(http://grails.org/index.html)

显示导航

build passing (https://travis-ci.org/grails-guides/gorm-without-grails)

获取代码

SSH

HTTPS

git@github.com:grails-g

下载ZIP

目录

使用GORM构建Spring Boot应用程序

了解如何使用GORM构建Spring Boot应用程序

作者：本莱茵，塞尔吉奥德尔阿莫

Grails版本： N / A - Spring Boot版本：1.5.6.RELEASE

1 Grails培训

改进这个文档

Grails培训 - 由创建并积极维护Grails框架的人员开发和提供！

2入门

改进这个文档

在本指南中，您将使用GORM构建Spring Boot应用程序。

2.1你需要什么

改进这个文档

要完成本指南，您需要具备以下条件：

- 有一段时间在你的手上

- 一个体面的文本编辑器或IDE
- 已JAVA_HOME正确安装JDK 1.7或更高版本

2.2如何完成指南

要开始，请执行以下操作：

- 下载 (https://github.com/grails-guides/gorm-without-grails/archive/master.zip)并解压缩源代码

要么

- 克隆Git (https://git-scm.com/)存储库：
  ```
  git clone https://github.com/grails-guides/gorm-without-grails.git
  ```

Grails指南存储库包含两个文件夹：

- `initial` 初始项目。通常是一个简单的Grails应用程序，其中包含一些额外的代码，可以让您领先一步。
- `complete`一个完整的例子。这是完成指南提供的步骤并将这些更改应用于`initial`文件夹的结果。

