13장

Ajax 기반 매시업(Mashup) 제작

: '매시업'이란 무엇인가?

: 데이터 형식에 대한 결정

: 서버로부터 로드된 XML 데이터의 수용

: 여러 오픈 API를 동시에 사용하기

'매시업 하기'는 으깬 감자 요리를 만들기 위해 부엌으로 뛰어들려고 하는 것을 의미하는 것이 아니며[1] 또한 열광적인 최신 춤을 추기 위해 조명이 비치는 플로어에 나가려고 하는것을 의미하는 것도 아니다.[2] 웹 애플리케이션에 적용되었듯이, 매시업은 여러 곳으로부터 얻어진 컨텐트를 결합한 웹 페이지이다. '매시업'이라는 용어가 꽤 새로운 것이긴 하지만, 개념은 전혀 새로운 것이 아니다. Ajax의 출현으로 Open API(12 장 참조)의 사용이 많아지고, 매시업을 더 쉽게 만들게 하고 있다.

이 장에서는 3장과 4장에서 검토하고 이 책 전반에서 사용한 Prototype 자바스크립트 라이브러리뿐만 아니라, 12 장에서 습득한 Open API에 대한 여러분의 지식을 한층 더 강화하고, 야후 지도와 플리커의 Open API를 사용하여 여러분만의 매시업을 제작해 볼 것이다.

만약 여러분이 3장과 12장을 아직 읽지 않았다면, 이 장을 더 진행하기에 앞서 돌아가서 읽는 것이 좋겠다. 특히 Prototype, 야후 지도와 플리커에 대한 내용을 자세히 보아둘 필요가 있다.

13.1 여행기록 공유 애플리케이션[3]의 소개

여행의 즐거움의 반 혹은 때때로 그 이상은 여행 후 경험에 대해 자랑하는 것이다. 종종 이 자랑에는 방대한 사진들이 동원되는데, 찍는 사람의 촬영 기술에 대한 진수를 보여주는 사진들도 있고, 따분하고 초점조차 맞지 않는 형편없는 사진들도 있다.

어떤 사진들이건 간에, 인터넷의 힘은 여러분에게 가족과 친구들보다 훨씬 더 큰 규모의 관객에게 이 사진들을 보여줄 수 있는 능력을 제공한다.

13.1.1 애플리케이션 목적

여행 사진들이 단조로운 것이 되지 않도록 하는 한 가지 방법은 '문맥(context)'을 부여하는 것이다. 그리고 이러한 문맥을 제공하기 위해 우리는 '여행기록 공유(Trip-o-matic)'이라 불리는 웹 애플리케이션을 제작할 것이다.

1 으깬 감자요리("mashed potato")를 빗댄 설명.

2 메시업이란 원래 힙합 용어로 여러 곡을 섞어서 리믹스하는 테크닉을 가리킨다.

3 원문은 "Trip-o-matic application"으로 여행 정보 보여주기 자동화의 의미를 담고 있다.

여기서 우리는 이렇게 생각할 수도 있다. "그냥 그 사진들을 플리커 계정에 넣어두면 어때?"

맞다. 그렇게 할 수도 있다. 하지만, 그것은 우리가 갈망하는 문맥을 얻기 위해서는 그다지 충분하지 않다. 물론, 우리는 "여기 오스틴의 사진들이 있다," "여기 오클라호마시의 사진들이 있다," "여기 토페카의 사진들이 있다," "여기 링컨의 사진들이 있다," 그리고 "오호, 봐! 그건 러시모어 산이잖아"와 같이 사진에 댓글을 추가함으로써, 어디에서 찍은 사진인지 방문자가 알 수 있도록 할 수 있다.

그러나 우리는 이것보다 훨씬 더 대단한 것을 할 것이다. 우리는 지도를 보여줄 것이다!

13.1.2 애플리케이션 개요 및 요구사항들

여행 기록 공유 애플리케이션은, 그 자체는 비교적 간단하지만 더 복잡한 매시업 애플리케이션을 위한 기반이 되기에 충분하다. 이 애플리케이션은 우리가 경험한 특별한 여행 정보를 가진 데이터 파일로부터 만들어진 하나의 페이지로 구성될 것이다. 어떤 여행 기록이 보여질지 몰라도 단일 페이지로 만들어질 것이기 때문에, 특정한 여행 정보가 페이지나 애플리케이션 코드에 하드코딩되면 안된다. 모든 여행 정보는 페이지가 읽게 될 데이터 파일로부터 생성하게 될 것이다.

아마 여러분은 JSP 또는 PHP와 같은 기술 기반의 서버사이드 템플릿 애플리케이션을 떠올릴지도 모른다. 그러나 우리의 애플리케이션은 한 가지만 제외하고 전적으로 클라이언트 측 코드로만 이루어질 것이다. 우리는 12.1.2항에서 개발한 크로스-서버 프록시 서블릿 서비스를 사용할 필요가 있다. 이 프록시는 우리가 Ajax 보안 샌드박스의 제약을 극복할 수 있도록 하며, 야후와 플리커 사진 서비스 사이트에 대한 크로스-서버 요청을 가능하게 만들어 준다.

이 애플리케이션에서 사용할 데이터 파일은 특정 여행에 대한 모든 정보를 포함한다. 이 파일에는 짧은 이름, 설명 그리고 심지어는 야후와 플리커 계정들에 대한 우리의 접근키 정보도 포함한다.

여행 정보에는 여행 경로상의 관심 지점(points of interest) 목록도 포함할 것이다. 또각 지점은 짧은 이름, 상세한 설명, 위치, 그리고 해당 관심 지점과 관계된 플리커 사진들을 가져오기 위한 정보를 포함한다.

애플리케이션은 여행 정보를 읽어, 여행 정보에 포함된 관심 지점들에 대한 짧은 이름

의 목록을 페이지에 출력할 것이다. 방문자가 이 목록 중에 하나를 클릭하면 해당 지점의 주변을 지도로 보여주도록 할 것이다. 지도를 클릭하면 해당 지점과 연관된 사진들의 썸네일을 보여주며, 썸네일을 클릭하면 전체 크기의 사진을 보여줄 것이다.

지속적인 관심을 유발하기 위한 방법으로 상호작용 기능 만한 것은 없다.

12장에서 본 짧은 예제들과는 달리, 사용성을 증가시키기 위한 기초적인 작업으로 애플리케이션 페이지에 적당한 양의 스타일링을 적용하려 한다. 그러나 이전의 예제들과 마찬가지로 이 애플리케이션도 UI 전문가들의 도움을 받거나, 실력 있는 디자이너의 도움을 받을 수 있다면 훨씬 더 좋을 것이다.

또 최소한의 기능을 가지도록 작성했기 때문에, 에러 체크가 충분이 이루어지지 않았음을 인정할 필요가 있다. 여러분이 이 예제를 활용하려 한다면, 이 예제에 대한 첫 번째 개선 작업은 에러 처리 부분을 보강하는 것이 되어야 할 것이다.

예제에서 볼 수 있듯이, 예제들은 '개념 검증(proof of concept)'[4] 차원에서 제공되는 것이며, 우리가 필요한 기술적인 내용을 얻는 데는 충분하다.

13.2 여행기록 공유 서비스의 데이터 파일

단 한 줄의 코드라도 작성하기 전에, 페이지에서 사용할 데이터 파일을 설계할 필요가 있다. 이 파일은 서버에 존재하며, 페이지가 로드될 때 Ajax 요청을 통해 가져오게 된다.

이때 데이터가 해석되고, 포함된 정보를 표현하는 클라이언트 측 자바스크립트 구조가 생성된다. 어떤 데이터 파일을 로드할지는 요청인자로 전달되는 URL로 식별하게 된다.

13.2.1 어떤 형식을 사용해야 할 것인가?

이전 장에서 보았듯이, Ajax 요청에 대한 응답은 일반 텍스트(plain text), HTML 조각, JSON 표기, 혹은 XML 문서의 형태 중 하나가 될 수 있다. 여행 데이터를 위한 데이터 형식으로 어떤 것이 가장 좋을까?

분명히 여행 데이터는 구조화된 정보의 집합이기 때문에, 일반 텍스트(plain text)와 HTML은 선택에서 제외 한다. 왜냐하면 전자는 데이터의 구조를 표현할 수 없고, 후자는

4 실제 서비스나 사용을 목적으로 하지 않은, 아이디어나 개념을 검증해 보는 차원의 구현. 약자로 "PoC"로 사용하기도 한다

데이터 형식이 아닌 표현을 위한 형식이기 때문이다. JSON 혹은 XML이 남았다.

JSON은 아주 매력적인 표기법이다. 왜냐하면 클라이언트 프로그램에서 해석하기가 무척 쉽기 때문이다. 이를테면 `eval()` 함수를 사용하여 간단하게 JSON 응답을 자바스크립트 구조체로 변환할 수 있다. 또 JSON은 프로그램 코드를 생성하기에도 좋다. 하지만, JSON이 우리가 작성할 여행 파일의 경우와 같이 커다란 데이터 집합에는 그다지 좋은 선택이 아닐 수 있다.

JSON은 간결하고 배열과 구조체를 나타내는 대괄호와 중괄호와 같은 간단한 구분 문자들을 사용한다. 중첩된 계층적 정보들로 구성되기 시작하면 (마치 여러분의 애완동물이 키보드 위를 걸어갈 때 표시되는 글자들처럼) JSON 데이터는 다소 읽기 어려워진다. 눈으로 해석하기 힘들 뿐만 아니라, 수정 혹은 추가할 때 의도하지 않은 에러를 포함하기도 더 쉬워진다.

반면, XML은 자바스크립트에서 해석하기 위해 추가적인 클라이언트 작업을 필요로 하지만, 손으로 코딩하기에는 더 잘 어울린다. XML의 장황한 마크업의 특성상 사람들이 데이터의 구조를 머리 속에서 미리 점검하도록 하기 때문이다. XML은 데이터를 더 쉽게 생성하고, 유지하고, 수정할 수 있도록 해준다.

