Grails 마스터하기: 첫 번째 Grails 애플리케이션 구축하기

Grails라는 작은 웹 프레임워크 패키지가 얼마나 강력한지 살펴보자

난이도 : 초급

Scott Davis, 편집장, AboutGroovy.com

2008 년 3 월 18 일

자바(Java™) 프로그래머는 최신 웹 개발 프레임워크에 적응하는 데 자신들이 가장 좋
없습니다. 자바 전문가인 Scott Davis의 새로운 연재인 *Mastering Grails*의 1회에서는 G
하는 방법을 설명하겠습니다.

또 다른 오픈 소스 웹 개발 프레임워크인 Ruby on Rails와 비슷한 Grails를 소개하겠다. 레일스가 처음 나왔을 때, 많은 개발자를 사로잡았다. 레일스의 스캐폴딩(scaffolding) 기능은 기존 방식에 드는 시간보다 훨씬 빠르게 새로운 프로젝트를 자체적으로 시작할 수 있게 해준다. 레일스의 '*설정보다 관례(convention over configuration)*'라는 아이디어는 성가시고 에러 발생이 쉬운 XML 설정 파일보다 명백한 이름 스키마를 기반으로 자체적으로 애플리케이션이 자동으로 묶이는 것을 의미한다. 루비의 메타 프로그래밍 능력은 놀랍게도 소스 코드를 어지럽히지 않고도 개체가 운영 시에 필요한 메서드와 개체를 상속하게 해준다.

레일스는 찬양과 숭배를 받은 (물론 지금도 받고 있는) 훌륭한 웹 프레임워크이지만, 자바 개발자들에게는 어려운 선택이다. 새로운 것을 위해 친숙한 자바 플랫폼을 어느 누가 포기하겠는가? 그리고 기존 자바 코드, 기존 운영중인 서버와 노련한 자바 개발자 직원들이 하는 일을 어떻게 대신 할 수 있을까?

Grails로 들어가보자. Grails는 검증된 자바 기술 기반을 그대로 유지한 채 레일스의 개발 경험을 줄 수 있다. 그러나 Grails는 단순하게 레일스를 자바 플랫폼에 "똑같이" 이식하고 있지는 않다. Grails는 레일스에서 배운 지혜를 현대적인 자바 개발의 섬세함과 조화시키고 있다. 레일스에서 *영감을 얻으려는 것*이지, 레일스로 *변경하는 것*이 아니다.

이번 연재에 대해

Grails는 '설정보다
친숙한 자바 개발
작성한 Grails는 스
서, 레거시 자바 코
Grails를 배운 후에
다.

Grails 마스터하기 연재를 시작하는 이번 글은 Grails 프레임워크를 소개하면서 설치하는 방법을 보여주고, 첫 번째 Grails 애플리케이션인 여행 계획 프로그램을 함께 구축한다. 여행 계획 프로그램은 앞으로 이어질 여러 회 동안 계속 작업을 이어나갈 것이다.

그루비의 힘

레일스가 루비 프로그래밍 언어와 깊은 관계를 맺고 있는 것처럼, Grails도 그루비(참고자료 참조)의 힘 없이 존재할 수 없다. 그루비(Groovy)는 JVM에서 실행되는 동적 언어로 자바 언어와 자연스럽게 통합된다. developerWorks에 오랫동안 연재된 *Practically Groovy* 를 읽어 본 적이 있다면 이미 그루비 언어의 막강한 힘에 친숙할 것이다. 읽은 적이 없어도 Grails를 배우는 데 걱정하지 않아도 된다. 그루비에 대해 전체적으로 많은 부분을 살펴 볼 것이기 때문이다. 그루비가 자바 개발자에게 매력적으로 다가갈 수 있도록 고안되었기 때문에 특별히 어렵거나 하진 않다.

예를 들어, 그루비는 일반적으로 작성해야 하는 자바 코드 양을 극적으로 줄일 수 있게 해준다. 더 이상 getter와 setter를 짤 필요가 없다. 그루비

가 자동으로 이를 제공하기 때문이다. 아이템에 대한 리스트에 반복 접근하기 위해 `for Iterator i = list.iterator()` 를 더 이상 작성할 필요 없이, `list.each`라고 쓰면 좀 더 간결하고, 의미 있는 표현으로 같은 동작을 할 수 있다. 간단히 말하자면 이렇다. 21세기에 자바를 개발한다면 오늘날 그루비의 모습과 같을 것이다.

