) 여기서는.
쪽으로~ 방향의 의미가 들어가야 좋을 듯요~)
독점적인 데이터저장 기술을 이용하는 SQL 을 더이상 사용하지 않는 것은 근래의 프로젝트
특성에 새롭게 요구되는 점이기도 합니다.
또한 요즘에는 프로젝트에 대한 요구사항이 많아져서, 데이터 저장기술로 SQL만 쓴다는 것은

옛날 얘기가 되었습니다.

웹 아키텍쳐는 실시간 처리를 하는 방향으로 빠르게 바뀌고 있습니다.
또한 요즘에는 프로젝트에 대한 요구사항이 많아져서, 데이터 저장기술로 SQL만 쓴다는 것은 옛날 얘기가 되었습니다.

At the programming language level
we've witnessed
some monumental changes
with several JVM languages,
including Scala, gaining popularity.

프로그래밍 언어부문에서도 실로 놀라운 변화가 있었습니다. Scala 를 포함한 몇몇 JVM 언어가 인기를 얻고 있죠.

That's why
we created Play 2.0,
a new web framework
for a new era.

이처럼 새로운 시대가 왔기 때문에, 이에 걸맞는 새로운 웹 프레임워크인 플레이 2.0을 만들었습니다.

007 - 찜해주세요~

Built for asynchronous programming

비동기방식의 프로그래밍을 위해 만들었어요!

Today's web applications
are integrating more concurrent real-time data,

so web frameworks
need to support
a full asynchronous HTTP programming model.

오늘날 웹 어플리케이션에서는 병렬 실시간 데이터를 더욱더 많이 다룹니다.
그래서 웹 프레임워크에서는 비동기 HTTP 프로그래밍 모델을 전폭 지원해주어야 합니다.

Play was initially designed
to handle classic web applications
with many short-lived requests.

'플레이'는 원래 고전적인 웹 어플리케이션을 다루도록 설계되었어요. 고전적인 웹 어플리케이션에서는 HTTP 요청이 대부분 잠깐 동안만 유지되었었죠.

But now,
the event model is the way
to go for persistent connections
- though Comet, long-polling and WebSockets. (<- 이 부분 어케 처리할지;)

하지만 요즘에 이벤트 모델은 지속적인 연결을 사용하는 방식을 씁니다. Comet과 롱 폴링, 웹소켓 등에서요.

008 - 찜해주세요~

Play 2.0 is architected
from the start
under the assumption
that every request is potentially long-lived.

플레이 2.0은 모든 요청이 오랫동안 지속된다는 가정 하에 처음부터 다시 개발되었습니다.

But that's not all:
we also need a powerful way
to schedule
and run long-running tasks.

그러나 그것이 전부가 아니었습니다. 우리는 오랫동안 실행되는 작업을 스케쥴하고 실행하기 위한 강력한 방법이 필요했습니다.

009 - 찜해주세요~

The Actor-based model

is unquestionably the best model today
to handle highly concurrent systems,

and the **best implementation** of that model
available for both Java and Scala
is Akka
- so it's going in. (going in?)

액터에 기반한 모델은 고도의 병렬 시스템을 다루는데 있어서 오늘날 의심할 여지없는 최고의 모델입니다.

Play 2.0 provides
native Akka support (native support:?)
for Play applications,
making it possible
to write highly-distributed(??) systems.

플레이 2.0은 플레이로 만든 어플리케이션을 위해서 네이티브하게 Akka를 지원합니다. 이러한 Akka의 지원은 고도의 분산화된 시스템을 만들수 있게 해줍니다.

010 - 찜해주세요~

Focused on type safety (type safety??)
플레이 프레임워크는 타입 안전성을 중요하게 여깁니다.

(역자주 : 타입 안정성 - 타입의 오류를 막거나 방지하는 정도)

* 타입 안정성 (type safety) : 타입 오류를 막거나 방지하는 정도
http://en.wikipedia.org/wiki/Type_safety

One benefit
of using a statically-typed programming language
for writing Play applications
is that
the compiler can check parts of your code.