자, 정리해보면 우리가 직면한 내용은 다음과 같다.

- JSON은 코드에서 해석하기 쉽지만, 손으로 작성하기에는 쉽지 않다.
- XML은 해석하기에는 더 복잡하지만, 손으로 작성하기에는 더 쉽다.

해석하는 코드는 한 번만 작성하지만 잠재적으로 많은 여행 데이터 파일을 작성할 것이므로, 우리의 작업을 쉽게 할 수 있도록 하기 위해 XML을 선택할 것이다. 왜냐하면 이렇게 하는 것이 우리가 더 많이 해야 할 작업을 간단하게 해주기 때문이다.

게을러진다는 뜻이 아니라, 스마트하게 작업을 진행한다는 뜻이다.

게다가, 만일 이 선택이 잘못된 것이라면, 나중에라도 이 선택을 쉽게 바꿀 수 있도록 작업을 진행할 것이다.

13.2.2 여행 데이터 형식

여행 XML 문서는 하나의 여행에 대한 정보를 포함한다. 우리가 각각의 그 여행에 대해 필요한 데이터는 다음과 같다.

- 여행의 제목
- 플리커 API 키(12.3절을 보라)
- 여행 사진들을 관리하는 계정에 대한 NSID(또는 12.3.1항을 보라)
- 관심 지점들을 위한 헤더로 사용될 텍스트
- 여행에 대한 자세한 설명

XML 문서의 루트 요소는 <trip> 요소가 될 것이며, 처음 네 개의 데이터들은 <trip> 요소의 속성으로써 기술될 것이다. 설명 데이터는 다소 길어질 수 있고, 특수문자 그리고 어쩌면 HTML 마크업도 담고 있을 수 있으므로, <trip> 요소의 자식 <description> 요소로 기술할 것이다. 이렇게 하면, <description> 요소의 텍스트는 CDATA 섹션 안에 담을 수 있다. 이렇게 해서 필요하다면 <description> 요소에는 HTML 마크업을 포함할 수 있으며, XML을 해석하는데 문제가 생기지 않을 것이다.

전형적인 <trip> 요소와 <description> 요소는 아래와 같을 것이다.

```
<trip title="The Trip Title"
      flickrNSID="97545223@N00"
      flickrKey="78eca5287f3f05397dfba77ef40c7df53"
      poiTitle="Where we went">
  <description>
    <![CDATA[
      This is a descriptive comment for the trip complete with
      <b>some HTML markup</b>.
    ]]>
  </description>
  ... <!-- 자식 노드가 반복된다. -->
</trip>
```

<trip> 요소는 <description> 요소 외에 다른 자식 요소 목록도 가진다. 각 요소들은 여행의 관심 지점을 정의한다. 각 관심 지점(Point of Interest)을 표현하는 <poi> 요소에 포함될 정보는 아래와 같다.

- 해당 지점에 대한(페이지에 표시될 목록에 사용할) 짧은 이름
- 해당 지점의 위도
- 해당 지점의 경도
- 해당 지점에 연관된 사진들에 대한 플리커 사진 세트 ID
- 해당 지점에 대한 설명 또는 주석

관심 지점에 대한 상세 설명을 제외한 모든 내용은 <poi> 요소에 대한 속성들로 제공된다. 관심 지점에 대한 상세 설명은 HTML 마크업을 포함할 수 있으므로, 상세 설명은 〈poi〉 요소의 텍스트 혹은 CDATA 섹션으로 기술된다.

이렇게 하면, 전형적인 〈poi〉 요소는 아래와 같은 모양이 될 것이다.

```
<poi name="Austin, TX"
      latitude="30.266748"
      longitude="-97.74176"
      photoSetId="1151001">
  <![CDATA[
    We started in Austin, TX on May 23, 2006. It was a
    <i>gorgeous</i> and sunny day. Little did we know what
    was in store for us...
  ]]>
</poi>
```

여행의 각 지점의 위도와 경도를 기록하기 위해 GPS를 사용하지 못하는 사람들을 위해, 12.1.3항에서 테스트한 야후 지오 코딩 API를 사용하여 위치 문자열을 좌표 값으로 변환할 수 있을 것이다.

다음은 여행 정보 공유 애플리케이션에서 쉽게 접근할 수 있도록, photoSetId 속성에 플리커 사진에 해당하는 값을 설정하는 방법을 살펴보도록 하자.

13.2.3 플리커 사진 세트 설정하기

매우 짧은 기간동안 플리커 사진 서비스가 대중적으로 크게 성장하게 된 이유 중 하나는 사용하기에 정말 쉽다는 것이다. 계정을 만들기 위해 등록하는 절차는 간단하고 매우 빠르다(만약 여러분이 이미 야후! 계정을 갖고 있다면, 그것을 사용할 수도 있다). 그리고 플리커의 웹 인터페이스를 이용해서 바로 사진들을 올리기 시작할 수 있다. 만일 여러분이 그 절차를 좀 더 쉽게 하고 싶다면 (한 번에 여러 장의 사진을 업로드를 쉽게 업로드 할 수 있는) 데스크탑 애플리케이션 도구를 다운로드하여 사용할 수 있다. 그것은 Mac OS X뿐만 아니라 윈도에서도 사용할 수 있다. 여러분의 사진 업로드를 더 쉽게 해주는 아이포토(iPhoto) 플러그인 또는 윈도 탐색기를 위한 플러그인도 있다. 심지어는 이메일을 통해서도 여러분의 사진들을 손쉽게 업로드할 수 있는 방법도 있다. 이런 방법을 통해 좀 더 손쉽게 플리커를 사용할 수 있다.

우리는 12.3절의 예제에서 대상 계정에 업로드된 모든 공개 사진들에 대한 정보를 얻을 수 있었다. 분명, 우리가 여기서 하고자 하는 것은 아니다. 우리는 대량의 사진 이미지들을 송수신하기보다는 우리 여행의 관심 지점 중 하나와 연관된 사진들에만 접근하고 싶다.

플리커는 사진 세트(photo sets)를 생성하고 여기에 사진들을 추가하여 사진들을 재구성할 수 있는 기능을 제공하고 있는데, 우리는 이 기능을 활용하려 한다. 사진세트에 사진을 추가하도록 함으로써, 다른 목적으로 추가된 사진들을 제외하고 특정한 목적의 사진들만 활용할 수 있다.

사진 세트을 생성하고 관리하는 것은 플리커의 다른 기능들과 마찬가지로 사용하기 쉽다. 간단히 플리커의 'Organize' 탭을 클릭하고 여기서 사진첩을 생성하고 관리할 수 있다. 여러분의 여행 사진들을 업로드하고 나서, 플리커의 'Organizer' 인터페이스를 이용해서 여행 데이터 파일의 각 관심 지점에 해당하는 사진 세트를 생성하라, 그러고 나서 해당 사진세트에 적절한 사진들을 추가하라.

우리 애플리케이션에서, 사진 세트의 정보를 얻기 위해 사용할 플리커 메서드는 사진 세트의 이름이 아닌 사진 세트의 ID를 필요로 한다. 이 ID는 플리커 웹사이트상에서는 보이지 않는다(적어도, 이 ID를 표시할 방법을 찾을 수 없을 것이다). 그러나 플리커 API는 사진 세트 ID 값을 포함하여, 사진 세트에 대한 정보를 얻을 수 있도록 해주는 `flickr.photosets.getList` 라는 메서드를 제공한다.

이것은 매번 얻어야 할 정보가 아니기 때문에 이 정보를 얻기 위해 코드를 작성할 필요는 없다. 단지 아래와 같은 URL로 플리커 서비스를 요청하라.

```
http://www.flickr.com/services/rest/?method=flickr.photosets.getList
&api_key=78eaa5287f3f0b37dfb47def40c7df13&user_id=95355920@N00
```

그리고 참조될 사진 세트의 ID에 대한 결과를 확인하자. 물론 이 URL에서 대상이 되는 플리커 계정의 API 키와 NSID를 반드시 여러분 것으로 바꾸어야 한다.

이제, 애플리케이션에 필요한 코드를 작성할 시간이 되었다.

13.3 TripomaticDigester 클래스

우리는 마음대로 사용할 수 있는 Prototype 툴킷을 가지고 있고, 이 책을 통해 진보적인 자바스크립트의 수 많은 예제들을 봐왔고, 우리의 플리커 사진 세트를 만들었다. 그리고

여행 데이터 파일을 위한 형식도 결정했다. 자 이제 본격적인 작업에 착수해 보자!

우리가 구현할 첫 번째 기능 중 하나는 우리가 앞에서 정의한 여행 데이터 파일을 해석하는 것이다. XML 문서를 획득하고, 해석하고, 페이지에 여행 데이터를 표현할 구조를 만들기 위해 필요한 모든 내용을 페이지의 스크립트에 포함하기보다는 그 임무를 담당할 자바 스크립트 객체를 생성할 것이다.

실제로, 우리는 이 기능을 여행 정보 공유 애플리케이션의 나머지 부분으로부터 완전히 분리되도록 할 것이다. 왜 이렇게 하는가?

애플리케이션 로직과 데이터 해석 코드를 분리하는 것은 조직화된 코드를 유지하기 위해서 뿐만 아니라, 애플리케이션의 나머지 코드에서 데이터를 읽는 임무와 처리하는 임무를 분리해, 임무를 완전히 추상화하는 데에도 도움이 된다. 또 향후 아무때나 데이터 파일 형식을 바꿀 수 있도록 허용할 것이다. 심지어, XML형식를 변경하려고 하거나, 또는 XML이 잘못된 선택이었고 JSON을 채택해야 한다고 할 때, 단지 이 TripomaticDigester 클래스만 영향을 받을 것이다. 이 정도의 추상화, 즉 이 예제에서 보여주는 '관심의 분리(separation of concerns)'[5]라는 개념의 구현은 다른 서버 측 언어들에서는 비교적 엄격하게 요구되는 것이며, 클라이언트 측 자바스크립트에서도 충분히 구현할 만한 가치가 있다.