기반이 되는 모든 애플리케이션을 다시 작성해야 한다면, 그루비를 이용하는 것이 자바 개발자에게 별로 매력적이지 않을지도 모른다. 다행히도 그루비는 기존 코드들과 자연스럽게 통합된다. 그루비는 자바 언어를 교체하는 것이 아니라 상승 효과를 가져오는 것이다. 결국 마지막에 가서는 그루비 코드가 자바 코드가 *되므로* 지체 없이 그루비를 선택할 수 있다. 이 두 언어는 .java 파일을 .groovy 파일로 이름만 바꾸는 작업으로도 완벽하게 상호 호환이 된다. 예를 들어, Person.java를 Person.groovy로 변경함으로써 완벽하게 유효한(그리고 실행 가능한) 그루비 파일(비록 그루비가 제공하는 달콤한 문법을 이용하지는 않지만)이 된다.

그루비와 자바 언어 사이에 이렇게 깊은 수준까지 상호 호환된다는 것은 오늘날 IT 현장에서 쓰는 핵심 기술을 Grails로 다시 고안해야 하는 일이 필요 없음을 의미한다. 대신 클래스의 그루비화된 표현들을 통해 친숙한 자바 라이브러리를 사용할 수 있다. JUnit `TestCase`는 그루비에 래핑되어 `GroovyTestCase`로 등장한다. Grails에서는 Ant를 순수 그루비로 구현해 새로 구성한 GANT를 제공한다. Grails는 그루비로 하이버네이트(Hibernate)를 래핑해 이를 GORM(Grails Object/Relational Mapper)이라 한다. 이것들은 단지 Grails를 통해 현대적인 웹 개발 실천법의 장점을 이용하면서 기존 자바 경험을 활용하는 방법에 대안 예를 세 가지만 든 것뿐이다.

Grails에 매우 감사하며, 여기서 우리는 첫 번째 경험을 하게 된다. 이제 Grails를 설치하고 첫 번째 웹 애플리케이션을 만들어 보자.

Grails 설치하기

Grails 애플리케이션을 실행하는 데는 단지 ZIP 파일 하나만 필요하다. 의존성이 있는 모든 라이브러리, 몇 가지 예를 들면 그루비, 스프링(Spring), 하이버네이트 등은 이미 파일 안에 들어가 있어 사용할 준비가 되어 있다. Grails를 설치하는 방법은 다음과 같다.

1. Grials 사이트(참고자료 참조)에서 grails.zip을 다운로드하고, 압축 풀기
2. `GRAILS_HOME` 환경 변수 생성
3. `PATH`에 $GRAILS_HOME/bin 추가

좋다. 이제 JDK를 설치해야 한다(Grails는 좋지만, JDK 설치는 별로다). Grails 1.0은 자바 1.4, 1.5와 1.6에서 실행된다. 설치된 JDK 버전을 모르면 프롬프트 화면에서 `java -version`을 입력하자. 필요하다면 Grails에 적합한 JDK를 다운로드해 설치하자(참고자료 참조).

설치 단계를 끝냈다면, `grails -version`을 입력해 확인해 보자. 친숙한 화면이 나오면 모든 설치가 올바르게 구성되었다고 보면 된다.

```
Welcome to Grails 1.0 - http://grails.org/
Licensed under Apache Standard License 2.0
Grails home is set to: /opt/grails
```

포함되어 있는 웹 서버와 데이터베이스

흥미롭게도 Grails 애플리케이션을 실행하는 데 웹 서버를 별도로 설치할 필요는 없다. Grails에는 임베드된 제티(Jetty) 서블릿 컨테이너가 들어 있다. `grails run-app`를 입력하면, 일반적인 배치 후프(hoop)를 통해 시작할 필요 없이 제티 컨테이너에서 애플리케이션을 실행할 수 있다. 기존에 사용하던 운영 서버에서 Grails 애플리케이션을 실행하는 것 또한 전혀 문제가 없다. `grails war`를 입력해 톰캣, JBoss, 제로니모, WebSphere®나 기타 자바 EE 2.4 기준에 부합하는 서버에서 배포 가능한 표준 파일을 생성할 수 있다.

