ruby on rails

ruby on rails

un framework ágil de desarrollo web abierto y multiplataforma
orientado a aumentar la productividad del programador

características

metaprogramación: programas que escriben programas (no generación de código)

registro activo: acceso a datos basado en el patrón de Fowler

convención sobre configuración

andamiaje (scaffolding)

pruebas integradas

entornos de desarrollo, pruebas y producción

ruby

Hola.java

```
public class Hola {
  public static void main(String[] args) {
    System.out.println("Hola, mundo");
  }
}
```

hola.rb

```
puts "Hola, Mundo"
```

elegante, potente, legible, conciso

duck typing, expresivo, auto-comentado

ruby

Bucle.java

```
public class Bucle {
  public static void main(String[] args) {
    String frutas[] = new String[]{"manzana", "naranja"};
    for(int i = 0; i < frutas.length; i++) {
      System.out.println(frutas[i]);
    }
  }
}
```

bucle.rb

```
frutas = ["manzana", "naranja"]
frutas.each do |fruta|
  puts fruta
end
```

¿la panacea?

NO

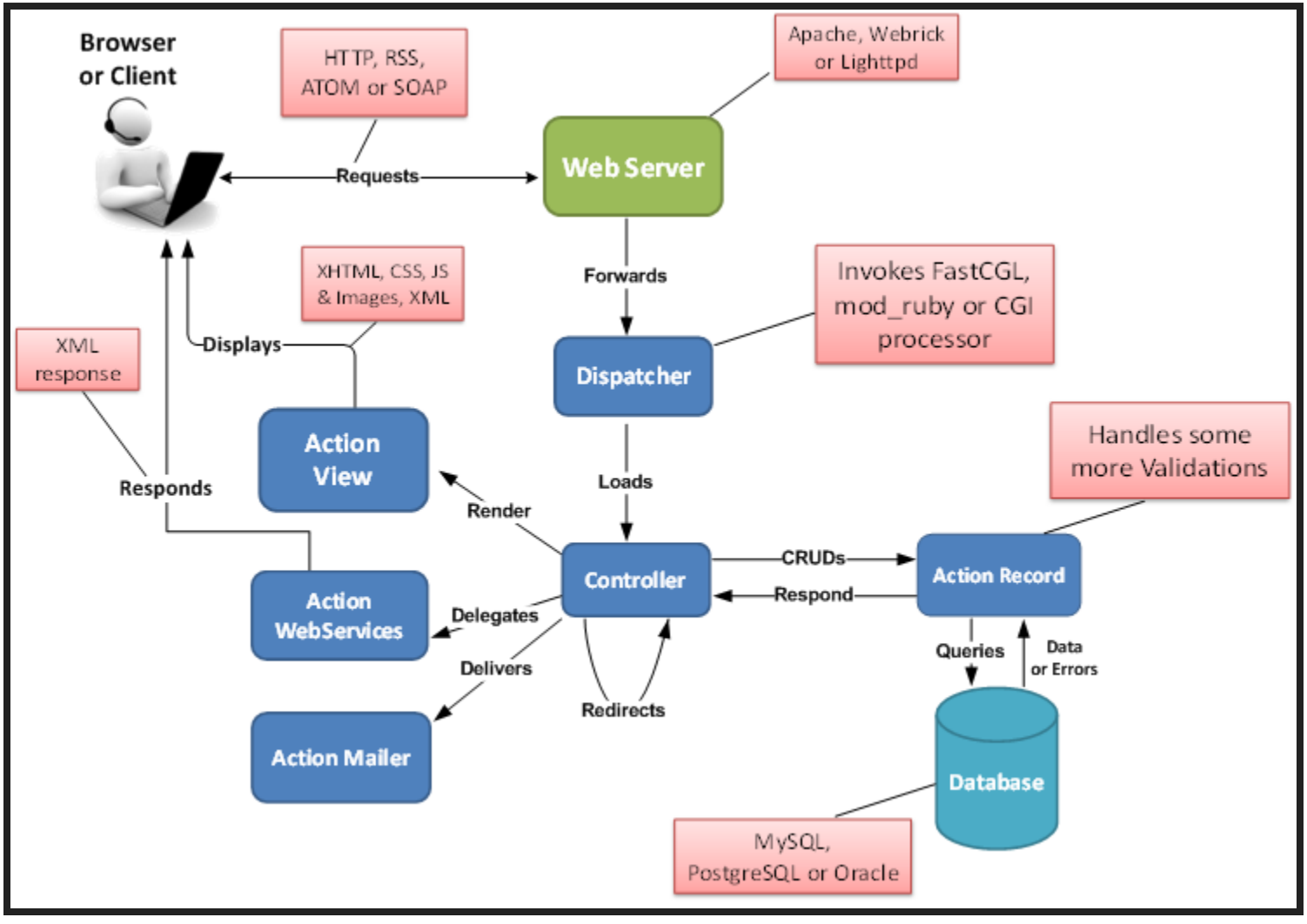
es **lento**: ~10 veces más que java

es menos maduro: menos desarrolladores, menos soporte...

es otra tecnología más...

rails

- convención sobre configuración
- DRY
- mapeo objeto - relacional
- patrón MVC
- reutilización de código
- prácticas ágiles (testing)
- seguridad
- enfocado en REST


Browser
or Client
HTTP, RSS,
ATOM or SOAP
Apache, Webrick
or Lighttpd
Requests
Web Server
Forwards
XHTML, CSS, JS
& Images, XML
Invokes FastCGI,
mod_ruby or CGI
processor
XML
response
Displays