우리의 해석 클래스를 TriopmaticDigester 라고 부를 것이며, `TriopmaticDigester.js` 파일에 정의될 것이다. 이제, 이 클래스 구현의 각 부분을 확인해 보자.

13.3.1 의존성 체크

알고 보면 결국 '자바스크립트 엔진에서 참조하려는 객체가 없다'는 의미를 가진 난해한 에러 메시지를 본 적이 있는가? 아마 여러분은 자바스크립트 코드에서 무엇이 잘못되었는지를 밝혀 내려고 머리를 쥐어 뜯으며 몇 시간을 허비한 적이 있을 것이다. 그러고 나서 결국 우리 코드가 의존하는 .js 파일을 포함하지 않았다는 사실 때문에 우리 자신을 자책했었다.

이 애플리케이션의 나머지 자바스크립트 코드뿐만 아니라, 우리의 해석기 자체도 Prototype 라이브러리에 상당히 의존하고 있다. 우리의 해석기 클래스를 임포트하기 전

5 다루어야 하는 문제가 복잡해 짐에 따라 관심에 따라 문제를 분리하여 해결하자는 것을 말한다. 정보은닉을 의미하며, 소프트웨어 공학의 아버지라 불리는 데이드 파르나스(David Parnas)가 제시하였다. MVC(Model-View-Controller)모델에서 모델과 뷰를 분리하는 것이나, EJB에서 세션 빈(session bean)과 엔터티 빈(entity bean)을 나눈 것도 "관심의 분리"라는 맥락이다.

에 Prototype 라이브러리가 로딩되지 않은 상황에서, 브라우저가 전형적인 난해한 에러 메시지 출력하도록 하기보다는, 명시적으로 확인하여 좀 더 명확한 메시지를 보여주도록 할 수 있다.

해석기 스크립트 파일의 처음 부분에, 다음 내용을 추가한다.

```
if (!Prototype) {
  throw new Error(
  "Prototype must be in scope to use Trip-O-Matic");
}
```

Prototype 라이브러리는 Prototype이라는 객체를 정의하고 있으므로, 이 객체가 존재하는지 확인한다. 만약 이것이 정의되지 않았다면 자바스크립트 콘솔(혹은 브라우저의 팝업 윈도우)에 나타날 자바스크립트 에러를 발생시켜 우리가 부주의했음을 알린다.

좀 더 까다롭게 확인하고자 한다면, Prototype 라이브러리는 Prototype 버전 속성을 가지고 있으므로, 이 버전 속성을 체크할 수도 있다.

이것은 다른 자바스크립트 라이브러리에 대한 의존들이 확인할 필요가 있을 때, 어디서든 활용할 수 있는 손쉬운 기법이다.

13.3.2 TripomaticDigester 생성자

TripomaticDigester 생성자에서 해야 할 작업은 여행 데이터 리소스의 URL을 얻어, 이 리소스에 의해 식별된 XML 문서를 해석하고, 나중에 해석된 여행 데이터를 사용하기 위해 저장하는 일이다. 그리고 이 절차가 완료되면 요청자(caller)에게 알린다. 이것이 결코 작은 일은 아니지만, 한 번에 하나씩 작업을 처리하도록 하고, 특정 구현 함수에게 작업을 위임함으로써 생각보다 어렵지 않게 구현할 수 있다.

자바스크립트 클래스를 생성하기 위해 Prototype 라이브러리를 사용하고 있기 때문에, 리스트 13.1에서 볼 수 있는 것처럼 TripomaticDigester 생성자는 클래스를 생성하고 prototype 속성에 initializer를 지정하는 내용으로 구성된다. 이 코드가 클래스 전체 코드는 아니다. 한 번에 한 부분씩 나누어 살펴보자.

리 스 트 13.1 TripomaticDigester 생성자 컴포넌트

```
TripomaticDigester = Class.create();

TripomaticDigester.prototype = {

  initialize: function(dataUrl,onLoadHandler) {
    this.dataUrl = dataUrl;
    this.onLoadHandler = onLoadHandler;
    new Ajax.Request(
      this.dataUrl,
      {
        onSuccess: this.onDigest.bind(this),
        onFailure: function() {
          throw new Error('failed to load ' + this.dataUrl);
        }
      }
    );
  },
```

❶ 초기화 인자를 저장

❷ Ajax를 통해 데이터 파일 컨텐츠를 가져옴

❸ onSuccess 핸들러를 인스턴스에 바인드

이 코드가 그다지 나쁘게 보이진 않지만, 솔직히 말해 힘든 작업을 모두 수행해 주는 것은 아니다. initialize() 메서드는 여행 데이터 리소스 URL과 여행 데이터 로딩이 완료될 때 호출될 핸들러를 인자로 받는다. 이 값들을 저장하고 난 다음❶, 생성자는 지정된 URL의 내용을 가져오기 위한 Ajax 요청을 날린다 ❷.

이 요청에 대한 onSuccess 이벤트 핸들러❸는 핸들러 메서드인 onDigest()에 대한 컨텍스트(context) 객체가 TripomaticDigester 인스턴스라는 것을 확실하게 하기 위해 Prototype bind() 메서드를 이용하여 지정했다는 점을 잘 살펴보아 두자.

만약 여러분이 컨텍스트(context) 객체 바인딩이 무엇에 관한 것인지 그리고 그것이 이 구아나와 무슨 관련이 있는지 복습할 필요가 있다면 3.1.2항을 다시 살펴보기 바란다.

실제 작업은 onDigest() 콜백 메서드가 호출되면서 시작된다.

13.3.3 여행 정보 해석하기

만약 생성자에 제공된 URL이 유효했고 그 밖에 잘못된 것이 아무것도 없다면, 서버로부터 XML 문서 수신이 완료될 때 onDigest() 메서드가 호출될 것이다. 이 메서드의 구현은 리스트 13.2에 있다.

리 스 트 13.2 TripomaticDigester.onDigest() 메서드

```
onDigest: function(request) {
  var xmlDoc = request.responseXML;
  var tripElement = xmlDoc.childNodes.item(0);
  if (tripElement.nodeName != 'trip')
     throw new Error( 'root element must be <trip>' );
  this.title = tripElement.getAttribute('title');
  this.flickrNSID = tripElement.getAttribute('flickrNSID');
  this.flickrKey = tripElement.getAttribute('flickrKey');
  this.poiTitle = tripElement.getAttribute('poiTitle');
  this.description = '';
  var self = this;
  var descriptionElements =
     tripElement.getElementsByTagName('description');
  $A(descriptionElements).each(function(descriptionNode) {
     self.description += self.collectText(descriptionNode);
  });
  this.points = new Array();
  var poiElements = tripElement.getElementsByTagName('poi');
  $A(poiElements).each(this.loadPoint.bind(this));
  this.onLoadHandler(this);
},
```

이 함수에서 수행되는 작업은 중요한데, 간단히 말해 XML을 해석하는 일이다. XML 문서는 (XMLHttpRequest의 인스턴스의) 요청 인자로부터 얻어진다. 그것의 첫 번째(그리고 단 하나) 자식이 얻어지고, 이 루트 요소가 약속된 <trip> 노드라는 것을 확인하기 위한 검사를 수행한다.

이 검사가 성공하면, 이 요소의 속성들을 얻어 해석기 객체의 인스턴스 변수들에 저장한다. 그 다음, <description> 자식 요소의 텍스트 내용들을 얻어 collectText() 메서드(곧 설명할 것이다)를 호출하여 저장한다. NodeList를 배열로 변환하기 위해 Prototype $A() 함수를 사용하고, 그런 다음 모든 '<description> 노드들'의 내용을 가져오기 위해 이 배열에 대해 each() 메서드를 호출한다는 점을 눈여겨 보아두자.

잠시만! 노드들이라고? 두 개 이상 된다는 말인가?

그렇다. 우리는 관대하게 사용자가 원한다면 여러 개의 요소들에 여행기록을 넣을 수 있도록 허용할 것이다. 오직 하나의 <description> 요소만 허용함으로써, 인색하게 제한할 수도 있다. 그러나 좀 친절하면 어떠한가?

다시 한 번 Prototype $A()와 each() 함수를 사용하여 <poi> 자식 요소들은 수집하고, 각 <poi> 요소의 loadPoint() 메서드를 호출한다(해석기 객체 인스턴스에 대한 참조를 바인딩한 후에).

loadPoint() 메서드는 <poi> 요소들을 해석한다. 다음 절에서 이 메서드에 대해 좀 더 살펴볼 것이다.

마지막으로, 생성자와 함께 등록된 클라이언트 핸들러가 호출되며, 이때 해석기의 인스턴스가 인자로 넘겨진다. 이렇게 함으로써, 핸들러가 전역 변수 또는 다른 외부 수단을 통해서 해석기를 참조하지 않아도 되도록 한다.

자 이제, 진짜 재미있는 데이터들을 살펴보자. 적절히 이름 지어진 관심 지점들로!

13.3.4 관심 지점 로드하기

리스트 13.2의 onDigest() 메서드에서 <trip> 요소의 각 <poi> 요소에 대해 loadPoint()라는 순환 메서드(iterator method)를 호출했다. 해석기 인스턴스에 바인드된 이 메서드는 전달된 요소의 정보를 수집하여, 여행의 관심 지점에 대한 정보를 담은 자바스크립트 객체를 생성한다. 이 함수의 구현은 리스트 13.3에 있다.

리 스 트 13.3 TripomaticDigester.loadPoint() 메서드

```
loadPoint: function(poiElement,index) {
     this.points.push({
        name: poiElement.getAttribute('name'),
        latitude: poiElement.getAttribute('latitude'),
        longitude: poiElement.getAttribute('longitude'),
        photoSetId: poiElement.getAttribute('photoSetId'),
        description: this.collectText(poiElement)
     });
},
```

넘겨진 요소의 속성들로부터 자바스크립트 객체를 생성하는 것은 쉬운 일이다. 각각의 값을 얻기 위해 각 요소의 getAttribute() DOM 메서드를 호출한다. 그리고 자바스크립트 객체에 대응하는 이름을 가진 속성을 생성한다. 새로 생성한 객체 인스턴스는 나중에 참조하기 위해 points 배열의 끝에 추가된다.