공짜 프로그램 사용

이번 글의 애플리케
터베이스를 사용한
운영할 수 있도록
안에는 grails.org를
(참고자료 참조)를

Grails에서는 개별 데이터베이스를 설치할 필요가 없다. Grails는 순수 자바 기반 데이터베이스인 HSQLDB(참고자료 참고)를 운영하기 때문이다. 생산성을 일순간에 향상시킬 수 있도록 데이터베이스를 사용할 준비를 해두었다. MySQL, PostgreSQL, 오라클, 혹은 DB2와 같은 또 다른 데이터베이스를 사용하는 것은 하이버네이트와 GORM에게는 매우 간단한 일이다. JDBC 드라이버 JAR와 일반적인 커넥션 설정만 직접 해준다면, DataSource.groovy 파일 하나만 수정해도 별도 시간을 들이지 않고 원하는 데이터베이스를 사용할 수 있다.

첫 Grails 애플리케이션 작성하기

필자는 1년에 적어도 40군데 이상을 돌아다니며 많은 여행을 즐긴다. *언제* 어디로 갈 필요가 있는지 알려주는 매우 훌륭한 일을 해주는 달력을 찾았지만, 그 달력은 그 장소가 *어디에* 있는지 까지는 알려주지 못했다. 온라인 지도는 정반대의 문제를 가지고 있다. 어디에 있는지 매우 잘 알려주지만, 언제 가야 하는지는 전혀 알려주지 못한다. 이런 점 때문에 이번 연재의 이번 회와 다음 회에서 여행을 계획하는 데 시간과 장소를 모두 도움을 줄 수 있는 맞춤 Grails 애플리케이션을 함께 만들어 볼 것이다.

스포일러 경고

이번 연재에서 구
을 앞으로 볼 수 있

시작하면서 디렉터리를 깨끗하게 정리하고, `grails create-app trip-planner`를 입력하자. 정신 없이 많은 작업이 수행된 후, trip-planner라는 이름으로 된 디렉터리를 확인해 보자. 메이븐(Maven), 레일스와 AppFuse처럼 Grails 역시 개발자를 위해 표준 디렉터리 구조를 잡아준다. 자신만의 맞춤 디렉터리 트리 디자인을 꼼꼼하게 하는 것이 아닌데도, 이 제약이 말도 안 되는 행동을 강요해 프레임워크로 작업할 수 없다고 느낀다면, Grails로 작업을 하는 것이 전혀 즐겁지 않을 것이다. '설정보다 관례'에서 *관례*에 익숙해지면 부분이 모든 Grails 애플리케이션에서 어떤 파일이 어떤 디렉터리에 있는지 즉시 알 수 있게 된다.

trip-planner 디렉터리로 변경하고, `type grails create-domain-class Trip`을 입력하자. 모든 작업이 정상적으로 종료되면, 새로 생성된 파일인 grails-app/domain/Trip.groovy와 grails-app/test/integration/TripTests.groovy가 기다리고 있을 것이다. 이에 대한 테스트는 차후 글에서 다룰 것이다. 여기서는 Listing 1처럼 보이는 소스로 시작하는 도메인 클래스에 집중할 생각이다.

Listing 1. Grails가 생성한 도메인 클래스

```
class Trip{

}
```

이렇게 보기만 해선 모르겠다고? Listing 2처럼 `Trip` 클래스에 필드를 추가해보자.

Listing 2. 필드를 추가한 `Trip` 클래스

```
class Trip {
  String name
  String city
  Date startDate
  Date endDate
  String purpose
  String notes
}
```

앞서 언급한 것처럼 그루비가 getter와 setter를 동적으로 생성해주기 때문에 getter와 setter 생성은 걱정할 필요가 없다. `Trip`은 새롭고, 유용한 다수의 동적 메서드를 가지고 있는데 이름만 보면 어떤 메서드인지 쉽게 알 수 있다.

