General FAQ

또 다른 FAQ 에서는 현 버전의 프레임워크에 관한 상세한 질문들을 다루고 있다.

목차

1. REST 프레임워크가 필요한 이유는 무엇인가?

프레임워크의 목표는 여러분의 애플리케이션을 좀 더 효율적으로 만드는데 있어 기반으로 사용될 재사용가능하고 확장가능한 클래스와 인터페이스들을 제공하는 것이다. Restlet 프로젝트는 개발자들이 RESTful 한 애플리케이션을 개발하는데 사용할 수 있는 자바 프레임워크가 없었음에 주목하여 시작되었다.

먼저 여러분 자신이 REST 아키텍처 스타일에 익숙해지는 것이 중요하다. 그러고 나면 여러분은 Restlet API 의 클래스명이 REST 관련 개념들(리소스, 리프레젠테이션, 커넥터, 컴포넌트, 미디어 타입, 언어 등)로부터 그대로 가져왔다는 것을 알게 될 것이다. 이는 여러분이 RESTful 하게 사고할 수 있으며 여러분의 코드를 좀 더 자연스럽게 솔루션으로 고칠 수 있음을 의미한다.

만약 여러분이 아직 REST 에 익숙치 않더라도 Restlet 에서 제공하고 있는 프로토콜 추상화를 얻을 수 있는데, 동일한 인터페이스들을 여러 가지 프로토콜(HTTP 서버, HTTP 클라이언트, SMTP 클라이언트, JDBC 클라이언트, 파일 시스템…)에서도 사용할 수 있다. 예를 들어, 여러분은 content negotiation 을 수행하기 위하여 HTTP 헤더를 알아야 할 필요는 없다. 게다가 여러분은 표준 웹 애플리케이션을 구현하기 위해 여러 API 들(예, 서블릿, HttpUrlConnection, JavaMail)을 사용하고 익혀야 할 필요도 없다.

2. 기반 개념은 무엇인가?

프로젝트명이 의미하고 있듯, Restlet 클래스는 이해해야 할 핵심적인 개념이다. Restlet 클래스는 RESTful 유니폼 클래스를 확장하며 커넥터, 컴포넌트, 애플리케이션, 파인더 등의 기초가 된다. Restlet 클래스는 프레임워크의 모든 부분들에 대한 계약(contract)인 “public void handle(Request request, Response)”이라는 하나의 메소드만을 포함하고 있다.

Request 와 Response 클래스가 중요한 것은 두 클래스가 모든 측면의 RESTful 호출을 캡슐화하기 때문이다. 모든 REST 개념들이 구현되어 있으며 REST 에서 사용하는 이름을 그것에 상응하는 자바 인터페이스의 이름으로

사용하려고 한다. URI 기반의 라우팅이나 대상 리소스 식별에 대한 호출을 처리하기 위해 Application, Filter, Finder, Router, Route 와 같은 몇 가지 특수한 Restlet 이 추가되었다.

REST 호출을 처리하는 것은 일반적으로 두 가지 과정을 거쳐 이루어진다. 첫 번째 과정에서 REST 호출은 REST 호출을 필터링(보안목적이나 URI 로부터 변수들을 추출하기 위한) 하기 위한 연쇄적인 Restlet 들을 처리하여 대상 리소스를 위치시키는데, 이 리소스는 URI 에 의해 식별된다. 파인더(Finder) 클래스는 연쇄 호출을 처리하는 과정의 마지막 Restlet 이며 실질적으로 처리된 요청 정보를 근거로 리소스 인스턴스를 찾아내는 책임을 맡고 있다. 이 과정이 완료되면 실질적인 처리과정은 대상 리소스 클래스, 혹은 여러분이 생성할 그 리소스 클래스의 하위 클래스 중의 하나에서 이루어질 것이다. 리소스와 리프레젠테이션의 개념들은 사실 REST 의 핵심이며, 따라서 Restlet API 의 핵심이다. 하나의 리소스는 여러 가지 리프레젠테이션을 다양한 형식, 언어 등으로 노출할 수 있다. 여러분의 리소스에 대한 최적의 리프레젠테이션을 반환하도록 하기 위해 자동적이고 투명한 content negotiation 알고리즘이 사용된다.