Dispatcher
Action
View
Handles some
more Validations
Responds
Loads
Render
Controller
CRUDs
Respond
Action Record
Action
WebServices
Delegates
Queries
Data
or Errors
Delivers
Redirects
Action Mailer
Database
MySQL,
PostgreSQL or Oracle

grails

qué es grails

---

*"un **framework web** abierto para la plataforma **java** cuyo propósito es aumentar la **productividad** del desarrollador"*

características

desarrollo ágil

enfoque en la **funcionalidad** y no en el bajo nivel

arquitectura basada en **plugins**

convención sobre configuración (sin XML)

reutilización de código *legacy* en java

scaffolding

entorno rápido de usar (desarrollo, pruebas, producción)

groovy!

funcionalidad disponible a través de **mixins**: métodos añadidos dinámicamente (funcionalidad inyectada)

calidad

soporte para *test-first programming*: geb y spock

rápida ejecución de tests y aplicación: contenedor empotrado

despliegues de código en caliente

menos código a revisar y plugins: codenarc

patrones de diseño para groovy

seguridad por defecto

SQL escapado para prevenir inyección SQL

las plantillas de **scaffolding** proporcionadas por defecto escapan todos los campos de datos al ser mostrados

las **etiquetas** de creación de links de GSP utilizan mecanismos de escape adecuados para prevenir inyección de código

prevención contra **XSS**

uso de **codecs** para escapar datos devueltos como HTML, Javascript o URLs de cara a prevenir ataques de inyección

spring security: plugin que integra el famoso framework de autenticación y autorización de usuarios