- `Trip.save()`는 HSQLDB 데이터베이스의 `Trip` 테이블에 데이터를 *저장한다*.
- `Trip.delete()`는 `Trip table` 테이블에 있는 데이터를 *삭제한다*.
- `Trip.list()`는 `Trip` 목록을 반환한다.
- `Trip.get()`은 단일 `Trip`을 반환한다.

이 모든 메서드가 필요하다면 조금 더 간단하게 만들 수 있다. `Trip` 클래스가 상위 클래스를 상속하지 않고, 다른 신비로운 인터페이스를 구현하는 것이 아님을 상기하자. 매우 간편하게도 이 메서드들이 적절한 클래스의 적절한 위치에 나타나도록 해주는 그루비의 메타 프로그래밍 능력에 감사의 말을 전한다.

컨트롤러와 뷰 구축하기

도메인 클래스를 생성한 것은 단순히 이 전쟁의 절반에 지나지 않는다. 더욱 완벽하게 구축하려면 모든 모델에 대한 좋은 컨트롤러와 일부 뷰가 필요하다(독자들이 MVC 패턴에 친숙하다고 가정한다. 참고자료를 참조하자). `grails generate-all Trip`을 입력해 grails-app/app/controllers/TripController.groovy 클래스 및 그에 대응하는 GSP (Groovy Server Pages)를 grails-app/views/Trip에 빌드한다. 컨트롤러에 있는 모든 list 액션은 `list` 파일에 대응된다. `create` 액션은 create.gsp를 가지고 있다. 여기에서 설정보다 관례의 장점을 잘 알 수 있다. 이 요소들을 일치시키기 위해 XML 파일이 전혀 필요하지 않다. 모든 도메인 클래스는 이름으로 컨트롤러와 짝을 이룬다. 또한 컨트롤러에 있는 모든 액션은 이름으로 뷰와 짝을 이룬다. 원한다면 이름에 기반을 둔 설정을 무시할 수 있지만, 대부분 관례에 따르는 것이 좋다. 이렇게 되면 애플리케이션은 즉시 작동하게 된다.

grails-app/controller/TripController.groovy를 보자(Listing 3).

Listing 3. `TripController`

```
class TripController {
    ...
    def list = {
        if(!params.max) params.max = 10
        [ tripList: Trip.list( params ) ]
    }
    ...
}
```

자바 개발자가 알아야 할 첫 번째 점은 몇 줄 안 되는 코드로 얼마나 많은 것을 달성할 수 있느냐다. 예를 들어, `list` 액션을 들 수 있다. 메서드 내에서 중요한 행동은 마지막 줄에 나온다. Grails는 `tripList`로 명명된 단일 요소를 갖는 hashmap을 반환한다(그루비 메서드 마지막 줄에는 암묵적으로 반환문을 포함한다. 또한 원한다면 직접 `return` 문을 입력할 수도 있다). `tripList` 요소는 `Trip.list()` 메서드를 사용해 데이터베이스에서 가져온 `Trip` 개체에 대한 `ArrayList`다. 일반적으로 이 메서드는 테이블에 있는 모든 레코드를 반환한다. 이 라인은 이렇게 말하고 있다. "이봐, URL에 최대 인자를 누구라도 전달하면, 반환하는 `Trip`들의 개수를 제한하는 데 사용해. 받은 인자가 없다면, Trip 수를 10개로 제한해." http://localhost:8080/trip-planner/trip/list URL은 위 액션을 호출한다. 예를 들어, htpp://localhost:8080/trip-planner/trip/list?max=3은 일반적인 10개의 결과 대신에 3개의 `Trip`을 보여준다. 더 많은 trip을 보여주고 싶다면, Grails는 앞, 뒤 페이지로 연결되는 링크를 자동으로 생성해 줄 것이다.

그럼 이 hashmap이 어디서 사용되는 걸까? Listing 4에서 나오는 grails-app/views/list.gsp를 확인해 보자.

Listing 4. list.gsp

```
<g:each in="${tripList}" status="i" var="trip">
  <tr class="${(i % 2) == 0 ? 'odd' : 'even'}">
    <td>
      <g:link action="show" id="${trip.id}">${trip.id?.encodeAsHTML()}</g:link>
    </td>
  </tr>
</g:each>
```

list.gsp는 GroovyTagLibs의 맛보기만 보여주는 가장 일반적이고 평범한 HTML이다. `g:`로 시작하는 어떤 것이든지 GroovyTag가 된다. Listing 4에 있는 `<g:each>`는 `tripList ArrayList`에 있는 각 `Trip`을 돌려주고, 제대로 구성된(well-formed) HTML 테이블을 만들어 준다.

