

오토마타와 컴파일러

레포트 # 1

이름(학번)	황 준 일 (32131766)
담당교수	이 상 범 교수님
제출일	2019. 09. 16

1. 객체지향언어

1) JAVA

가) java의 실행(번역) 과정

일단 java로 작성한 코드가 어떠한 과정으로 번역되고 실행되는지 알아보자.

(1) Hello.java를 다음과 같이 작성한다.

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

(2) 그리고 Hello.java를 터미널에서 javac로 통해 class 파일(byte code)로 생성한다.

```
> javac Hello.java
```



(3) **Hello.class** 가 생긴것을 확인할 수 있다. 이 파일을 에디터로 열어보면 이렇게 생겼다. 마치 기계어처럼 생겼지만 기계어는 아니다.

01	cafe babe 0000 0034 001d 0a00 0600 0f09	14	7973 7465 6d01 0003 6f75 7401 0015 4c6a
02	0010 0011 0800 120a 0013 0014 0700 1507	15	6176 612f 696f 2f50 7269 6e74 5374 7265
03	0016 0100 063c 696e 6974 3e01 0003 2829	16	616d 3b01 0013 6a61 7661 2f69 6f2f 5072
04	5601 0004 436f 6465 0100 0f4c 696e 654e	17	696e 7453 7472 6561 6d01 0007 7072 696e
05	756d 6265 7254 6162 6c65 0100 046d 6169	18	746c 6e01 0015 284c 6a61 7661 2f6c 616e
06	6e01 0016 285b 4c6a 6176 612f 6c61 6e67	19	672f 5374 7269 6e67 3b29 5600 2000 0500
07	2f53 7472 696e 673b 2956 0100 0a53 6f75	20	0600 0000 0000 0200 0000 0700 0800 0100
08	7263 6546 696c 6501 000a 4865 6c6c 6f2e	21	0900 0000 1d00 0100 0100 0000 052a b700
09	6a61 7661 0c00 0700 0807 0017 0c00 1800	22	01b1 0000 0001 000a 0000 0006 0001 0000
10	1901 000b 4865 6c6c 6f20 576f 726c 6407	23	0001 0009 000b 000c 0001 0009 0000 0025
11	001a 0c00 1b00 1c01 0005 4865 6c6c 6f01	24	0002 0001 0000 1b00 0212 03b6 0004
12	0010 6a61 7661 2f6c 616e 672f 4f62 6a65	26	b100 0000 0100 0a00 0000 0a00 0200 0000
13	6374 0100 106a 6176 612f 6c61 6e67 2f53	26	0300 0800 0400 0100 0d00 0000 0200 0e

(4) 그리고 다시 이것을 수행하기 위해서는 "java" 라는 명령어를 사용해야한다.

```
PS C:\Users\junil\Desktop\폴더> java Hello  
Hello World  
PS C:\Users\junil\Desktop\폴더> █
```

* 이 때 Java 명령어는 Hello라는 이름의 class 파일을 찾아서 실행시킨다.

Hello.class는 Java Virtual Machine이라는 Platform에서 실행된다.

(5) 정리하자면 [*.java] => [javac *.java] => [*class] => [java *]

나) Java Virtual Machine

위에서 java의 코드가 어떻게 실행되는지 살펴봤다. *.java 파일은 " javac " 라는 명령어로 컴파일된다.

그리고 컴파일된 *.class 파일은 " java " 라는 명령으로 JVM에서 인터프리터 형태로 실행된다.

즉, java는 compile language이기도 하며, interpreter language이기도 하다.

그럼 JVM을 사용하는 이유는 무엇일까? 그것은 바로 “이식성” 때문이다.

JVM은 OS(Window, Linux, MacOS, ...)에 종속적이다. 그리고 java는 JVM에 종속적이다. 따라서 JVM만 잘 만들면, java code 자체에는 신경쓰지 않아도 된다는 것이다. 그리고 JVM을 이용하는 언어는 java 뿐만이 아니다. 안드로이드 공식언어로 지정된 Kotlin(코틀린), 그리고 자바 개발자의 생산성을 높일 목적으로 만들어진 Scala(스칼라), 이 외에도 Clojure(클로저), Groovy(그루비) 등 굉장히 다양한 언어들이 존재한다. Kotlin과 Scala의 경우 Java 코드와 섞어서 사용해도 무관하며, java package를 import 해서 사용할 수도 있다.

다) 속도

java는 말했듯이 JVM에서 인터프리터로 실행된다. 즉, 느리다. C로 작성한 프로그램의 경우 기계어로 번역되기 때문에 굉장히 빠르지만 java의 경우 jvm을 통해서 실행되기 때문에 C에 비해 상대적으로 느린편이다. 그래서 **java의 byte code를 기계어로 변환해주는 JIT 컴파일러**가 등장했고, 덕분에 속도 격차가 줄어들었다.

라) 객체지향언어

java는 잘 알려진 객체지향언어이다. 그리고 무조건 class형태로만 구성된다. python이나 c++은 절차지향 형태로도 작성 가능하지만, java의 경우는 “무조건” 객체지향으로 코드를 작성해야한다.

그래서 *.java의 파일명과, 해당 코드의 Class가 일치되어야 한다.

Hello.java의 경우 class Hello {}를 “무조건” 가지고 있어야한다.