플레이 어플리케이션을 작성할 때 정적 프로그래밍 언어를 사용하면 한 가지 장점이 있는데요, 컴파일러가 코드의 자료형을 검사할 수 있다는 것입니다. (역자주: 정적 프로그래밍 언어:

컴파일 단계에서 자료형에 대한 검사를 수행하는 언어)

플레이 어플리케이션을 작성할 때 정적 프로그래밍 언어를 사용하면 한 가지 장점이 있는데요, 컴파일러가 코드의 자료형을 검사할 수 있다는 것입니다. (역자주: 정적 프로그래밍 언어: 컴파일 단계에서 자료형에 대한 검사를 수행하는 언어)

This is not only useful
for detecting mistakes
early in the development process,
but it also makes it a lot easier
to work on large projects
with a lot developers involved.

정적 프로그래밍 언어를 사용하면, 개발 진행 초기에 실수를 발견할 수 있어서 좋습니다. 뿐만 아니라 개발자들이 많이 참여하는 대형 프로젝트를 수행하기가 훨씬 수월합니다.

011 - 찜해주세요~ 확인필요

Adding Scala **to the mix** for Play 2.0,
we clearly benefit
from even **stronger compiler guarantees**
- but that's not enough.

플레이 2.0에서 Scala를 추가적으로 지원하면서 컴파일러가 훨씬 더 강한 보장을 함으로써 분명한 이득을 있지만 그것으로는 충분치 않습니다.

플레이 2.0에서 Scala를 추가적으로 지원하면서 컴파일러가 자료형을 더 확실하게 검사해준다는 장점이 분명 있습니다. 하지만 이것만으로는 충분하지 않죠.

In Play 1.x,
the template system was dynamic,
based on the Groovy language(이런 언어가 있는건지? - 있는거군요 ㅎㅎ),
and the compiler couldn't do much for you.

플레이 1.x에서 템플릿 시스템은 동적이였습니다. 그리고 그루비 언어를 기반이었기 때문에 컴파일러는 많은 것을 할 수 없습니다.

플레이 1.x에서 템플릿 시스템은 동적이었습니다. 그리고 그루비 언어 기반이었기 때문에 컴파일러가 해줄 수 있는 게 별로 없었어요. (역자주 : Groovy Language : http://groovy.codehaus.org)

As a result,
errors in templates were
could only be detected
at run-time.

결과적으로 템플릿에서의 오류는 런타임시에만 발견될 뿐입니다.

The same goes for verification
of glue code with controllers.

마찬가지로 glue code with controllers의 오류도 런타임시에야 발견됩니다. (역자주: * Glue Code : http://en.wikipedia.org/wiki/Glue_code)

012 - 찜해주세요~

In version 2.0,
we really wanted to push this idea
of having Play check
most of your code
at compilation(?) time
further.

우리는 버전 2.0에서는 플레이가 대부분의 코드를 컴파일 타임에 검사할 수 있도록 하는 아이디어를 꼭 적용하고 싶었습니다.

This is why
we decided to use the Scala-based template engine
as the default
for Play applications
- even for developers
using Java
as the main programming language.

그래서 플레이 어플리케이션을 위해서 Scala에 기반한 템플릿 엔진을 기본으로 사용하기로 했습니다. 심지어 자바를 주 언어로 사용하는 개발자들에게까지 말이죠.

013 - 찜해주세요~

This doesn't mean
that you have to become a Scala expert
to write templates in Play 2.0,
just as you were not really required to know Groovy
to write templates in Play 1.x.

이 말이 여러분이 플레이 2.0에서 템플릿을 작성하기 위해서 Scala 전문가가 되어야 한다는 것을 뜻하지는 않습니다. 단지 플레이 1.x에서 템플릿을 작성하기 위해 Groovy를 알아야 할 필요가 없던 것처럼 말이죠.