<trip> 요소의 자식 노드 <description>와 마찬가지로 <poi> 요소의 데이터는

collectText() 메서드를 통해 수집한다.

그다지 어렵지 않지 않은가?

13.3.5 요소 텍스트 수집하기

XML 문서에서 속성의 값을 설정하는 것은 쉬운 일이다. 이미 다른 해석기 클래스에서 생성한 메서드들에서 보았듯이, 어떤 특정한 요소들의 자식 요소들을 얻는 것도 약간 더 복잡할 뿐이다. 그러나 요소의 몸체로부터 텍스트 데이터를 모으는 것은 조금 더 고려할 점이 있다.

속성 또는 자식 요소와 달리, 요소의 몸체 내에 있는 모든 텍스트를 반환하는 DOM 메서드는 없다. 왜 그럴지 좀 더 생각해 보면, 어렵지 않게 이해할 수 있다. 문서 내에 존재할 수 있는 다음 XML 문서에 대해 생각해 보자.

```
<description>
  The quick young cub jumped over the lazy bear.
</description>
```

매우 단순해 보인다. 대부분의 XML파서는 〈description〉 요소의 몸체에서 단일한 텍스트를 반환할 수 있을 것이다. 하지만, 다음과 같은 경우는 어떨까?

```
<description>
  The quick young cub <!-- was he that quick? --> jumped over the lazy bear.
</description>
```

주석 노드가 추가되었다. 이것은 XML 파서가 텍스트를 (최소한) 주석 노드로 분리된 두 개의 텍스트 노드로 나눌 것이다. 아니면 아래 내용은 또 어떤가?

```
<description>
  <![CDATA[
    The <strong>quick</strong> young cub jumped over the
    <i>lazy</i> bear.
  ]]>
</description>
```

위 내용의 몸체 텍스트는 문서 내에 구문 에러를 초래할 수도 있는 문자들(위 경우에는, HTML 마크업)을 포함할 수 있도록 CDATA 섹션으로 구성되어 있다. CDATA는 그 자체가 하나의 노드이다.

이 이슈들을 어떻게 다룰지는 애플리케이션의 요구에 달려 있다. 예를 들어, 모든 텍스트와 CDATA 요소들(주석은 무시하고)을 수집하고 그것들을 노드의 내용으로 제공될 하나의 텍스트 블럭에 연결할 수 있다. 이 방법을 여행정보 상세 설명 부분과 관심 지점 요소에 적용할 것이다.

마지막으로 collectText() 메서드를 만든다. 메서드의 코드는 리스트 13.4에 있다.

리 스 트 13.4 TripomaticDigester.collectText() 메서드

```
collectText: function(element) {
  var text = '';
  $A(element.childNodes).each(
     function(child) {
        if ((child.nodeName == '#text') ||
              (child.nodeName == '#cdata-section')) {
           text += child.data;
        }
     }
  );
  return text.strip();
}
```

이 메서드는 하나의 인자를 전달받아 모든 자식들에 대해 각각 #text와 #cdata-section의 노드명으로 식별된 텍스트와CDATA노드들을 찾는다. 우리가 찾는 이 노드들의 내용을 텍스트로 합쳐서 문자열 변수에 저장하고, Prototype String.strip() 메서드를 사용하여 정리된 문자열을 반환한다.

이렇게 하여 독립형 해석기 클래스를 완성했다. 다시 이야기하지만, 이 클래스에서 데이터를 읽는 부분은 애플리케이션의 나머지 부분으로부터 분리되어 있다. 이것은 애플리케이션에 에러를 초래하지 않고도, 데이터 형식을 포함한 해석 절차의 세부 내용을 바꿀 수 있는 자유를 가져다 준다.

13.4 Tripomatic 애플리케이션 클래스

잠시 흐름을 벗어나 데이터를 해석하는 부분을 살펴보았다. 이제 애플리케이션 자체를 살펴보자. 애플리케이션이 될 페이지에 자바스크립트를 포함시켜 코딩할 수도 있지만, 이런 방식보다 좀 더 똑똑하게 작업하자. 우리가 이 책 전체를 통해 지지해온 객체지향 관점을

유지하면서 애플리케이션을 구현하도록 하기 위해 자바스크립트 클래스를 먼저 구축하려고 한다.

이 애플리케이션은 해야 할 작업이 많고 추적해야 할 것도 많다. 생성, 관리, 그리고 수정될 썸네일과 사진들뿐만 아니라 여행정보, 관심 지점들, 지도들도 있다. 그러나 한 번에 여러 가지 일을 처리하고, 객체지향 프로그래밍, 자바스크립트, 그리고 Prototype 라이브러리의 힘을 이용하여 우리가 이루고자 하는 바를 달성할 것이다.

이 애플리케이션이 수행할 내용은 다음과 같다.

- 컨텐트가 표시될 DOM 요소를 생성한다(다음 내용을 포함할 요소들을 생성한다)
  - 여행 제목
  - 여행 상세 설명
  - 클릭 가능한 관심 지점들
  - 관심 지점 목록 라벨을 위한 헤더
  - 클릭된 관심 지점을 보여 주는 지도
  - 관심 지점에서 찍힌 사진의 썸네일 이미지들
  - 클릭된 썸네일의 전체 크기 사진
- 아래 내용을 위한 이벤트 핸들러를 지정한다.
  - 관심 지점상의 클릭은 야후 지도를 이용하여 연관된 지도를 표시
  - 표시된 지도를 클릭하면 플리커 서비스를 이용하여 맵의 관심 지점과 관련된 사진 세트의 썸네일 이미지를 가져와서 표시
  - 썸네일 이미지를 클릭하면 다시 플리커 서비스를 이용하여, 해당 썸네일에 대한 전체 사이즈 사진을 표시

그리고 이 모든 것은 전역 변수 또는 하드코딩된 요소 ID를 사용하지 않고 객체지향 방식으로 개발되어야 한다. 자, Tripomatic 클래스와 이 클래스의 생성자를 구현하는 것으로 시작해 보자.

13.4.1 Tripomatic 클래스와 생성자

시작하기 위해 Tripomatic.js 라는 이름의 파일을 생성하고, TripomaticDigester 클래스에서 사용했던 의존성 체크 코드와 유사한 내용을 추가한다. 리스트13.5에 보인 것처럼,

Prototype의 클래스 생성방식을 활용하여 클래스의 뼈대를 만든다. 이 목록은 클래스의 완전한 정의와는 거리가 멀다. 온전하고 작동하는 애플리케이션 애플리케이션을 얻을 때까지 우리는 이 13.4절 전체에 걸쳐 코드를 추가할 것이다. 따라서, 이 13.4절의 리스트는 이미 보여준 코드를 반복하지 않는다. 새로 추가된 코드만 기술할 것이다.

리 스 트 13.5 Tripomatic클래스의 뼈대

```
if (!Prototype) {
  throw new Error(
     "Prototype must be in scope to use Trip-O-Matic");
}

if (!TripomaticDigester) {
  throw new Error(
     "TripomaticDigester must be in scope to use Trip-O-Matic");
}

Tripomatic = Class.create();

Tripomatic.prototype = {
/* instance methods will go here */
}
```

Prototype으로 정의한 클래스들이 흔히 그러하듯이, 실제 생성 코드는 initialize()라는 메서드 안에 위치하게 된다. 이 메서드에 대한 인자들은 여행 정보를 담은 데이터 파일에 대한 URL, 애플리케이션 컨텐트를 포함할 DOM 요소, 그리고 임의의 정보를 넘기기 위한 옵션 해시가 될 것이다. 이 메서드는 리스트 13.6에 보여진다.

리 스 트 13.6 Tripomatic.initialize() 메서드

```
initialize: function(dataUrl,container,options) {
  this.container = $(container);
  this.options = Object.extend(
     {
        enablePanAndZoom: false
     }, options
  );
  this.createContent();
  this.digester = new TripomaticDigester(
     dataUrl,
     this.onDataLoaded.bind(this));
  this.map = new YMap(this.mapContainerElement);
```

```
  if (this.options.enablePanAndZoom) {
     this.map.addPanControl();
     this.map.addZoomLong();
  }
},
```

이 메서드는 보기보다 단순하다. 왜냐하면 대부분의 작업을 다른 메서드에 위임하기 때문이다. 넘겨진 container는 인스턴스 변수에 기록되고, 넘겨진 options는 기본 옵션들과 합쳐진다. 이 클래스가 enablePanAndZoom이라는 단지 하나의 옵션을 정의하고 있다는 점을 살펴보자. 이 옵션은 pan과 zoom 컨트롤을 야후 지도에 추가할 것인지를 정하는 옵션이다.

단지 하나의 옵션만 갖는다면, 왜 귀찮게 오브젝트 해시를 사용할까? 왜 그냥 선택적인 세 번째 인자를 정의하지 않는가? 답은 확장성 때문이다. 이 정도 복잡도를 가진 애플리케이션의 경우, 앞으로 더 많은 옵션들을 추가할 필요가 생긴다. 만일 옵션 해시를 쓰지 않는다면, 나중에 추가적인 인자 사용을 위해 이 인자를 해시로 바꾸려한다면, 생성자의 시그니처(signature)[6]를 바꾸어야 할 것이다. 이렇게 하면 이 클래스를 사용하는 다른 모든 사람의 코드를 바꾸어야 할 것이다. 이런 요구 사항을 미리 예측함으로써, 생성자 시그니처를 바꾸지 않고도 향후 옵션들을 추가할 수 있도록 한 것이다. 그 덕분에 이 클래스를 사용하는 개발자는 불필요한 변경을 피하고, 보다 유연하게 이 클래스를 사용할 수 있다.