3. 서블릿 대신 Restlet 을 사용하는 것의 이점은 무엇인가?

여러 가지 이점이 있긴 하지만 가장 주요한 것은 Restlet 이 HTTP 프로토콜의 공저자인 Roy T. Fielding 이 정의한 대로의 REST 아키텍처 스타일을 엄격하게 따르고 있다는 점이다. 이는 여러분의 애플리케이션이 손쉽게 RESTful 한 방식으로 개발될 수 있음을 보장한다. REST 의 각 개념들은 그것에 대응되는 자바 클래스들을 Restlet API 에 포함하고 있다. 또한 Restlet 은 RESTful 하지 않으며 서블릿 애플리케이션의 가용성과 확장성의 주요 병목 지점이기도 한 인메모리 사용자 세션(in-memory user session)의 개념을 없앴다.

또 다른 이점으로는 정확히 동일한 API 를 사용하는 여러 가지 프로토콜(HTTP, JDBC, SMTP 등)과 함께 작동하는 능력이다. 이것은 클라이언트측과 서버측 애플리케이션 모두에 해당된다. 추가적으로 Restlet 애플리케이션은 동시에 클라이언트와 서버가 될 수도 있다. Restlet 은 진정한 웹 중심의(Web centric) 프레임워크이다. 만약 Restlet 을 서블릿 API 에 견주어 보자면 Restlet 은 서버측 코드를 지원하는 HTTP 전용 API 라 할 수 있다. 클라이언트측 코드를 지원하고자 한다면, 여러분은 전혀 다른 API 인 JDK 의 저수준 HttpURLConnection 클래스나 아파치 HTTP 클라이언트 프로젝트에 의존할 필요가 있을 것이다.

또한 Restlet API 는 애플리케이션 측면으로부터 깔끔하게 저수준 프로토콜의 측면을 분리시킨다. 대부분의 애플리케이션에서는 여러분이 요청의 미디어 타입을 알아내기 위해 HTTP 헤더에 접근할 필요는 없으며, 단순히 request.getEntity().getMediaType()을 사용하기만 하면 된다. 이러한 분리는 서버측 content negotiation 에 대한 자동 지원을 가능케 하며, 또한 프로토콜 커넥터로 하여금 애플리케이션 코드와 충돌할 필요 없이 모든 최적화를 구현할 수 있도록 해준다. 서블릿을 사용할 경우 애플리케이션에서 직접적으로 HTTP 헤더를 변경하고 출력 스트림을 제어하기 때문에 모든 것이 섞여버린다.

추가적으로 라우터(Router)와 라우트(Route)를 사용하여 Restlet API 는 호출 처리에 대한 완전한 제어권을 개발자에게 부여한다. 서블릿 세계에서의 표준화된 접근방법은 컴포넌트의 설정 파일에 들어있는 URI 매핑을 설정하는 것이며, 이는 유연하지 못하다. 언제라도 수동으로 디스패치를 처리하거나 스트러츠나 스프링과 같은 써드 파티 프레임워크에 의존해야할 가능성이 있으며, 이렇게 하는 것은 Restlet API 에서 제공하고 있는 통합된 지원만큼 편리하고 강력하지는 않다.

마지막으로 Restlet API 는 I/O agnostic 한데, 왜냐하면 핵심 리프레젠테이션 인터페이스는 BIO 스트림(java.io 패키지)과 NIO 채널(java.nio 패키지)와 동등하게 작동할 수 있기 때문이다. 이는 완전히 NIO 에 컴플라이언트한 커넥터를 사용할 수 있을 경우 상당한 성능 향상이 이루어질 수 있는 가능성을 가져다 준다. 이렇게 될 경우의 주요 이점은 현 서블릿 컨테이너가 가지고 있는 주요 병목지점인 각각의 열린 커넥션에 대한 하나의 쓰레드에 연결할 필요성을 제거함으로써 확장성이 증가할 것이라는 점이다.