tecnologías

servicios web

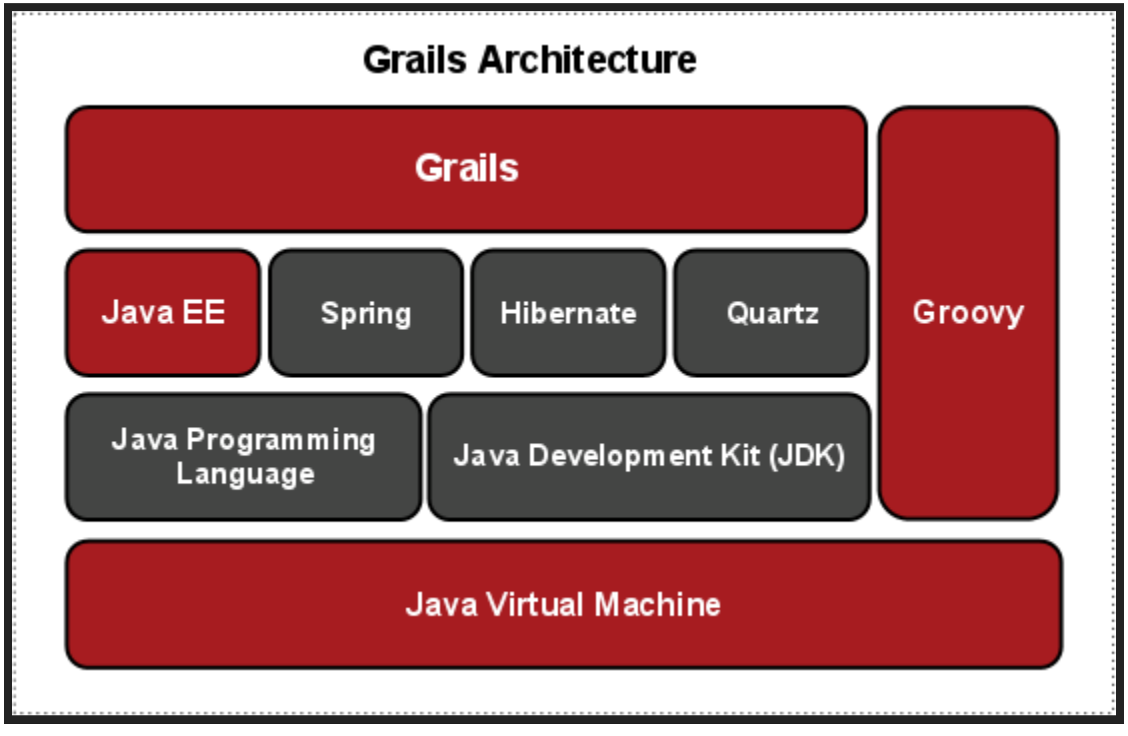
- REST por defecto
- SOAP a través de plugins