이 메서드는 createContent()라는 메서드를 호출한다. 이 메서드는 애플리케이션의 데이터 내용을 표시할 모든 요소를 생성할 것이다. 이 메서드에 대한 내용은 잠시 후 살펴볼 것이다.

켄텐트 요소들이 생성된 후에, 넘겨진 dataUrl을 사용하여 TripomaticDigester의 인스턴스를 생성한다. 그리고 이 인스턴스는 나중의 참조하기 위해 저장한다. onDataLoaded() 메서드는 해석기가 자신의 일을 끝마치면 호출된다. onDataLoaded()가 호출될 때 Tripomatic 인스턴스가 함수 컨텍스트가 되도록 하기 위해 어떻게 함수 컨텍스트(this)를 이 콜백에 바인드했는지 눈여겨 봐두자.

6 메서드 혹은 함수가 외부에 보여지는 모습. 시그니처는 메서드의 이름과 인자의 개수와 타입으로 구성된다.

마지막으로, 야후 지도는 createContent()에 의해 생성된 컨테이너 내에서 생성되고 초기화된다(이것이 우리가 바로 보게 내용이다).

이제, 애플리케이션은 완전히 초기화되고 사용자 상호작용에 대한 준비가 되었다. 하지만, createContent()와 onDataLoaded() 메서드에서 더 많은 작업들이 이루어진다. 그 메서드들이 어떤 작업을 수행하는지 살펴보자.

13.4.2 컨텐트 요소 생성하기

이제 우리는 "생성할까? 아니면 붙일까?"라는 수수께끼와 만났다. 우리는 이 두 가지 방식으로 우리 애플리케이션을 다룰 수 있다. 방법 중 하나는, 3장의 Button 클래스 예제와 같이 사용자가 HTML 요소를 정의하게 하는 것이다, 그것은 우리 클래스가 그것의 초기화의 부분으로 추가할 것이다.

이 전략은 HTML이 매우 간단하거나 혹은 너무 많은 변화 요인을 포함하고 있어 우리가 미리 예측하기 힘든 경우에 적합하다. 그러나 다소 복잡하지만 예측 가능한 요소 계층 구조가 필요하다면, 스크립트가 요소들을 동적으로 생성하게 하는 것이 때대로 더 낫다.

이 후자의 전략이 우리의 '여행 기록 공유 애플리케이션'을 만드는데 사용할 방법이며, 생성자로부터 호출할 createContet() 메서드에서 수행하게 된다.

이 메서드는 DOM 계층 구조를 생성하는 역할을 수행한다. DOM 계층구조는 사용자가 우리에게 넘겨준 컨테이너 요소 내에 내재되며, 다음에 있는 HTML 조각과 같다.

```
<h1></h1>
<h2></h2>
<div class="tripomaticPoiContainer">
  <h3></h3>
  <ul></ul>
</div>
<div class="tripomaticMapContainer"></div>
<div class="tripomaticPoiDescription"></div>
<div class="tripomaticThumbsContainer"></div>
<div class="tripomaticPhotoContainer"></div>
```

우리의 여행 데이터 컨텐트는 이 요소들 안에 삽입될 것이다.

<h1>과 <h2> 요소들은 여행 제목과 설명을 위해 각각 사용되고, <h3>과 <ul> 요

소들을 포함하는 요소는 관심 지점들의 목록을 표시하게 된다. <trip> 요소의 poiTitle 속성에 의해 제공된 문자열은 <h3> 요소에 위치하며, <ul> 요소에는 관심 지점 항목들이 자리잡게 될 것이다.

각각의 요소에는 고유한 CSS 클래스(모든 CSS 클래스 명에는 사용자의 CSS 클래스 명과 충돌하지 않도록 tripomatic이 앞에 붙는) 이름이 지정되는데, 이 클래스는 어떤 요소가 놓여져 사용되는지를 식별하는데 도움을 줄 뿐만 아니라, 생성된 모든 요소를 사용자가 꾸밀 수 있도록 해준다.

사용자 입장에서 요소들을 생성할 때 이것은 중요한 점이다. 우리는 사용자가 CSS 셀렉터(CSS selector)를 통해서 각 요소들을 가리킬 수 있도록 해야 한다. 선택될 수 있는 요소만 사용자에 의해 꾸며질 수 있다. 가 아닌 요소들은 CSS 클래스 명을 필요로 하지 않는다. 왜냐하면 생성시에 요소 이름으로 CSS 셀렉터에 의해 고유하게 식별될 수 있기 때문이다.

코드 내에 문자열들을 하드코딩하지 않는 것이 좋은 것과 마찬가지로, 우리는 클래스 명을 나타내는 문자열들을 코드에서 꺼내어 클래스-레벨 참조에 넣고 싶다. 3장에서 논의했듯이, 클래스의 프로퍼티로 지정함으로써 클래스-레벨의 '상수'를 흉내낼 수 있다. 물론, 우리는 이들이 실제 상수가 아니라는 것을 안다. 하지만, 우리는 이것들을 마치 상수인 것처럼 다룰 것이다. 리스트 13.7에 볼 수 있는 것처럼, 이 할당문들은 Tripomatic.js 파일의 클래스와 prototype 선언 사이에 위치하게 된다.

리 스 트 13.7 클래스의 상수들

```
Tripomatic = Class.create();

Tripomatic.CLASS_POI_CONTAINER = 'tripomaticPoiContainer';
Tripomatic.CLASS_MAP_CONTAINER = 'tripomaticMapContainer';
Tripomatic.CLASS_POI_DESCRIPTION = 'tripomaticPoiDescription';
Tripomatic.CLASS_THUMBS_CONTAINER = 'tripomaticThumbsContainer';
Tripomatic.CLASS_PHOTO_CONTAINER = 'tripomaticPhotoContainer';

Tripomatic.prototype = {
```

이렇게 해서, 리스트 13.8의 DOM 요소를 생성하는 메서드를 살펴볼 준비가 되었다.

리 스 트 13.8 Tripomatic.createContent() 메서드

```
createContent: function() {
  this.container.innerHTML = '';
  this.tripTitleElement = document.createElement('h1');
  this.tripDescriptionElement = document.createElement('h2');
  this.poiContainerElement = document.createElement('div');
  this.poiTitleElement = document.createElement('h3');
  this.poiListElement = document.createElement('ul');
  this.mapContainerElement = document.createElement('div');
  this.poiDescriptionElement = document.createElement('div');
  this.thumbsContainerElement = document.createElement('div');
  this.photoContainerElement = document.createElement('div');
  this.container.appendChild(this.tripTitleElement);
  this.container.appendChild(this.tripDescriptionElement);
  this.container.appendChild(this.poiContainerElement);
  this.container.appendChild(this.mapContainerElement);
  this.container.appendChild(this.poiDescriptionElement);
  this.container.appendChild(this.thumbsContainerElement);
  this.container.appendChild(this.photoContainerElement);
  this.poiContainerElement.appendChild(this.poiTitleElement);
  this.poiContainerElement.appendChild(this.poiListElement);
  this.poiContainerElement.className =
    Tripomatic.CLASS_POI_CONTAINER;
  this.mapContainerElement.className =
    Tripomatic.CLASS_MAP_CONTAINER;
  this.poiDescriptionElement.className =
    Tripomatic.CLASS_POI_DESCRIPTION;
  this.thumbsContainerElement.className =
    Tripomatic.CLASS_THUMBS_CONTAINER;
  this.photoContainerElement.className =
    Tripomatic.CLASS_PHOTO_CONTAINER;
},
```

Tripomatic.createConent() 메서드는 약간 길지만 꽤 직관적이다. 첫째로, 페이지 저작자에 의해 생성자로 넘겨진 컨테이너가 초기화된다. 그러고 나서 애플리케이션에 필요한 다양한 요소들이 생성되고 인스턴스의 프로퍼티로 저장된다.

그 다음, 이 메서드는 우리가 이 13.4절의 앞부분에서 HTML로 기술한 구조에 이 요소들을 결합시킨다. 페이지 제작자 컨테이너의 하위 요소로 말이다. 마지막으로, 생성된 요소들에게 CSS 클래스명들이 지정된다.

뭐, 그리 나쁘지는 않았다. 그러나 이것은 실제 여행 데이터들이 없는 빈 요소 세트를 생성한다. 어떻게 데이터를 가져와 채우는지 살펴보자.

13.4.3 여행 데이터 채우기

이전에 살펴본 initialize() 메서드에서 수행한 단계 중 하나는 TripomaticDigester의 인스턴스를 생성하는 것이었다. 우리는 이 메서드에서 여행 데이터 파일과 그것이 여행 데이터 파일의 해석을 완료했을 때 호출될 onDataLoaded() 메서드에 대한 참조를 전달해 주었다.

우리는 컨텐츠 요소를 생성한 후에 해석기 인스턴스를 생성했다. 이렇게 함으로써 onDataLoaded()가 호출될 때 우리가 DOM 계층구조가 이미 조립되었다고 가정할 수 있다. 이 메서드에 대한 코드를 살펴보자.(리스트 13.9)

리 스 트 13.9 Tripomatic.onDataLoaded() 메서드

```
onDataLoaded: function(digester) {
  this.tripTitleElement.innerHTML = digester.title;
  this.tripDescriptionElement.innerHTML = digester.description;
  this.poiTitleElement.innerHTML = digester.poiTitle;
  digester.points.each(this.makePointOfInterest.bind(this));
  this.showPoint(digester.points[0]);
},
```

해석기는 유용하게도 자신에 대한 참조를 Tripomatic.onDataLoaded() 메서드에 넘겨주며, 우리는 그 참조를 사용해서 해석된 데이터를 얻는다. 그러나 비록 해석기 자체가 그렇게 도움이 되지 않았더라도 별로 상관없다. 이 메서드가 현재 Tripomatic 인스턴스에 바인드되어 있어서, 인스턴스 프로퍼티에 저장된 해석기 참조를 통해 접근할 수 있도록 해준다. 객체 지향은 참으로 유용하다!