要完成指南，请转到该`initial`文件夹

- `cd` 成 `grails-guides/gorm-without-grails/initial`

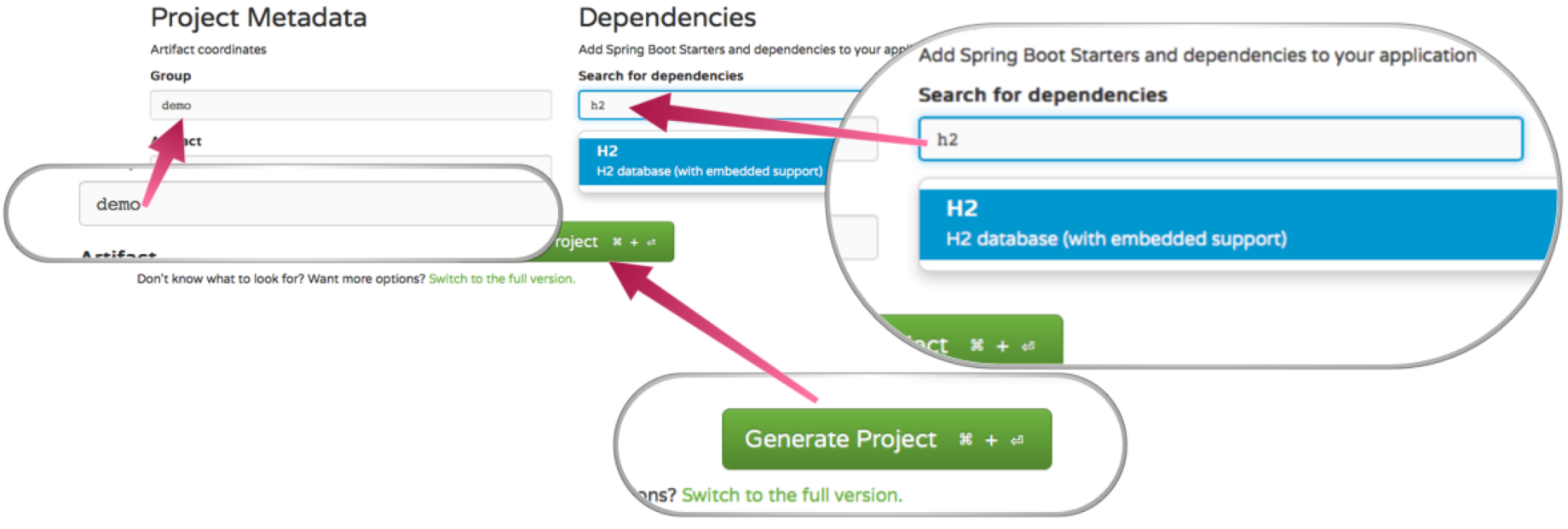
并按照下一节中的说明进行操作。

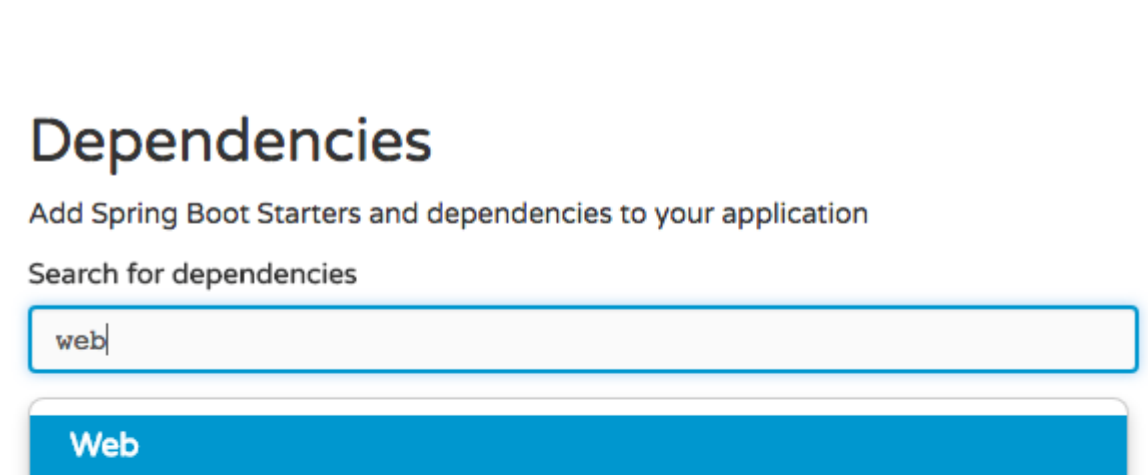
如果你进入， 你可以直接进入**完成的例子**`cdgrails-guides/gorm-without-grails/complete`

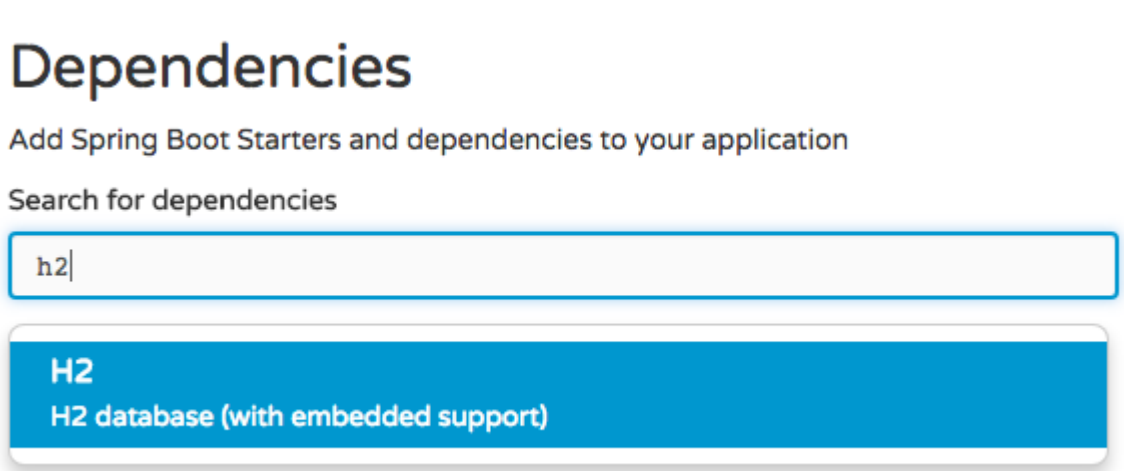
如果您想从头开始，请使用Spring Initializr (https://start.spring.io/)创建一个新的Spring Boot应用程序 。