internacionalización

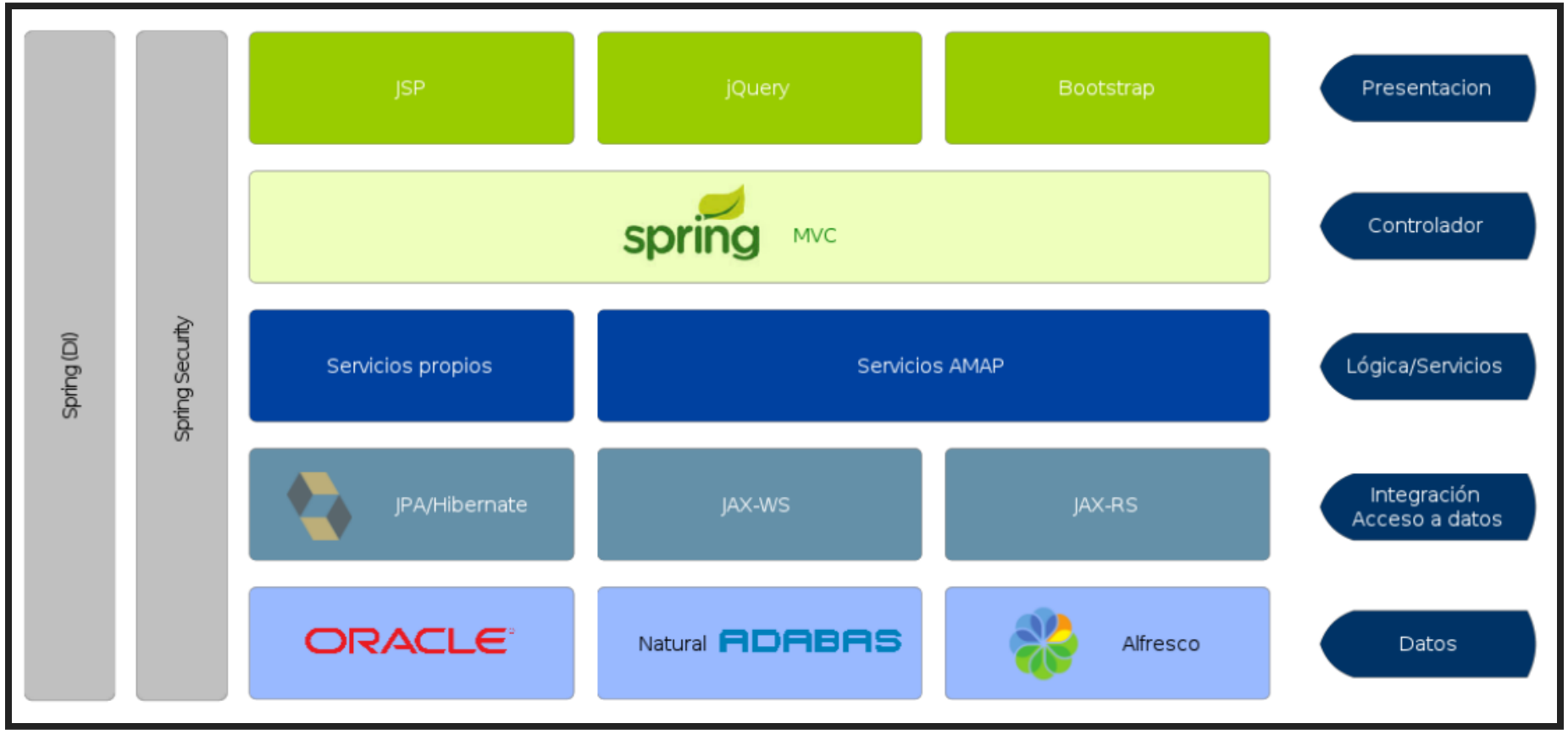
inyección de dependencias

programación asíncrona

arquitectura



¿os suena?


Spring (DI)
Spring Security
JSP
jQuery
Bootstrap
Presentacion
spring
MVC
Controlador
Servicios propios
Servicios AMAP
Lógica/Servicios
JPA/Hibernate
JAX-WS
JAX-RS
Integración
Acceso a datos
ORACLE
Natural ADABAS
Alfresco
Datos

spring/hibernate tradicional

configurar el arquetipo lleva tiempo

la cosa se complica rápidamente

hay demasiadas capas de abstracción: DAOs, DTOs...

hay demasiados archivos de configuración

hoy día, ya no tantos...

groovy

un lenguaje dinámico rico en características para la plataforma java

- completamente orientado a objetos
- tipado estático o dinámico
- duck typing
- metaprogramación
- closures, beans, gstrings...
- excelente lenguaje de scripting

java con esteroides

pero... más lento

groovy

imports por defecto (`java.lang.*`, `java.util.*`, `java.net.*`)

elementos opcionales: punto y coma, tipos de variables, declaración `return`

métodos y clases públicos por defecto, miembros privados

closures: funciones anónimas que pueden aceptar parámetros, devolver valores, ser asignadas a variables, ser pasadas como argumentos...

métodos adicionales para el trabajo con listas

operador null safe: `?.`

groovy vs java

java

```
for (String it : new String[] {"Pepe", "Paco", "Ana"}) {
  if (it.length() <= 4) {
    System.out.println(it);
  }
}
```

groovy

```
["Pepe", "Paco", "Ana"].findAll { it.size() <= 4 }.each { println it }
```

domain driven design

un modelo de dominio no es sólo un diagrama,
sino una abstracción rigurosamente organizada y
selectiva del conocimiento que reside en un
experto de negocio