여행 제목과 상세 설명을 표시하는 것은 (createElement() 메서드에서 설정한) 적절한 컨텐트 요소의 innerHTML 속성에 적절한 값을 설정하기만 하면 되는 비교적 간단한 일이다. Prototype의 each() 메서드를 사용하여 makePointOfInterest() 메서드를 반복 함수로 지정함으로써, 모든 지점에 대해 이 함수를 수행하도록 한다.

페이지 방문자에게 제어를 넘기기 전에(기억하라, 이 모든 것은 페이지 로드의 결과로써 발생하는 것이다), 관심 지점 리스트의 첫 번째 항목에 대한 참조와 함께 showPoint()라는 메서드를 호출한다. 이 함수는 목록의 첫 번째 지점에 대한 정보가 표시되도록 한다. 그리고 특정 지점명을 사용자가 클릭하면 동일한 함수가 사용된다. 우리는 13.4.4항에서 이것을 구현하게 된다.

해석기가 로드한 각 지점별로, 반복함수로 makePointOfIneterest() 메서드가 호출되도록 할 것이다. 이 메서드는 방문자가 그 지점에 대한 지도를 로드하기 위해, 클릭할 수 있는 활성화된 지점 목록 항목을 생성하는 임무를 맡는다. 이 메서드의 구현은 리스트 13.10에 있다.

리 스 트 13.10 Tripomatic.makePointOfInterest() 메서드

```
makePointOfInterest: function(point) {
  var pointItem = document.createElement('li');
  pointItem.appendChild(document.createTextNode(point.name));
  pointItem.onclick = this.onPoint.bindAsEventListener(this);
  pointItem.point = point;
  this.poiListElement.appendChild(pointItem);
},
```

DOM 조작 API를 사용하는 Tripomatic.makePointOfInterest() 메서드는 지점 항목을 담는 <li> 요소를 생성한다. click 이벤트를 위한 이벤트 핸들러로 onPoint() 메서드를 설정하고, 현재 인스턴스에 대한 이벤트 리스너로서 바인드된다. 이벤트 핸들러에서 이 아이템과 연결된 지점 정보를 쉽게 얻을 수 있도록, <li> 요소에 point라는 속성에 할당한다. 생성된 <li> 요소는 앞의 초기화시에 생성한 부모인 <ul> 요소에 추가된다.

마지막 관심 지점에 대한 이 함수의 호출을 끝으로, 페이지가 로드될 때 발생하는 꽤 긴 이벤트 체인이 완료된다.

www.manning.com/crane2(또는 wikibook.co.kr)에 있는 이 장에 대한 다운로드용 소스코드와 예제 XML 데이터 파일을 함께 수행하면, 그림 13.1에 보이는 것과 같은 결과가 방문자에게 표시된다.

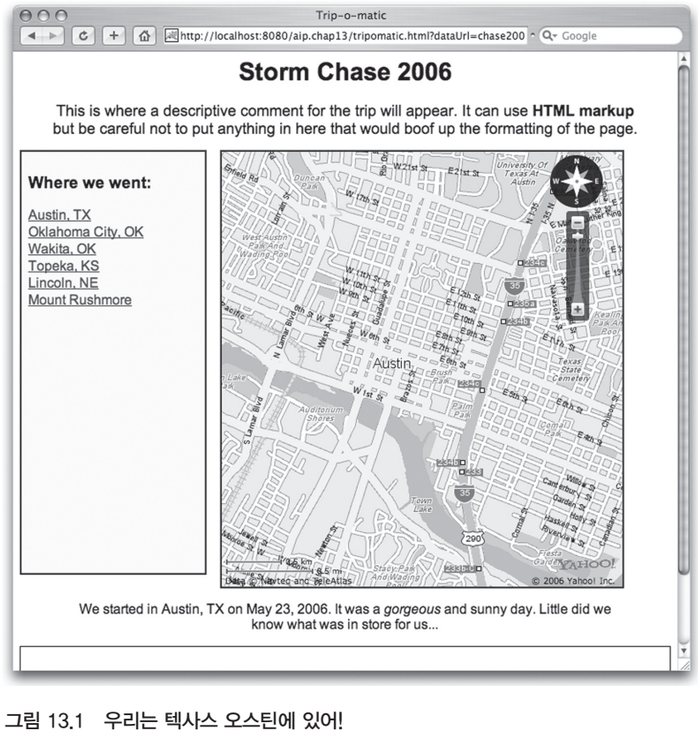

그림 13.1 우리는 텍사스 오스틴에 있어!

제목, 상세 설명, 그리고 관심 지점이 표시되고 첫 번째 지점에 대한 지도가 로드된다는 점을 기억해 두자. 페이지 하단의 빈 공간은 썸네일들이 로드되며, 이 썸네일을 클릭하면 전체 크기의 사진으로 확장되는 곳이다. 그런데, 어떻게 여기에 지도가 표시될 수 있을까?

13.4.4 지도 보여주기

각 관심 지점 항목들에 이벤트를 핸들러를 설정하였기 때문에, 방문자가 지점 이름 중 하나를 클릭하면 `onPoint()` 메서드가 실행된다. 이 메서드는 이벤트 핸들러로 동작하며, 인자로서의 Event 인스턴스와 함께 호출되고, 이 인자를 통해 우리는 어떤 요소가 클릭되었는지를 알아낼 수 있다. 이전에 우리는 이 메서드를 쉽게 접근하기 위해 `point`라는 속성에 요소에 대한 지점 정보를 저장했었다. 이 이벤트 핸들러 메서드에 대한 코드는 리스트 13.11에 있다.

리 스 트 13.11 Tripomatic.onPoint() 이벤트 핸들러 메서드

```
onPoint: function(event) {
  this.showPoint(Event.element(event).point);
},
```

이 이벤트 핸들러가 수행하는 동작은 관심 지점 정보를 이벤트 대상 요소에 설정하고, 이 정보를 `showPoint()` 메서드로 전달하는 것이다. `showPoint()` 메서드는 관심 지점에 연결된 지도를 보여준다. 초기에 로드되는 것과 마찬가지로 사용자가 관심 항목을 클릭할 때마다 호출된다. 상세한 구현은 리스트 13.12에 있다.

리 스 트 13.12 Tripomatic.showPoint() 메서드

```
showPoint: function(point) {
  this.currentPoint = point;
  var geoPoint = new YGeoPoint(point.latitude,
                               point.longitude);
  this.map.drawZoomAndCenter(geoPoint,4);
  this.mapContainerElement.onclick =
     this.showThumbnails.bindAsEventListener(this);
  this.mapContainerElement.point = point;
  this.poiDescriptionElement.innerHTML = point.description;
  this.thumbsContainerElement.innerHTML = '';
  this.photoContainerElement.innerHTML = '';
},
```

이 메서드에 넘겨진 지점을 현재 관심 위치(current point)로 기록하고, YGeoPoing 인스턴스를 생성하고, 12.1.1절에서 살펴본 예제와 같이 관심 지점을 사용하여 해당 위치가 중앙에 오도록 중앙에 YMap을 표시한다.

그런 다음 `showThumbnails()` 메서드를 지도 요소에 대한 click 이벤트 핸들러로 설정한다.

지점의 상세 설명을 표시하고, 이전에 표시된 지점들의 사진들을 제거하기 위해 썸네일과 사진 컨테이너를 초기화한다.

이제 우리는 가만히 앉아서 방문자가 지도에 클릭하는 것을 끈기있게 기다리는 일만 남았다.

13.4.5 썸네일 로드하기

만약 방문자가 지도를 클릭해서 뭔가 흥미로운 일이 일어난다는 것을 발견한다면–기억하라, 앞서 우리 애플리케이션이 사용성의 본보기는 아니라고 이야기했다 – 사용자들은 그들의 직관력에 대한 보상을 받을 만하다. 이제 우리는 방문자가 지도를 클릭했을 때 사진 세트 ID를 가지고, 현재 표시되고 있는 지도의 관심 지점과 함께 연관된 플리커 계정에 저장된 썸네일에 대한 정보를 가져오려 한다.

우리는 예전에 플리커 요청들을 만들어 봤다–12.3절을 확인하자. 다른 플리커 메서드를 사용하겠지만 작성하고자 하는 코드는 이미 독자들에게 친숙할 것이다.

각 관심 지점과 일치하도록 플리커 사진 세트를 만들었고, 이 세트들의 ID를 여행 데이터 파일에 기록했다는 것을 기억하라. 그렇기 때문에, 공개 사진을 모두 가져오기 위해서 `flickr.people.getPublicPhotos` 플리커 메서드를 사용하지 않고, 우리가 한정한 사진 세트만 조회하기 위해 `flickr.photosets.getPhotos` 메서드를 사용할 것이다. 그림 13.2는 지도를 클릭한 후의 지도와 연결된 썸네일의 모습을 보여주고 있다.