그렇다고 해서 플레이 2.0에서 템플릿을 작성하려면 Scala 전문가가 되어야 하는 것은 아닙니다. 마치, 플레이 1.x에서 템플릿을 작성하기 위해 Groovy를 잘 알 필요는 없었던 것처럼 말이죠.

014 - 찜해주세요~ 확인필요

In templates,
Scala is mainly used
to navigate your object graph
in order to display relevant information,
with a syntax that is very close to Java's.

템플릿에서는, 여러분의 오브젝트 그래프(object graph)를 탐색하는데에 주로 스칼라가 쓰입니다. 자바의 문법과 매우 비슷하게 관련된 정보를 보여주죠.

However,
if you want to unleash the power of Scala
to write advanced templates abstractions,
you will quickly discover
how Scala,
being expression-oriented and functional,
is a perfect fit
for a **template engine(**템플릿 엔진**...??).**

하지만, advanced templates abstraction을 쓰는 데에, 스칼라의 강력함을 제대로 사용하려면, 스칼라가 템플릿 엔진에 얼마나 완벽한지 얼른 이해하시는 게 좋을 거에요. Scala is expression-oriented and functional.

>> expression-oriented.. 는 어떤 의민지 몰겠어요 ㅜ functional도..

=> 표현을 제안해주세요~

015 - 찜해주세요~ 확인필요

And that's not only true
for the template engine:
the routing system(이건 모하는 시스템인지?) is also fully type-checked(어케 표현해야...?).

템플릿 엔진의 경우에만 그런게 아닙니다.
routing system도 자료형을 모두 체크해주는 방식으로 돌아갑니다.

Play 2.0 checks your routes' descriptions,
and verifies that
everything is consistent,
including the reverse routing part(??).

플레이 2.0에서는 경로 기술한 것을 확인해서, 모든 것이 일관적인지, reverse routing part까지도 일관적인지를 검증합니다. (확신이 안서요!)

A nice side effect
of being fully compiled
is that the templates and route files
will be easier
to package and reuse.

이처럼 플레이 2.0에서 완전하게 컴파일 해주기 때문에 생겨나는 부수적인 이점이 있는데요, 템플릿과 경로 파일(route files)을 더욱 쉽게 패키지하고 재사용할 수 있다는 점입니다.

You also get a significant performance gain
on these parts

at run-time.

이런 부분 때문에 실행(run-time)할 때의 성능도 훨씬 더 좋아집니다.

016 - 찜해주세요~

Native support for Java and Scala

Early in the Play project's history,
we started exploring the possibility
of using the Scala programming language
for writing Play applications.

We initially introduced this work
(this work가 가리키는 건, 스칼라 언어를 지원하는 작업.. 인 것 같아요..)
as an **external module,**
to be able to experiment freely
without impacting the framework itself.

자바와 스칼라를 네이티브 방식으로 지원**(native support)**합니다**.**

플레이 프레임워크 프로젝트 초기에 이미, 플레이 어플리케이션을 작성하는 언어로 스칼라를 사용하는 것에 대해서 고려하였습니다.

처음에는 스칼라를 외부 모듈로 사용했습니다. 이렇게 하면 프레임워크 전체에 영향을 주지 않으면서도 자유롭게 실험해볼 수 있었죠.

017 - 찜해주세요~

Properly **integrating** Scala
into a Java-based framework
is not trivial.

스칼라를 자바기반 프레임워크에 제대로 통합하는 것은 쉬운 일이 아닙니다.

Considering Scala's compatibility with Java,
one can quickly achieve

a first naive integration
that simply uses Scala's syntax
instead of Java's.

스칼라가 자바와 호환성이 좋으니까, 자바 문법대신 스칼라의 문법을 사용해서 시험삼아 시도를 해볼 순 있겠죠.

This, however,
is certainly not the optimal way
of using the language.
하지만, 이것은 언어를 사용하는 최적의 방법은 분명 아닙니다.