选择`Gradle Project`使用`Groovy`与`Spring Boot 1.5.6`。选择构建和项目类型后，将组设置为您的组织即。`com.example`在我们的例子中我们将使用`demo`。

完成后，将项`Web`和`h2`依赖项添加到项目中。







点击Generate Project。

3配置Build

改进这个文档

编辑*build.gradle*文件以包含GORM依赖项。

/build.gradle

GROOVY

```
buildscript {
    ext {
        springBootVersion = '1.5.6.RELEASE'
    }
    repositories {
        mavenCentral()
    }
    dependencies {
        classpath("org.springframework.boot:spring-boot-gradle-plugin:${springBootVersion}")
    }
}

apply plugin: 'groovy'
apply plugin: 'eclipse'
apply plugin: 'org.springframework.boot'

version = '0.0.1-SNAPSHOT'
sourceCompatibility = 1.8

repositories {
    mavenCentral()
}


dependencies {
    compile('org.springframework.boot:spring-boot-starter-web')
    compile('org.codehaus.groovy:groovy')
    runtime('com.h2database:h2')
    testCompile('org.springframework.boot:spring-boot-starter-test')

     1
    compile("org.grails:gorm-hibernate5-spring-boot:6.1.6.RELEASE")
    compile "org.hibernate:hibernate-core:5.1.0.Final"
    compile "org.hibernate:hibernate-ehcache:5.1.0.Final"

     2
    runtime "org.apache.tomcat:tomcat-jdbc:8.5.0"
    runtime "org.apache.tomcat.embed:tomcat-embed-logging-log4j:8.5.0"
    runtime "org.slf4j:slf4j-api:1.7.10"

    runtime "org.glassfish.web:el-impl:2.1.2-b03"
}
```

1	将GORM所需的依赖项添加到项目中。
2	用于连接池

阅读GORM文档以了解更多信息 (http://gorm.grails.org/latest/hibernate/manual/index.html#springBoot)

3.1配置

改进这个文档

src/main/resources/application.yml使用Spring Boot时，可以使用文件配置GORM for Hibernate 。

SRC /主/资源/ application.yml

GROOVY

```
dataSource:
    pooled: true
    dbCreate: create-drop
    url: jdbc:h2:mem:devDb
    driverClassName: org.h2.Driver
    username: sa
    password:
hibernate:
    cache:
        queries: false
        use_second_level_cache: true
        use_query_cache: false
        region.factory_class: org.hibernate.cache.ehcache.EhCacheRegionFactory
```

4编写应用程序

改进这个文档

我们将创建一个类似于您在Grails应用程序中找到的包结构：

- demo.controller
- demo.domain
- demo.service
- demo.init

4.1创建域

改进这个文档

现在有了我们的软件包，让我们继续创建我们的域对象。

- Manufacturer.groovy
- Vehicle.groovy

您可以随意使用自己喜欢的IDE来创建这些或执行以下操作

BASH

```
$ cd complete/src/main/groovy/demo/domain/
$ touch Manufacturer.groovy
$ touch Vehicle.groovy
```

既然我们所有的类存根都已到位，那就继续编辑它们吧。

/src/main/groovy/demo/domain/Manufacturer.groovy

GROOVY

```
package demo.domain

import grails.gorm.annotation.Entity
import groovy.transform.ToString
import org.grails.datastore.gorm.GormEntity

@ToString
@Entity
class Manufacturer implements GormEntity<Manufacturer> {

    String name

    static hasMany = [vehicles: Vehicle]

    static constraints = {
        name blank: false
    }
}
```