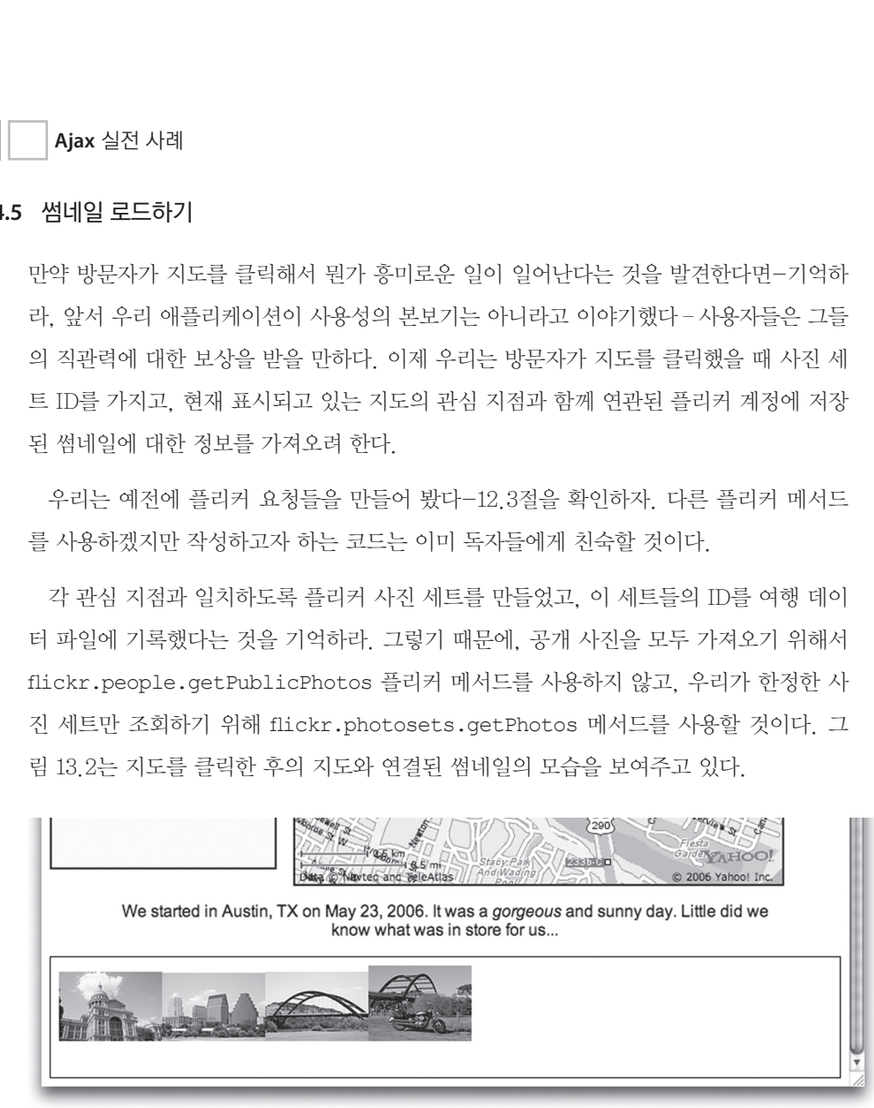

그림 13.2 오스틴에서 찍은 사진들의 썸네일들

클릭에 대한 응답으로 이 액션을 수행하는 이벤트 핸들러는 리스트 13.13에 있다.

리 스 트 13.13 Tripomatic.showThumbnails() 메서드

```
showThumbnails: function(event) {
  new Ajax.Request(
     '/aip.chap13/proxy',
     {
        onSuccess: this.onPhotosetList.bind(this),
        method: 'get',
        parameters: {
           '.serviceUrl.':
              'http://www.flickr.com/services/rest/',
           api_key: this.digester.flickrKey,
           method: 'flickr.photosets.getPhotos',
           photoset_id: this.currentPoint.photoSetId
        }
     }
  );
},
```


다른 Ajax 요청들과 마찬가지로, 이 메서드는 서버사이드 프록시 에이전트 방식으로 플리커 서비스에 연결한다. 12.1.2 항에서 이 프록시 에이전트에 대한 필요성뿐만 아니라 동작을 테스트 했었다.

편리하게, 그리고 의도적으로 `currentPoint`라는 인스턴스 프로퍼티에 현재 로드된 관심지점에 대한 정보를 저장했다. 이 핸들러는 전역변수나 지점 배열에의 인덱스가 없어도 `currentPoint` 속성을 통해 현재 지점의 `photoSetId` 프로퍼티를 참조할 수 있다.

이 요청에 대한 `onSuccess` 이벤트 핸들러(리스트 3.14)로 지정된 `onPhotosetList()` 메서드는 플리커 서비스가 응답을 반환하는 XML 문서를 해석하는 역할을 수행한다. 이 메서드의 구현을 살펴보자.

리 스 트 13.14 수신된 사진 세트 문서 처리하기

```
onPhotosetList: function(xhr) {
  var doc = xhr.responseXML;
  var status =
     doc.getElementsByTagName('rsp')[0].getAttribute('stat');
  if (status == 'ok') {
     this.thumbsContainerElement.innerHTML = '';
     var photos = doc.getElementsByTagName('photo');
     $A(photos).each(this.makeThumbnail.bind(this));
  } else {
     throw new Error('getPhotos request failed');
  }
},
```

우리가 12.3에서 사용했던 것과 다른 플리커 메서드를 사용했음에도 불구하고, 플리커 서

비스는 동일한 형식의 XML 문서를 반환한다. 그러므로 이 문서를 해석하는 방식 역시 동일하다.

첫째로 응답 문서를 가져온다. 그리고 그 서비스가 성공적으로 응답했는지 확인한다. 만약 모든 것이 잘 되었다면, 이전의 모든 썸네일을 초기화 하고, 문서 내의 모든 <photo> 요소의 리스트를 가져온다. 그리고 각각의 그 요소 노드에 대한 반복 함수로 makeThumbnail() 메서드를 호출한다.리스트 13.15의 makeThumbnail() 메서드는 썸네일 이미지를 생성하기 위한 대부분의 복잡한 작업이 이루어지는 곳이다.(실패 하면, 전체 응답 텍스트로부터 생성된 Error 인스턴스를 던진다는 것을 일러 둔다. 실제로, 여러분은 응답 문서에서 단지 에러 메시지만 추출하고 싶을 수도 있다. 하지만, 이 글에서는 계속 코드의 간결함을 유지하기 위해 이렇게 하지 않는다. 연습 삼아, 여러분이 이 코드에 더 나은 에러 검사와 복구 기능을 추가하여 확장하는 것이 어떨까?)

리 스 트 13.15 Tripomatic.makeThumbnail() 메서드

```
makeThumbnail: function(photo,index) {
  var baseUrl = 'http://static.flickr.com/' +
                photo.getAttribute('server') + '/' +
                photo.getAttribute('id') + '_' +
                photo.getAttribute('secret');
  var thumbUrl = baseUrl + '_t.jpg';                    ❶ 사진 URL 생성
  var photoUrl = baseUrl + '.jpg';
  var thumb = document.createElement('img');
  thumb.src = thumbUrl;
  thumb.style.cursor = 'pointer';
  thumb.onclick = this.showPhoto.bindAsEventListener(this);
  thumb.photoUrl = photoUrl;                            ❷ 나중에 참조하기 위해
  this.thumbsContainerElement.appendChild(thumb);         URL을 저장
},
```

살펴 본 바와 같이, 어려운 작업이라고 이야기했던 것들도 사실 그렇게 복잡하지는 않다. 단지 우리가 12.3.2항에서 했던 것처럼, 플리커 사이트에 호스팅된 썸네일 이미지를 참조하는 URL을 만들고, 그 URL을 참조하는 <img> HTML 요소를 생성한다.

우리가 썸네일 이미지의 URL을 생성하고 있는 동안, 썸네일이 클릭되었을 때 보여주고자 하는 전체 크기의 사진에 대한 URL도 생성한다. 그리고 그것을 썸네일에 대한 <img> 요소의 속성으로 저장한다.

그 후에 <img> 요소에 대한 onclick 이벤트 핸들러로 설정한 showPhoto() 메서드에서, 우리는 전체 크기 사진을 위한 <img> 요소를 생성하면서, 저장된 URL을 이용할 것이다.

13.4.6 사진 보여 주기

여기까지 오기 위해 우리는 상당히 많은 작업을 한 것 같다. 마지막으로 이제 사진들을 보여줄 준비가 되었다. 페이지 하단에 표시된 굉장한 사진들이 모습이 그림 13.3에 있다.

리스트 13.15에 있는 makeThumbnail() 메서드에서 각 썸네일 이미지 요소에 대한 click 이벤트 핸들러로 showPhoto() 라는 메서드를 지정했다. 이 메서드(리스트 13.16)는 우리가 12.3.2항에서 설정한 대응 함수와 거의 유사하다.

리 스 트 13.16 Tripomatic.showPhoto() 메서드

```
showPhoto: function(event) {
  if (this.photoElement == null) {
    this.photoElement = document.createElement('img');
    this.photoContainerElement.appendChild(this.photoElement);
  }
  this.photoElement.src = Event.element(event).photoUrl;
}
```

그림 13.3 텍사스 오스틴의 스카이라인

이 함수는 썸네일 이미지 요소에 대한 click 이벤트 핸들러이기 때문에, 이 함수가 호출될 때 그 <img> 요소는 이벤트의 대상 요소가 된다는 것을 기억하자. 현명하게도 썸네일에 대한 클릭의 결과로써 보이게 될 사진의 URL을 지정하는 속성을 그 요소에 미리 로드했기 때문에, 이 함수가 수행하는 작업은 꽤 간단하다. 사진들을 보여주는 <img> 요소가 이미 생성되었는지를 확인하고, 생성되어 있지 않다면 생성한다. 그런 다음 표시할 사진을 src 속성에 지정한다.

이 시점에서 여러분은 왜 createContext() 메서드에서 전체 사이즈의 사진에 대한 <img> 요소를 생성하지 않았는지 궁금해 할 것이다. 왜 사진을 바로 가져오지 않는 걸까?

우리는 createContext() 메서드가 실행되는 시점에서 표시하고자 하는 사진의 URL을 가지고 있지 않았다. 사실, 우리는 페이지 방문자가 지도를 클릭한 다음, 그 클릭의 결과로써 생성된 썸네일을 얻고, 이것을 클릭한 후에야 해당 사진의 URL을 얻는다. 만약 우리가 src 속성 없이, 또는 빈 것으로 <img> 요소를 생성하려고 했다면, 어떤 브라우저들은 유효한 src 속성 값이 지정될 때까지 '깨진 이미지(broken image)' 아이콘을 표시할 것이다. 이 것을 페이지에 표시하고 싶은 사람은 없을 것이다.

이렇게 해서 우리의 애플리케이션 클래스 제작을 완료하였다. 자 이제 이것을 HTML 페이지에 넣어 보자.