Scala is a mix of true object orientation
with functional programming.

스칼라는 진정한 객체기반언어와 함수적 언어의그래밍의 특징을 모두 가지고 있습니다.

Leveraging the full power of Scala
requires rethinking most of the framework's APIs.

스칼라의 힘을 극대화하려면 프레임워크 API를 거의 모두 다시 생각해봐야합니다.

018 - 찜해주세요~

We quickly reached the limits
of what we can do with Scala support
as a separate module.

얼마 지나지 않아서 별도의 모듈로 제공되는 Scala 지원을 갖고 할 수 있는 것에는 한계가 있다는 것을 알게 되었습니다.

Initial design choices
we made in Play 1.x,
relying heavily on Java reflection API and byte code manipulation,

have made it harder to progress
without completely rethinking some essential parts
of Play's internals.

Play 1.x 버전 때에는 자바 리플렉션 API와 바이트코드 조작(byte code manipulation)에 심하게 의존했습니다. 이때 설계한 방식 때문에, Play 내부의 핵심적인 부분들을 뒤집어어 엎어야만 진행할 수 있었습니다.

(역자주: Java Reflection API는 임의의 자바 클래스를 읽어 들이고 조사하고 실행시킬 수 있는 방법을 제공하는 API. 어떤 클래스를 이용할 지 개발 시점에는 알 필요가 없으므로, 임의의 코드를 실행해야 하는 프레임워크나 개발 및 테스트 도구를 개발할 때 유용하나, 성능이 떨어지고 보안상 문제를 초래할 수 있음

(역자주 : 바이트코드조작은 바이트코드로 컴파일된 자바 클래스를 실행시간에 직접 조작하는 것)

019 - 찜해주세요~

Meanwhile,
we have created several awesome components
for the Scala module,
such as the new type-safe template engine
and the brand new SQL access component Anorm.

한편, 우리는 스칼라 모듈 용으로 멋진 컴포넌트를 몇 개 만들었습니다. 이를테면, 자료형을 자동으로 검사해주는 템플릿 엔진(type-safe template engine)을 새로 만들었고요, SQL 엑세스 콤포넌트인 Anorm을 완전히 새로이 만들었어요.

This is why
we decided that,
to fully unleash the power of Scala with Play,

we would move Scala support
from a separate module
to the core of Play 2.0,
which is designed
from the beginning
to **natively support** Scala
as a programming language.

그래서 우리는 이렇게 판단했습니다. 플레이 프레임워크 내에서 스칼라의 강력함을 제대로 활용하기 위해서는, 별도의 모듈로 스칼라를 지원하는게 아니라, 플레이 2.0의 핵심으로서 지원해야 한다고요. 그래서 플레이 2.0을 설계할 때, 프로그래밍 언어로서 스칼라를 네이티브

방식으로 지원하도록 설계하였습니다. (역자주 : 네이티브 방식 지원 : 인터프리터나 컨버터를 사용하지 않고 스칼라 언어를 그대로 사용할 수 있도록 지원하는 것)

020 - 찜해주세요~

Java, on the other hand,
is certainly not getting any less support
from Play 2.0;
quite the contrary.

한편, 플레이 2.0에서 자바를 덜 지원한다는 의미는 절대 아닙니다. 오히려 그 반대죠.

The Play 2.0 build
provides us
with an opportunity
to enhance the development experience
for Java developers.

자바 개발자가 플레이 2.0의 빌드를 사용하면, 개발을 할 때 새로운 경험을 하실 수 있습니다.

Java developers
get a real Java API
written with all the Java specificity in mind.

자바의 장점들을 모두 고려한 진정한 자바 API를 사용할 수 있습니다.

021 - 찜해주세요~

Powerful build system
강력한 빌드 시스템

From the beginning of the Play project,
we have chosen a fresh way
to run,
compile

and deploy Play applications.