/src/main/groovy/demo/domain/Vehicle.groovy

```groovy
package demo.domain

import grails.gorm.annotation.Entity
import org.grails.datastore.gorm.GormEntity
import groovy.transform.ToString

@ToString
@Entity
class Vehicle implements GormEntity<Vehicle> {
    String name
    Integer year
    static belongsTo = [manufacturer: Manufacturer]

    static constraints = {
        name nullable: false, blank: false
    }
}
```

Manufacturer并且Vehicle有一对多的关系。

我们在Grails之外使用GORM。因此，我们需要使用。注释我们的域类grails.gorm.annotation.Entity。另外我们实现了这个GormEntity特性。它只是为了在Grails之外帮助GORM支持GORM。

4.2种子数据

改进这个文档

当应用程序启动时，我们保存一些数据。

/src/main/groovy/demo/Application.groovy

```groovy
package demo

import org.springframework.boot.ApplicationArguments
import demo.init.BootStrap
import groovy.transform.CompileStatic
import org.springframework.beans.factory.annotation.Autowired
import org.springframework.boot.ApplicationRunner
import org.springframework.boot.SpringApplication
import org.springframework.boot.autoconfigure.EnableAutoConfiguration
import org.springframework.boot.autoconfigure.SpringBootApplication
import org.springframework.context.annotation.ComponentScan

@EnableAutoConfiguration
@ComponentScan
@CompileStatic
@SpringBootApplication
class Application implements ApplicationRunner {

    @Autowired
    BootStrap bootStrap

    static void main(String[] args) {
        SpringApplication.run Application, args
    }

    void run(ApplicationArguments args) throws Exception {
        bootStrap.init()
    }
}
```

/src/main/groovy/demo/init/BootStrap.groovy

GROOVY

```groovy
package demo.init

import demo.domain.Manufacturer
import demo.domain.Vehicle
import grails.gorm.transactions.Transactional
import groovy.transform.CompileStatic
import org.springframework.stereotype.Component

@Component
@CompileStatic
class BootStrap {

    @Transactional
    void init() {
        Manufacturer audi = new Manufacturer(name: 'audi')
        audi.addToVehicles(new Vehicle(name: 'A3', year: 1996))
        audi.addToVehicles(new Vehicle(name: 'A4', year: 1994))
        audi.save()

        Manufacturer ford = new Manufacturer(name: 'ford')
        ford.addToVehicles(new Vehicle(name: 'Ford KA', year: 1996))
        ford.save()
    }
}
```

4.3创建服务层

改进这个文档

接下来让我们为我们的应用程序创建服务层。

BASH

```bash
$ cd src/main/groovy/demo/service
$ touch ManufacturerService
$ touch VehicleService
```

我们将使用GORM 6.1数据服务。

> *数据服务通过添加使用GORM逻辑自动实现抽象类或接口的功能，从而实现了已实现的服务层逻辑。*

/src/main/groovy/demo/service/ManufacturerService.groovy

GROOVY

```groovy
package demo.service

import demo.domain.Manufacturer
import groovy.transform.CompileStatic
import org.springframework.stereotype.Service

@CompileStatic
@grails.gorm.services.Service(Manufacturer)
@Service
interface ManufacturerService {
    List<Manufacturer> findAll()
}
```

/src/main/groovy/demo/service/VehicleService.groovy

GROOVY

```groovy
package demo.service

import demo.domain.Manufacturer
import demo.domain.Vehicle
import grails.gorm.services.Where
import groovy.transform.CompileStatic
import org.springframework.stereotype.Service

@CompileStatic
@grails.gorm.services.Service(Vehicle)
@Service
interface VehicleService {

    @Where({ manufacturer.name == manufacturerName })
    List<Vehicle> findAllByManufacturer(String manufacturerName)

}
```

此外，我们所有的服务类都使用@Service('someName')注释。这个注释告诉Spring我们的服务被调用了什么，所以它可以正确映射我们的依赖注入。

4.4创建控制器

改进这个文档

最后，我们创建一些控制器，以便我们可以对数据进行基本的UI访问。

BASH

```bash
$ cd src/main/groovy/demo/controller
$ touch ManufacturerController
$ touch VehicleController
```