기존 서블릿 컨테이너를 이용하거나 좀 더 규모가 큰 서블릿 애플리케이션에 통합하고자 할 경우, 여러분은 Restlet API 을 써드 파티 서블릿 라이브러리로서 사용하게 해줄 경량 어댑터도 사용할 수 있다.

3. REST 에 관한 일반적인 질문들에 대해서는 어디에서 도움 받을 수 있나?

웹 상에는 REST 와 관련된 수많은 참고자료들이 있다. 이것들은 종종 찾기 어렵기 때문에, 우리는 구글에서 제공해주는 전문화된 검색 엔진을 마련해 두고 있다. 추가적으로 아래는 인기있는 링크들을 나열한 것이다:

- REST page on Wikipedia
- How I explained REST to my wife...
- Thesis chapter defining REST: English version, French translation
- Architecture of the World Wide Web: English version, French translation
- REST Resources of Paul Prescod
- Building Web Services the REST Way
- REST Wiki, including a FAQ
- REST discussion mailing list
- HTTP 1.1 specification, a common application of REST
- WebDAV specification, an extension of HTTP

4. 루비 온 레일즈와 비교하자면?

공평하게 비교하기란 언제나 어려운 일인데, 여러분이 어느 한쪽 편 출신이고 다른 쪽에 대해서는 상세한 지식을 갖고 있지 않을 때 특히 그러하다. 우리가 일반적으로 느끼고 있는 것은 레일즈는 그 핵심에 있어 REST 원칙을 준수하여 만들어지지는 않았다는 것이다. 그러한 원칙들은 1.2 버전에서 다소 바뀌긴 하였다. 또한 새로운 버전에서 REST 를 지원하는 것 역시 여러가지 데이터베이스 스키마와 URI 네임스페이스상의 제약사항들을 동반한다.

그에 반해 Restlet 프로젝트는 명시적으로 REST 개념을 염두에 두고 구축되었으며 투명한 content negotiation 이나 디렉터리(mini-WebDAV 모드로도 알려져 있는) 노출시 PUT/DELETE 메소드에 대한 자동 지원과 같은 주요 기능들을 자연스럽게 지원한다. 게다가 여러분은 여러분의 리소스에 대해 어떠한 퍼시스턴스 솔루션도 자유로이 선택할 수 있다.

7. 장고(Django)와 비교하자면?

다시 한번 말하지만 공평하게 비교하기란 언제나 어려운 일이며, 여러분이 어느 한쪽 편 출신이고 다른 쪽에 대해서는 상세한 지식을 갖고 있지 않을 때 특히 그러하다. 우리가 일반적으로 느끼고 있는 것은 장고 그 핵심에 있어 REST 원칙을 준수하여 만들어지지는 않았다는 것이다. 장고 프로젝트는 아직 1.0 파이널 버전까지 도달하지는 않았지만 아직까지는 아무런 특정 REST 지원도 하고 있지는 않아 보인다. 그럼에도 불구하고 장고를 일반적인 웹 프레임워크로서 RESTful 한 애플리케이션을 만드는데 완벽하게 사용할 수 있다.

예를 들면 장고는 기본적으로 URI 템플릿을 지원하지 않으므로 여러분이 직접 그것들을 정규 표현식으로 변환할 필요가 있다. 또한 만약 여러분이 HTTP 캐싱이나 인증이 가능하게 만들고자 할 경우에는 여러분이 직접 “Last-

modified"나 "WWW-Authenticate" 헤더를 설정해야만 한다. 여러 가지 비슷한 프레임워크와 같이 장고는 여러분이 RDBMS 를 사용하여 유일한 솔루션이 아니거나 최적이 아닌 리소스를 영속화하고 있다고 가정한다.

그에 반해 Restlet 프로젝트는 명시적으로 REST 개념을 염두에 두고 구축되었으며 투명한 content negotiation 이나 디렉터리(mini-WebDAV 모드로도 알려져 있는) 노출시 PUT/DELETE 메소드에 대한 자동 지원과 같은 주요 기능들을 자연스럽게 지원한다. 게다가 여러분은 여러분의 리소스에 대해 어떠한 퍼시스턴스 솔루션도 자유로이 선택할 수 있다.