플레이 프로젝트를 시작하면서, 플레이 애플리케이션 실행 및 컴파일 그리고 배포를 하기 위한 새로운 방식을 선택하였습니다.

It may have looked
like an esoteric design at first,
but it was crucial
to providing
an asynchronous HTTP API
instead of the standard Servlet API,
short feedback cycles
through live compilation
and reloading of source code
during development,
and promoting a fresh packaging approach.

이러한 설계방식은 처음에는 난해하게 보일 수도 있습니다.
하지만, 표준 서블릿 API 가 아닌 비동기 HTTP API 를 제공하는데 있어서 꼭 필요했습니다.
또한 실시간 컴파일을 통해 피드백 주기를 짧게 줄이고, 개발하는 동안 소스 코드를 새로 불러오는 것을 불러오고, 새로운 패키징 방식을 추진하기 위해서도 필요한 선택이었습니다.

이러한 방식은 처음에는 난해하게 보일 수도 있습니다.
하지만 표준 서블릿 API가 아닌 비동기 HTTP API를 제공하는데 있어서 이러한 방식이 꼭 필요했습니다. 또한 실시간 컴파일을 통해 피드백 주기를 짧게 하기 위해서도 필요했습니다.
그리고 개발하는 동안 소스 코드를 새로 불러오는 것을 제공하기 위해서도 필요했습니다.
마지막으로 새로운 패키징 방식을 추진하기 위해서도 필요한 선택이었습니다.

Consequently,
it was difficult to make Play
follow the standard JEE conventions.

그 결과, 표준 JEE 규약을 따르도록 플레이를 만들기가 어려웠습니다.

022 - 찜해주세요~

Today,

this idea of container-less deployment
is increasingly accepted
in the Java world.

오늘날에는 컨테이너 없이 배포한다는 아이디어가 자바 세상에서 점점 더 받아 들여지고 있습니다.

It's a design choice
that has allowed the Play framework
to run natively on platforms
like Heroku,
which introduced a model
that we consider
the future of Java application deployment
on elastic PaaS platforms.

그것은 Heroku와 같은 플랫폼에서 네이티브하게 플레이 프레임워크를 실행하는 것을 허용하는 설계적 결정이었다. Heroku는 신축적인 PaaS 플렛폼에 자바 어플리케이션 배포를 고려한 모델을 소개했다.

이렇게 설계한 덕분에 플레이 프레임워크는 Heroku 같은 플랫폼에서 네이티브한 방식으로 돌아갈 수 있었습니다. Heroku는 신축적인 PaaS 플램폿에 자바 어플리케이션을 배포하는 것의 미래를 감안한 모델을 도입하였고요.

023 - 찜해주세요~

Existing Java build systems,
however,
were not flexible enough
to support this **new approach.**

그러나 기존에 존재하는 자바 빌드 시스템들은 이러한 새로운 접근방법을 지원하기에는 충분히 유연하지 못했습니다.

그러나 기존의 자바 빌드 시스템은 유연하지 않아서, 이런 새로운 접근 방법을 지원하지 못했습니다.

Since we wanted to
provide

straightforward tools to run and deploy Play applications,
in Play 1.x
we created a collection of Python scripts
to handle build and deployment tasks.

우리는 플레이 어플리케이션을 실행하고 배포하기 위한 쉬운 도구를 제공하기 원했기 때문에, 플레이 1.x 에서는 빌드와 배포 작업을 하는 파이썬 스크립트의 모음을 제작했습니다.

플레이 어플리케이션을 쉽게 실행하고 배포할 수 있도록, 플레이 1.x에서는 파이썬 스크립트 모음(collection)을 제작해서 빌드와 배포작업을 하도록 했습니다.

024 - 찜해주세요~

Meanwhile,
developers
using Play for more enterprise-scale projects,
which require
build process customization
and integration
with their existing company build systems,
were a bit lost.