컨트롤러를 이해하려면 최종적으로 *return*, *redirect*, *render*(3*R*s)를 제대로 이해해야 한다. 같은 이름의 GSP 페이지에서 데이터를 반환하는 반환문을 내부적으로 사용하고 있다. 일부 액션은 재지정(redirect)을 한다. 예를 들어, URL에 특정 값을 정하지 않았다면, `index`는 호출시 받은 액션을 수행한다.

```
def index = { redirect(action:list,params:params) }
```

이런 경우에 TripController는 list 액션으로 재지정되고, params hashmap에 인자(혹은 QueryString)를 모두 함께 전달한다.

마지막으로 save 액션(Listing 5 참조)은 대응되는 save.gsp 페이지를 가지고 있지 않다. 에러 없이 데이터베이스에 저장된 레코드를 받았다면 show 액션 페이지로 재전송된다. 그렇지 않다면 에러와 함께 다시 시도할 수 있도록 create.gsp 페이지가 보인다.

Listing 5. save 액션

```
def save = {
  def trip = new Trip(params)
  if(!trip.hasErrors() && trip.save()) {
    flash.message = "Trip ${trip.id} created"
    redirect(action:show,id:trip.id)
  }
  else {
    render(view:'create',model:[trip:trip])
  }
}
```

Grails가 어떻게 작업을 하는지에 대해 더 이야기해 보는 것보다는, 실제 실행해보면서 살펴보자.

애플리케이션 실행

명령행에 grails run-app를 입력하자. Log4j 메시지가 정신 없이 콘솔에 출력된 후에, 다음 메시지가 나온다.

```
Server running. Browse to http://localhost:8080/trip-planner
```

이미 8080 포트로 서버가 실행되고 있다면, 코어 덤프(core dump) 맨 마지막에서 다음과 같은 잔소리를 듣게 될 것이다.

```
Server failed to start: java.net.BindException: Address already in use
```

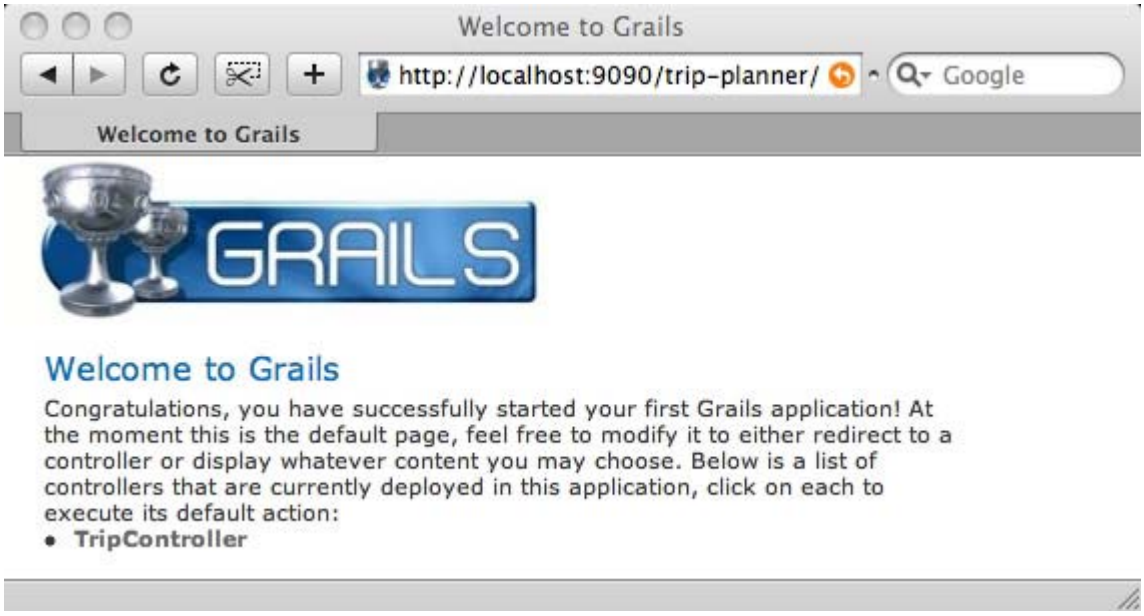
두 가지 방법으로 제티가 실행되는 포트를 쉽게 바꿀 수 있다. grails -Dserver.port=9090 run-app로 임시로 변경할 수 있다. 완전히 바꾸려면 $GRAILS_HOME/scripts/Init.groovy에서 serverPort로 시작하는 줄을 살펴보고, 값을 변경하면 된다.

```
serverPort = System.getProperty('server.port') ?
             System.getProperty('server.port').toInteger() : 9090
```