现在让我们编辑我们的控制器src/main/groovy/demo/controller。

控制器将具有以下注释

- @RestController - 表示控制器是restful，并且它可以通过url返回数据
- @Autowired - 允许使用依赖注入来访问我们的服务
- @RequestMapping("/") - 设置方法的url映射

/src/main/groovy/demo/controller/VehicleController.groovy

GROOVY

```groovy
package demo.controller

import demo.domain.Vehicle
import demo.service.VehicleService
import org.springframework.beans.factory.annotation.Autowired
import org.springframework.web.bind.annotation.PathVariable
import org.springframework.web.bind.annotation.RequestMapping
import org.springframework.web.bind.annotation.RestController

@RestController
class VehicleController {

    @Autowired VehicleService vehicleService

    @RequestMapping("/{manufacturerName}/vehicles")
    List<String> vehiclesByManufacturer(@PathVariable String manufacturerName) {
        vehicleService.findAllByManufacturer(manufacturerName)*.name
    }
}
```

/src/main/groovy/demo/controller/ManufacturerController.groovy

```groovy
package demo.controller

import demo.service.ManufacturerService
import org.springframework.beans.factory.annotation.Autowired
import org.springframework.web.bind.annotation.RequestMapping
import org.springframework.web.bind.annotation.RestController

@RestController
class ManufacturerController {

    @Autowired
    ManufacturerService manufacturerService

    @RequestMapping("/")
    List<String> index(){
        manufacturerService.findAll()*.name
    }
}
```

运行应用程序:

```
./gradlew bootRun
```

你应该能够调用端点:

```
curl "http://localhost:8080"
```

得到回应:

```
["audi","ford"]
```

或检索制造商的车辆:

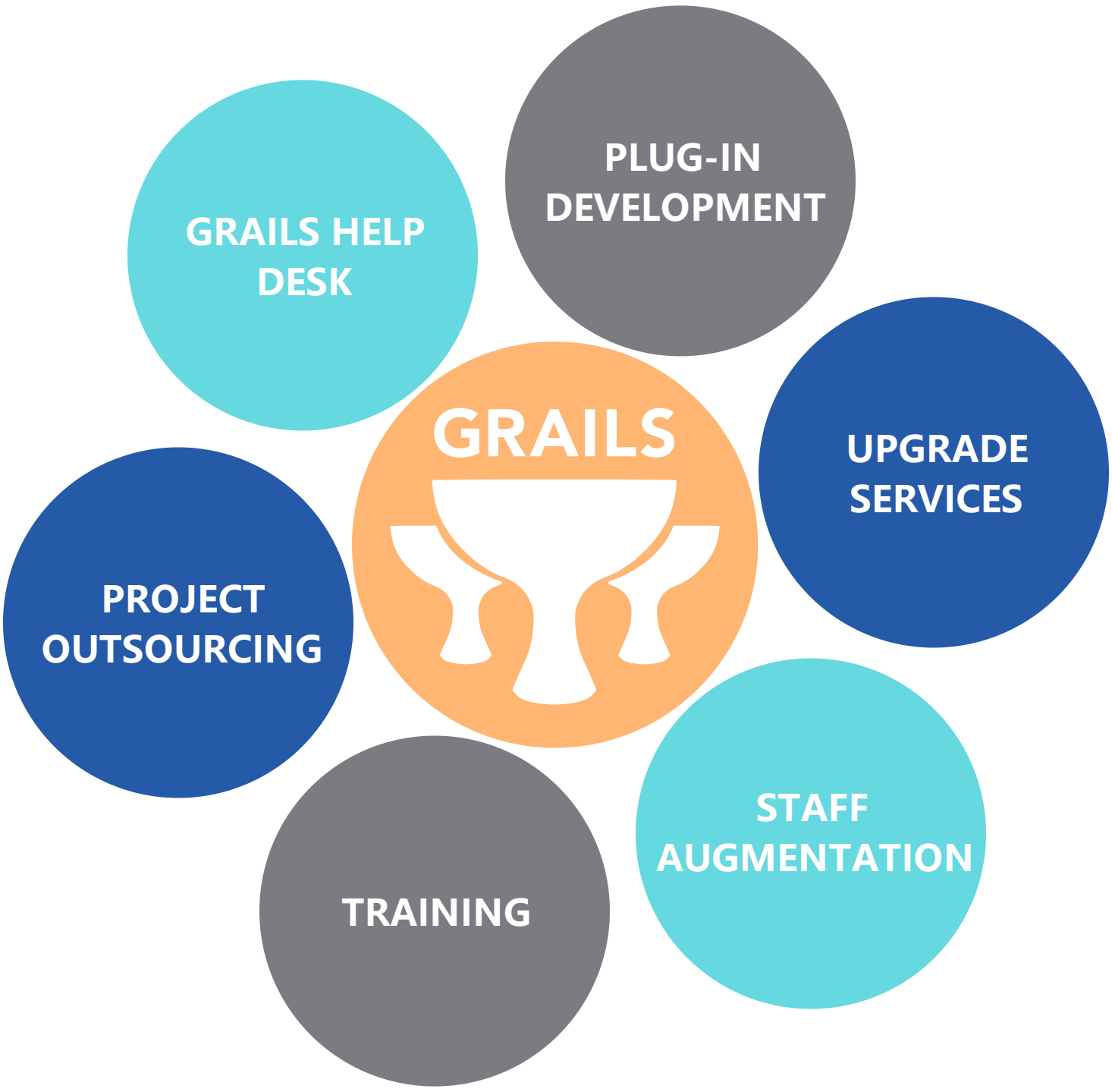
```
curl "http://localhost:8080/audi/vehicles"
```