한편, 기업의 프로젝트처럼 큰 규모의 프로젝트를 진행할 때 플레이를 사용하는 개발자들은 좀 난처해졌죠. 왜냐하면 기업용 프로젝트에서는 빌드 프로세스를 조정할 수 있어야 하고(build process customization), 회사에서 기존에 쓰는 빌드 시스템하고 통합도 할 수 있어야 하거든요.

The Python scripts
we provide with Play 1.x
are in no way a fully-featured build system
and are not easily customizable.

플레이 1.x에서 제공하는 파이썬 스크립트는 모든 기능이 다 갖춰진 빌드 시스템(fully-featured build system)하고는 거리가 멉니다. 그리고 조정(customize)하기에도 쉽지 않고요.

That's why
we've decided to go for
a more powerful build system for Play 2.0.

그래서 플레이 2.0에서는 좀 더 강력한 빌드 시스템을 제공하기로 했습니다.

025 - 찜해주세요~

Since we need a modern build tool,
flexible enough to support Play original conventions
and able to build Java and Scala projects,
we have chosen to integrate sbt in Play 2.0.

플레이의 원래 규칙을 지원하기에 충분히 유연하고, 자바와 스칼라 프로젝트를 빌드할 수 있는 현대적인 빌드 도구가 필요했기 때문에, 우리는 플레이 2.0에 sbt를 통합시키기로 결정하였습니다.

우리에게는 현대적인 빌드 도구가 필요했습니다. 현대적인 빌드 도구는 플레이의 원래방식을 지원해줄 수 있도록 유연해야 하고, 자바와 스칼라 프로젝트를 빌드할 수도 있어야 했습니다. 그래서 우리는 플레이 2.0에 sbt를 통합시키기로 결정하였습니다.

This, however,
should not scare existing Play users
who are happy with the simplicity
of the original Play build.

그러나 이것은 원래 플레이 빌드의 간편함에 만족하고 있는 기존의 플레이 유저를 겁나게 하지는 않습니다.

하지만 플레이 빌드가 간편하다는 이유로 만족하시는 기존의 플레이 유저 분들은, 이 결정 때문에 겁내시지 않아도 됩니다.

026 - 찜해주세요~ 확인필요

We are leveraging
the same simple `play new`, `run`, `start` experience
on top of an extensible model:
Play 2.0 **comes with** a preconfigured build script
that will just work for most users.

우리는 이전과 같은 단순한 플레이 생성, 실행, 시작 경험을 확장된 모델의 위에 영향을 받도록

하였습니다다. 플레이 2.0에는 미리 설정된 빌드 스크립트가 포함되어 있는데요, 대부분의 사용자들에게는 아주 매끄럽게 작동합니다.

On the other hand,
if you need to change
the way your application is built and deployed,

the fact that
a Play project is a standard sbt project
gives you all the power you need
to customize and adapt it.

반면에 어플리케이션을 빌드하고 배포하는 방법을 바꿔야 하는 경우에는, 플레이 프로젝트가 sbt 프로젝트라는 사실이 유용할 것입니다. 빌드하고 배포하는 방법을 커스터마이징할 수 있기 때문입니다.

027 - 찜해주세요~ 표현 약간 바꾸어보기

This also means
better integration with Maven projects
out of the box,

the ability
to package and publish your project
as a simple set of JAR files
to any repository,

and especially live
compiling and reloading
at development time
of any depended project,

even for
standard Java or Scala library projects.

이러한 사실은 메이븐 프로젝트와 특별히 더 나은 통합을 할 수 있다는 것을 뜻합니다. 어떠한 저장소에도 간단히 JAR 파일들을 패키징하고 배포할 수 있는 능력과 배포 시점에 의존적인 프로젝트를 컴파일하고 다시 로딩할 수 있는 능력에 있어서 말입니다.

>> 표현 약간 바꾸어보기~

028 - 찜해주세요~

Datastore and model integration

'Data store'
is no longer synonymous
with 'SQL database',
and probably never was.
A lot of interesting data storage models
are becoming popular,
providing different properties
for different scenarios.