일단 선택한 포트로 Grails가 동작중이라면 웹 브라우저에 URL을 입력

하자. 그림 1에서 볼 수 있는 모든 컨트롤러에 대한 리스트가 있는 환영 화면을 볼 수 있다.

그림 1. Grails 애플리케이션 환영 화면



TripController 링크를 클릭하자. 지금 곧바로 다룰 수 있는 완전한 CRUD(create, read, update, delete) 애플리케이션을 갖고 있다.

그림 2에 보이는 페이지를 사용해 새로운 trip을 생성하자.

그림 2. trip 페이지 만들기

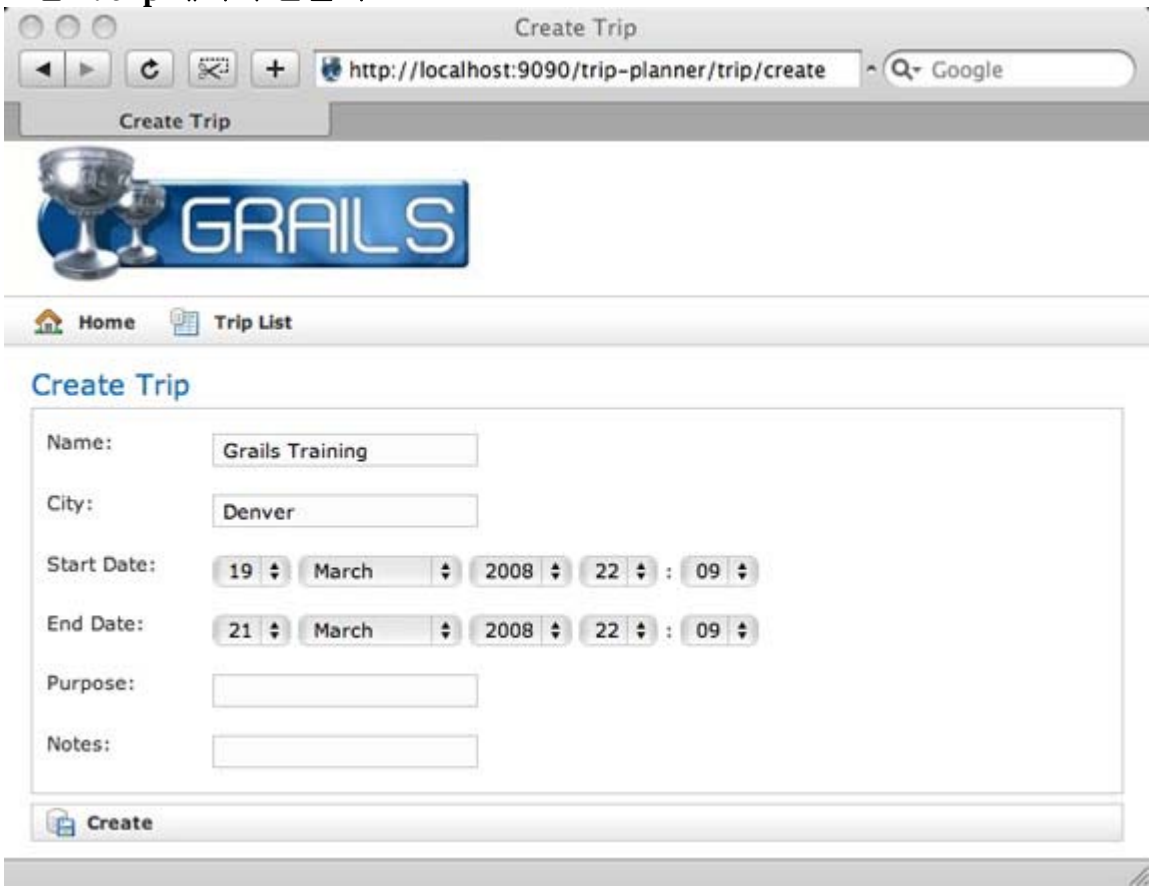
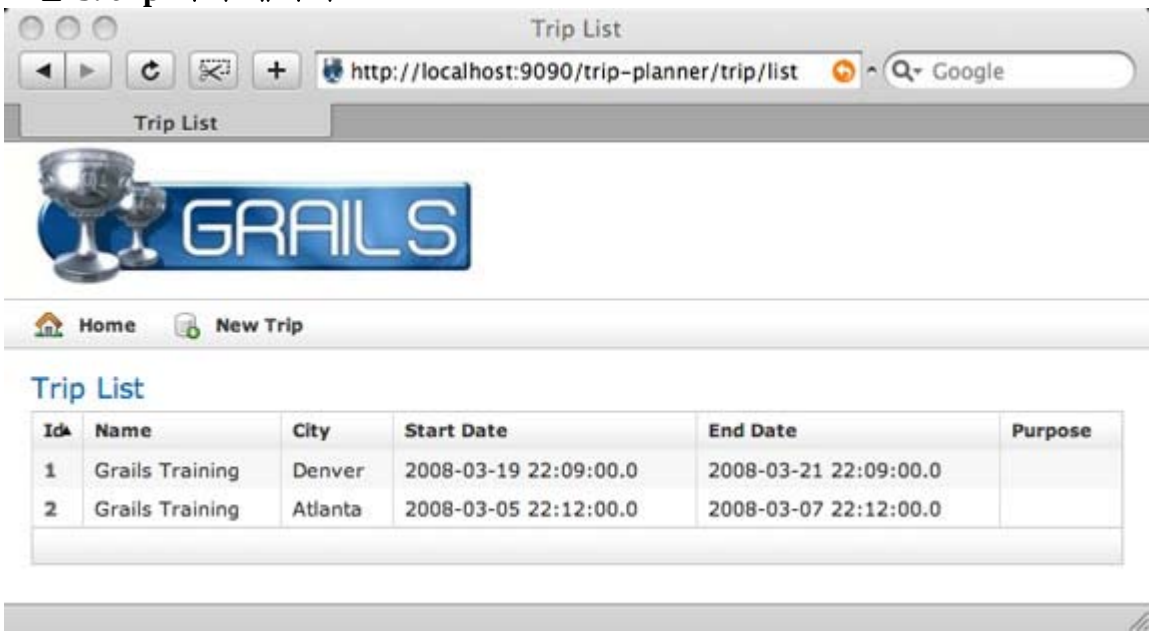


그림 3에서 보이는 화면을 사용해 trip을 수정하자.

그림 3. trip 목록 페이지



애플리케이션을 만들고 실행하는 데 어느 정도의 시간이 걸렸을까? 이를 달성하기 위해 얼마나 많은 코드가 들어갔을까? 이를 찾을 수 있는 방법이 있다.

1. Ctrl-c를 입력해 Grails를 종료하자.
2. `grails stats`를 입력하자.

화면에 아래 내용이 출력될 것이다.

```
+----------------------+-------+-------+
| Name                 | Files |  LOC  |
+----------------------+-------+-------+
| Controllers          |     1 |    66 |
| Domain Classes       |     1 |     8 |
| Integration Tests    |     1 |     4 |
+----------------------+-------+-------+
| Totals               |     3 |    78 |
+----------------------+-------+-------+
```

애플리케이션의 기능을 구현한 코드가 전부 100줄도 안 된다. 나쁘지 않다. 그러나 마치기 전에 몇 가지 트릭을 더 보여주겠다.