得到回应:

```
["A3","A4"]
```

5您需要GORM或Grails的帮助吗?

改进这个文档



免费咨询

First Name *

Last Name *

Email *

Phone

Company

提交

该OCI Grails的团队 (https://objectcomputing.com/products/grails/consulting-support/) 包括Grails的联合创始人**杰夫•斯科特•布朗和格雷姆罗彻**。查看我们的Grails课程，(https://objectcomputing.com/training/catalog/grails/)并向开发，成熟和维护Grails的工程师学习。

Graeme Rocher
Grails Co-Founder
& Project Lead

Jeff Scott Brown
Grails Practice Lead

Paul King
OCI Groovy Project Lead


Álvaro Sánchez


James Kleeh


Dave Klein


Colin Harrington


Zachary Klein


Jack Frosch


Ryan Vanderwerf


Ben Rhine


Will Buck


Sergio del Amo


Iván López

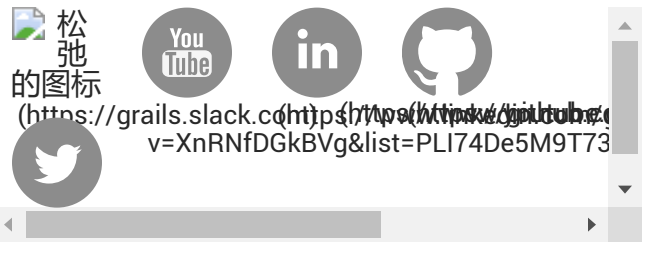

Matthew Moss


Nirav Assar


Dean Del Ponte

由...赞助

(https://objectcomputing.com/products/grails/)



Grails的存储库由Artifactory (http://artifactoryonline.com/)托管 (http://artifactoryonline.com/)

网站托管由Pivotal (https://run.pivotal.io/)提供 (https://run.pivotal.io/)

YourKit支持Grails及其 Java Profiler (https://www.yourkit.com/java/profiler/index.jsp)

Grails是开源 Apache 2许可证 (https://www.apache.org/licenses/LICENSE-2.0.html)