- el modelo y el corazón del diseño se dan forma el uno al otro
- el modelo es la **columna vertebral** del lenguaje utilizado por todos los miembros del equipo
- el modelo es **conocimiento** destilado

modelado efectivo

1. unir el modelo con la implementación
2. cultivar un lenguaje basado en el modelo
3. desarrollar un modelo rico en conocimiento
4. destilar el modelo
5. experimentar

todo dentro de un marco de aprendizaje continuo: **cuando hablamos de software, nunca sabemos lo suficiente**

ddd en grails

grails se basa en `fat models` y `skinny controllers`

el desarrollador no tiene por qué conocer detalles de implementación de la base de datos...

simplemente modela el dominio

y a partir del dominio, a través de scaffolding, se puede generar una aplicación CRUD base

convención sobre configuración

- `UserService.groovy` genera automáticamente un bean llamado `userService`
- los campos de los objetos se "autoenganchan" por nombre
- los nombres de clases se mapean a nombres de tabla
- los nombres de campos se mapean a nombres de columna
- los métodos dinámicos de búsqueda se añaden en tiempo de ejecución
- los controladores y sus acciones se mapean automáticamente a urls basadas en sus nombres: `UserController.list()` se mapea a `/user/list`

gorm

grails object relational mapping

hibernate con esteroides

- diseño orientado a dominio
- asociaciones: `hasOne, hasMany...`
- inyección automática de finders
- consultas avanzadas con HQL, criteria...
- basado en *pogos*

spring MVC

lógica básica de controlador

validación y data binding

configuraciones de tiempo de ejecución

transaccionalidad

scaffolding

generación automática de **interfaces crud** para un dominio

muy útil para demostraciones y aprendizaje

- dinámico o estático
- generación de controladores y vistas a partir del modelo

consola y shell

útiles para trabajar de forma interactiva con la aplicación

- depuración
- verificar comportamiento y estado de objetos
- probar el efecto de `finders`

gsp y recursos estáticos

groovy server pages: jsp con esteroides

```
<g:each in="${[1,2,3]}" var="num">
  <p>Number ${num}</p>
</g:each>
```

más flexible y sencillo

asset pipeline plugin: procesado de CSS y JS

- procesado al vuelo
- fácil depuración
- compresión, minificación, cache digests

plugins

el **núcleo de grails**: en sí mismo, se construye a partir de un conjunto de plugins

en grails, el plugin es la unidad estándar de código reutilizable

cualquiera puede escribir un plugin

spock

testing elegante y altamente expresivo

```
def "puede añadir un elemento"() {

    given:
    def list = new ArrayList<String>()
    def elemento = "Hola, Spock"

    when:
    list.add(elemento)

    then:
    list.size() == 1
    list.contains(element)
}
```

spock

tests orientados a los datos

```
def "máximo de dos números"() {

    expect:
    Math.max(a, b) == c

where:
    a | b | c
    1 | 8 | 9
    7 | 3 | 9

}
```

grails vs ror

característica	grails	ror
lenguaje	groovy	ruby
servidores	tomcat,jboss,jetty...	mongrel, fastcgi
enfoque	orientado a dominio	orientado a db
orm	hibernate	active record
tipado	estático/dinámico	dinámico
testing	spock, junit, mockito	rspec, cucumber
threads	nativo	pobre

grails vs ror

característica	grails	ror
infraestructura	alta	baja
aprendizaje	fácil	fácil + db
madurez	joven	más maduro
configuración	fácil	peor en Windows
rendimiento	escala	escala mal
estructura	más compleja	sencilla
comunidad	pequeña	grande

problemas detectados

magia negra: generación de código misteriosa en ocasiones

lejanía de la fuente: dificultad para depurar

pesado: producción de **mega-artefactos**

orientación al plugin:

- más niveles de complejidad
- más fuentes de error desconocidas

el futuro de grails

encontrar patrocinador: OCI

¿apache software foundation?

- mantener las partes positivas
- introducir perfiles de aplicaciones
- mejorar el rendimiento
- limitar la generación de código

spring boot

Acceso al repositorio del curso

Miguel Expósito Martín