컨트롤러와 뷰를 생성해 보는 것은 매우 좋은 학습 과제다. 어떻게 모든 것이 연동되는지 쉽게 알 수 있게 디스크에 물리적인 파일을 가지고 있는 것이 좋다. 그러나 나는 `TripController` 클래스 내용을 삭제하고, 아래 내용으로 교체하는 것을 더 좋아한다.

```
class TripController{
  def scaffold = Trip
}
```

위 단 한 줄의 코드를 보면 Grails가 동작시에 동적으로 메모리에 `list`,

save, edit하는 액션을 모두 생성해 이전 컨트롤러가 해냈던 모든 일을 하는 것을 알 수 있다.

grails run-app를 다시 입력하자. 맞다. 모든 데이터는 삭제되었다. 걱정하지 말라. Ctrl-C를 눌러 Grails를 종료하자. 이번에는 grails prod run-app를 입력하자. 이제 프러덕션 모드에서 실행되는데, 기존 데이터가 서버가 재시작되는 사이에 저장됐음을 의미한다. TripController를 통해 작업을 선택하고, 추가로 레코드를 저장하자. 애플리케이션 행동에서는 별다른 점을 보지 못할 수도 있다. 지금 브라우저에서 보는 모든 것이 단지 Grails 코드 열다섯 줄에서 나온 힘이라는 생각을 기억하자.

결론

Grails의 첫 번째 맛보기를 즐겼기를 바란다. 이 엄청난 기능이 작은 패키지에 담겨 있고, 단지 살짝만 건드리면 원하는 기능이 구현된다는 것은 정말 놀라운 일이다. 프레임워크 설치는 파일 압축을 해제하는 정도다. 단순히 약간 명령어만 입력해 무에서 애플리케이션을 생성해냈다. Grails가 좀 더 매력적으로 다가오는 힘찬 여정이 되길 바란다. 이 예제에서 더 나아가 새롭고 흥미로운 방향을 찾을 수 있는 시간이 됐을 것이다.

다음 회에서는 GORM과 함께 좀 더 의미 있는 시간을 보낼 것이다. MySQL 데이터베이스에 사용할 데이터를 저장하고, 적절한 데이터인지 유효성 검증을 넣고, 일 대 다 관계를 설정해볼 것이다. 많은 코드 추가 없이 여행 계획 애플리케이션의 능력을 두드러지게 향상시킬 수 있다.

그 때가 되면 그루비와 Grails가 함께 하는 것이 매우 재미있어질 것이다. 다시는 결코 이전과 같은 방법으로 웹 개발을 하고 싶지 않을 것이다.

참고자료

교육

- Grails: Grails 웹 사이트를 방문해보자.
- *Practically Groovy:* 이 developerWorks 연재는 그루비의 실질적인 사용방법을 살펴보고
- 그루비: 프로젝트 웹 사이트에서 그루비에 대해 더 배워보자.
- AboutGroovy.com: 최신 그루비 뉴스와 관련 글 링크가 있다.
- "What's the secret sauce in Ruby on Rails?" (Bruce Tate, developerWorks, 2006년 5월): develop 재에 있는 이 글은 Big splash처럼 레일스를 만든 이유를 설명하고 있다.
- HSQLDB와 제티: Grails에 탑재된 순수 자바 기반 데이터베이스와 서블릿 컨테이너
- 모델-뷰-컨트롤러(Model-View-Contrller): Grails가 준수하는 대중적인 아키텍처 패턴
- 기술 서적 서점에서 Grails와 관련된 책과 기술적인 주제를 찾아보자.

- 한국 developerWorks 자바 기술 존: 자바 프로그래밍의 다양한 측면에 대해 작성된 수많은

제품 및 기술 얻기

- Grails: 최신 Grails 배포 버전을 다운로드하자.
- 자바 SE: 사용하는 Grails에 적합한 JDK를 다운로드하자.
- 자바 플랫폼을 위한 IBM developer kits: IBM developer kit으로 Grails를 사용하고 싶다면,

토론

- 한국 developWorks 블로그를 확인해보고, 한국 developWorks 커뮤니티에 참여해보자.

필자소개

Scott Davis는 국제
저서로는 *Groovy I*
Developers: Adding
*At Work*가 있다.

이 문서 북마킹 하기

mar.gar.in naver eolin del.icio.us