13.5 여행 기록 공유 애플리케이션 페이지

지금쯤 여러분은 아마도 간절히 바랄 것이다. "제발 우리 페이지 만들기를 시작하죠". 여러분의 소망은 이루어질 것이다. 실제 우리는 애플리케이션 자체를 위한 어떤 HTML 작성도 없이 많은 작업을 해왔다. 그 모든 힘든 작업에 대해 분명 보상이 있을 것이다. 또한 분명 그것을 보게 될 것이다. 이미 모든 필요한 기반 코드를 작성하는데 많은 시간을 할애했다. 이제 우리가 해야 할 일은 페이지에 HTML, CSS, 그리고 애플리케이션 클래스 인스턴스를 생성하는 자바스크립트 코드를 작성하는 것뿐이다.

13.5.1 trip-o-matic.html 문서

애플리케이션을 위한 HTML 페이지를 구축해 보자. 페이지를 위한 온전한 코드는 리스트 13.7절에 있다.

리 스 트 13.17 trip-o-matic.html 페이지

```
<html>
  <head>
    <title>Trip-o-matic</title>
    <link rel="stylesheet" href="styles.css"/>
    <script type="text/javascript" src="prototype-1.5.1.js">
    </script>
    <script type="text/javascript" src=
       "http://api.maps.yahoo.com/ajaxymap?v=2.0&appid=YahooDemo">
    </script>
    <script type="text/javascript" src="TripomaticDigester.js">
    </script>
    <script type="text/javascript" src="Tripomatic.js">
    </script>
    <script type="text/javascript">
       window.onload = function() {
          var dataUrl = document.URL.toQueryParams().dataUrl;
          if (dataUrl == null)
             throw new Error('the dataUrl parameter must be set');
          new Tripomatic(
             dataUrl,
             'tripomatic',
             {
                enablePanAndZoom: true
             });
       };
    </script>
  </head>

  <body>
    <div id="tripomatic"></div>
  </body>

</html>
```

❶ 스타일, 라이브러리, 클래스 임포트

❷ 애플리케이션의 인스턴스 생성

❸ 애플리케이션 요소들을 위한 컨테이너 생성

<head> 요소❶는 페이지에 기본 스타일을 적용하기 위해 사용할 CSS 스타일 시트를 가져오는 내용을 포함하고 있다. 또한 우리가 필요로 하는 다양한 외부 자바스크립트 파일, 이를 테면 Prototype 라이브러리, 야후 지도 라이브러리, 해석기 클래스, 그리고 Tripomatic 클래스 자신 같은 파일을 가져온다.

또한 페이지에 대한 onload 이벤트 핸들러를 정의하는 스크립트 요소도 포함한다.

이 핸들러 함수 안에서, Prototype의 toQueryParams() 문자열 메서드를 사용하여 dataUrl를 요청 인자로 넘겨지는 데이터 URL을 얻는다. 만일 이런 인자가 없다면 예외를 발생시킨다.

그런 다음 데이터 URL, 애플리케이션 요소들이 생성되는 컨테이너에 대한 참조❸, 그리고 옵션 해시 오브젝트를 지정해서 Tripomatic 클래스의 인스턴스를 생성한다❷.

페이지의 <body> 안에서, 애플리케이션이 요소들을 생성할 빈 요소를 생성한다.

그게 전부다. 그외 모든 것은 우리가 이미 작성한 코드들에 의해 처리된다. 페이지 작성자에게 얼마나 쉬운 일인가?

드디어, 우리의 애플리케이션 페이지가 완성되었다! 그러나 축하하러 가기 전에, 다음 13.5.2항에서 간단하게 우리 페이지의 헤더에 링크된 style.css 파일의 내용에 대해 설명한다. 페이지 레이아웃이 어떻게 만들어졌는지 궁금하다면 살펴보기 바란다.

13.5.2 스타일과 함께 살펴보기.

우리가 페이지의 모양을 내기 위해 사용하는 sytle.css 파일은 상상의 나래를 펴봐도 별로 대단한 것은 아니다. 그것의 목적은 페이지의 핵심 기능에 대한 우리의 논의를 복잡하게 하지 않으면서도 최소한의 사용성을 부여하기 위해 만들어졌다.

좋은 스타일이나 사용성이 있는 요소에 초점을 맞추는 것이 아니라, 매시업 애플리케이션을 생성하는 기술적인 메커니즘을 논의하는 것이 이 예제의 목적이었다. 훨씬 더 나은 스타일과 더 사용성 있는 애플리케이션에 기반을 제공하기 위해 애플리케이션의 HTML 뿐만 아니라 style.css 파일을 최대한 사용하라.

이 스타일 시트에서 한 가지 중요한 것은 생성하는 DOM 요소들을 적절하게 설계했다는 것이다. 페이지 작성자는 요소들이 페이지상에 어떻게 나타나고 펼쳐질지에 대한 다양한 제어를 부여할 수 있다.

style.css 파일은 리스트 13.18절에 있다.

리 스 트 13.18 | 기본 스타일 시트

```
body {
  font-family: Arial,Helvetica,sans-serif;
  padding: 8px;
  margin: 0px;
}

#tripomatic h1,h2 {
  text-align: center;
}
#tripomatic h1 {
  font-size: 1.8em;
}

#tripomatic h2 {
  font-size: 1.1em;
  font-weight: normal;
  padding: 0px 32px;
}

.tripomaticPoiContainer {
  float: left;
  border: 2px ridge maroon;
  background-color: #ffffcc;
  padding: 8px;
  width: 200px;
  height: 480px;
  overflow: auto;
}

.tripomaticPoiContainer li {
  cursor: pointer;
  list-style-type: none;
  margin-bottom: 2px;
  color: maroon;
  text-decoration: underline;
}

.tripomaticPoiContainer ul {
  margin: 0px;
  padding: 0px;
}

.tripomaticMapContainer {
  float: left;
  width: 440px;
  height: 480px;
  border: 2px ridge maroon;
  padding: 8px;
  margin-left: 16px;
```

```
  cursor: pointer;
}

.tripomaticPoiDescription {
  clear: both;
  margin: 0px 32px;
  text-align: center;
  padding: 16px 32px;
}

.tripomaticThumbsContainer {
  clear: both;
  border: 1px solid black;
  padding: 8px;
  height: 100px;
  overflow: auto;
  left: 8px;
  right: 8px;
}
```

13.6 요약

13 장에서 여러분은 매시업이 자동차 다중 충돌이 아니라, 여러 소스로부터 수집된 컨텐츠를 엮어 놓은 페이지 또는 애플리케이션이라는 사실을 알았다. 우리는 Ajax의 힘이 Open API들과 조합될 때, 어떻게 매시업 애플리케이션을 쉽게 생성할 수 있는지를 보여주는, 야후 지도와 플리커 사진 서비스들을 조합하는 작은 예제들을 만들어 보았다.

또한 HTML 페이지에 아무런 코드(물론, 클래스를 초기화하기 위해 필수적인 것들 외에)도 필요로 하지 않는 자급자족적인 자바스크립트 클래스로 애플리케이션을 생성했다. 애플리케이션 또는 컴포넌트를 생성했었나? 사실, 애플리케이션/컴포넌트 안에 전역 변수 또는 엘리먼트 ID를 사용하지 않았기 때문에, 심지어는 페이지에 하나 이상의 인스턴스를 포함하는 것도 가능했다. 이렇게 함으로써, 컴포넌트에 대한 특별한 이해 없이도 여러분은 이 장에서 사용된 기술들을 여러 페이지에 사용될 수 있는 여러분 자신만의 컴포넌트들에 적용할 수 있다.

이 장의 예제는 이 책 전반에 걸쳐 논의한 많은 기술들을 한데 묶어 놓은 것이다. 우리는 객체 지향 기술을 활용했고, Prototype을 많이 활용했으며, 많은 Ajax 요청을 만들고,

우리의 크로스-서버 프록시를 통해 Open API들을 사용했다. 그리고 수많은 이벤트들을 처리했다.

게다가, 여러분은 Prototype 라이브러리의 다양한 클래스와 함수들을 활용하여 간결하고, 모듈화된, 그리고 잘 조직화된 자바스크립트를 작성하는 방법을 살펴보았다. 우리의 여행 기록 공유 애플리케이션에서 비중있는 사용을 보여준 예제는 자바스크립트 배열에 대한 확장으로 Prototype에 추가된 `each()` 메서드였다. 배열 요소 처리를 전형적인 for 루프를 사용하는 대신, 인자로 개별 배열 요소들을 받는 반복 함수로 모듈화하여 처리 코드를 간결하고, 이해하기 쉬운 함수로 격리시킬 뿐만 아니라, 재사용성을 증진시킨다.

우리는 다시 지도를 표시하기 위해 야후 지도 API를 사용했다. 야후 지도 API에 대한 더 많은 것들이 있다. 여전히 더욱 인터랙티브한 지도 애플리케이션을 만들기 위해 여러분이 적용할 수 있는 더 많은 기능들이 있다.

여러분은 플리커 사진 서비스들에 대해 더 많은 것을 배웠다. 특히 사진 세트(photo set)를 이용해서 사진을 그룹화하고 조직화하는 것을 배웠다. 우리는 이용 가능한 기능들 중 중요한 부분들만 집중적으로 다루었다. 인터랙티브한 사진들을 사용하고 싶은 사이트들 또는 애플리케이션을 제작하고자 한다면, 플리커를 통해 우리가 여기서 조사해온 것들보다 더 많은 것들을 할 수 있다.

자, Open API를 통해 웹상에서 가용한 모든 컨텐츠를 찾아보자. 분명히 여러분은 여러분 자신의 매시업을 만들기 위한 멋진 조합을 찾아낼 수 있을 것이다.

최신 뉴스

최근 야후 파이프(http://pipes.yahoo.com)서비스가 공개되어 매시업의 본질이 무엇인지를 보여주는 예제들을 선보이고 있다. 이 서비스는 끌어 놓기 가능한 그래픽 매시업 저작을 통해 프로그래밍 요소와 기본 데이터 소스(구글 베이스, 야후 기타 등등)외 다양한 내용을 다룰 수 있게 하고 있다. 꼭 확인해 보기 바란다!