데이터저장과 모델 통합

'데이터 저장'이라고 하면 예전에는 'SQL 데이터베이스'를 떠올리는 게 당연했는데요, 최근에는 새롭고 흥미로운 데이터 저장 모델이 인기를 얻고 있습니다. 이들은 여러 가지 가능한 상황에 적절한 다양한 속성을 각기 제공합니다.

029 - 찜해주세요~

For this reason
it has become difficult
for a web framework like Play
to make bold assumptions
regarding the kind of data store
that developers will use.

그래서 플레이 같은 웹프레임워크로서는 개발자들이 어떤 종류의 데이터 저장방식(데이터베이스)을 사용하는지 가정하기가 어려워 졌습니다.

A generic model concept in Play
no longer makes sense,
since it is almost impossible
to abstract over all these kinds of technologies
with a single API.

플레이 프레임워크의 일반적인 모델 개념(generic model concept)은 더 이상 유효하지 않습니다. 왜냐하면 단 하나의 API만으로 이 모든 종류의 (data store) 기술을 모두 다루기란

불가능하기 때문입니다.

030 - 찜해주세요~

In Play 2.0,
we wanted to make it really easy
to use any data store driver, ORM, or any other database access library
without any special integration
with the web framework.

플레이 2.0에서 우리는 어떠한 데이터 저장 드라이버, ORM, 웹 프레임워크와 특별히 통합되지 않은 어떠한 데이터베이스 접근 라이브러리라도 쉽게 사용할 수 있길 원했습니다.

플레이 2.0에서는 데이터 저장 드라이버나, ORM, 또는 다른 데이터 베이스 접근 라이브러리를 아주 사용하기 쉽게 만들고 싶었습니다. 플레이 2.0하고 특별히 통합시키지 않고서도 쉽게 사용할 수 있게 만들고 싶었습니다.

We simply want to offer
a minimal set of helpers (helper가 의미하는 것은?)
to handle common technical issues,
like managing the connection bounds.

우리는 연결 한계(?)(connection bounds)를 관리하는 것과 같은 공통적인 기술 이슈를 다루기 위해 간단히 최소한의 헬퍼만을 제공하길 원했습니다.

connection bounds를 관리하는 것 같은 흔한 기술적 이슈를 다루는 최소한의 헬퍼를 제공하고 싶습니다. (역자주 : 헬퍼 - 도와주는 역할을 하는 클래스, 모듈, 코드 등. 예를 들어 사용하기 어려운 클래스의 경우, 한꺼풀 더 씌워서 사용하기 편하게 만든 클래스)

031 - 찜해주세요~

We also want, however,
to maintain the full-stack aspect
of Play framework
by bundling default tools
to access classical databases
for users WHO don't have specialized needs,

and that's why Play 2.0
comes with built-in relational database access libraries
such as Ebean, JPA and Anorm.

또한 우리는 특별한 요구가 없는 사용자들이 고전적인 데이터베이스에 접근하기 위한 기본적인 도구들을 플레이 프레임워크에 풀 스택 관점 (full-stack aspect)에서 유지하기를 원했습니다.
그것이 플레이 2.0이 Ebean, JPA, Anorm 같은 관계형 데이터베이스 접근 라이브러리를 포함해 놓은 이유입니다.

또한, 플레이 2.0을 풀 스택 방식으로 유지하기 위해 고전적 데이터베이스에 접근할 수 있는 기본적인 도구들을 제공하기로 했습니다. 그래서 플레이 2.0에 Ebean, JPA, Anorm 같은 관계형 데이터베이스를 엑세스할 수 있는 라이브러리가 포함되어 있습니다. 데이터베이스에 관해 특별한 요구사항이 있는 경우가 아니라면 기본 라이브러리를 사용하면 됩니다.

(역자주 : full-stack: 프레임워크 내에 기본적으로 필요한 기반 라이브러리가 다 포함된 형태로 배포가 되면 full-stack이라고 부릅니다.)