객체지향언어이기 때문에 {캡슐화, 정보은닉, 상속, 다형성, 메세지 패싱} 등의 특징을 가지고 있다.

마) 함수형 프로그래밍(람다식)

최근에 함수형 프로그래밍이 대두되면서 java 8 부터는 함수형 프로그래밍(람다식)을 지원하고있다.

바) 메모리관리

C나 C++의 경우 개발자가 코드를 작성할 포인터를 사용할 수 있다. 따라서 개발자가 메모리 관리를 신경써야 한다.

하지만 java는 메모리를 관리해주는 Garbage Collector가 존재한다. 즉, 메모리 관리에 대한 책임이 개발자에서 Garbage Collector로 대부분 위임되었다고 볼 수 있다(완벽하게 위임 된 것은 아니다). 그렇기 때문에 java에는 pointer라는 개념이 없고, 개발자는 메모리에 직접적으로 접근할 수 있는 권한이 없다.

자바에서 객체를 생성할 때(인스턴스가 만들어질 때) 자동적으로 메모리를 할당하고, 사용이 완료되어 접근되지 않을 경우 Garbage Collector가 이를 자동으로 제거한다.

사) 다양한 앱 개발

java는 윈도우, 리눅스, 유닉스, 맥 등 다양한 환경에서 작동한다. 그리고 콘솔/데스크톱/서버/안드로이드 등의 앱을 개발할 수 있도록 되어있다. 사실상 거의 앱을 만들 수 있는 언어라고 볼 수 있다.

아) 기타

멀티스레드 구현, 동적 로딩, 오픈소스 라이브러리 등의 특징을 가지고 있다. 또한 프로그램 실행시 발생할 수 있는 모든 예외들을 처리해야 하며, 이를 지키지 않으면 컴파일 과정에서 에러가 발생한다. 덕분에 규모가 큰 엔터프라이즈 환경에서 개발할 때 이점이 있으며, 반대로 가벼운 앱을 만들기에는 적절하지 못한 언어라고 할 수 있다.

2) Javascript

바로 위에서 java에 대해 언급했다. java는 대부분의 앱을 만들 수 있지만, “브라우저 앱”은 만들 수 없다. 왜냐하면 브라우저 환경에서 작동하는 언어는 오직 “javascript” 밖에 없기 때문이다.

가) 자바와의 연관성?

자바스크립트는 넷스케이프에서 제작되었으며, 처음에는 모카(Mocha)라는 이름으로 개발되었고 이후에 라이브스크립트(LiveScript)라는 이름으로 되었다가 최종적으로 자바스크립트(Javascript)가 되었다. 자바스크립트가 썬 마이크로시스템즈의 자바와 구문이 유사한 점도 있지만, 이는 사실 두 언어 모두 C 언어의 기본 구문에 바탕을 뒀기 때문이고, **자바와 자바스크립트는 직접적인 관련성이 없다. 많은 사람들이 착각하는게 javascript가 java에서 비롯되었다는 것이다.** 사실 이건 마케팅을 노린 효과일뿐이고 직접적인 연관성은 “전혀” 없다. 오히려 javascript 문법과 관련있는 ECMAScript가 발전하면서 C#과 유사해지고 있다.

나) Javascript의 구성

(1) Document Object Model

HTML을 Parsing하여 Tree 형태로 구성하고 접근할 수 있다.

즉, Tag를 선택/수정/삭제/추가 할 수 있는 API라고 할 수 있다.

(2) Browser Object Model

웹 사이트를 이용할 때 우리는 브라우저(크롬, 익스플로러, 파이어폭스, 사파리 등)를 사용한다.

그리고 javascript는 브라우저를 제어할 수 있다.

(3) CSS Object Model

CSS는 태그를 꾸미는 것들이라고 생각하면 된다. javascript는 이러한 CSS 정의 목록에 대한 것을 Tree형태로 구성하고, DOM과 매칭하여 브라우저에 렌더링을 한다. 뿐만 아니라 Javascript CSS를 제어할 수 있다.

(4) ECMAScript (줄여서 ES)

ECMAScript는 Javascript의 문법(스펙)이라고 볼 수 있다. 아마 이 과제에서 중요한 관점은 javascript가 아니라 바로 ECMAScript이다.

다) “객체 지향” 언어가 아닌 “객체 기반” 언어

ECMAScript는 객체지향언어가 아닌 객체기반언어이며 Prototype이라는 개념을 이용하여 “객체지향”을 유사하게 구현할 수 있다.

```
function Person (name) {  
    this._name = name  
    this.getName = function () { return this._name } // dynamic method  
}  
Person.prototype.setName = function (name) { this._name = name } // dynamic method  
Person.getClassName = function () { return 'This is Person' } // static method  
  
var person1 = new Person('황준일1')  
var person2 = new Person('황준일2')  
console.log(person1.getName()) // 황준일1  
console.log(person2.getName()) // 황준일1  
person1.setName('황준일1 수정')  
person2.setName('황준일2 수정')  
console.log(person1.getName()) // 황준일1 수정  
console.log(person2.getName()) // 황준일2 수정
```

이러한 형태의 코드를 구성할 때, 실제로 다음과 같은 Chain 구조가 형성된다.

Person --> Person.prototype

Person.prototype.constructor --> Person

person1.__proto__ --> Person.prototype

person2.__proto__ --> Person.prototype

person1.getName --> Person.prototype.getName

person2.getName --> Person.prototype.getName

person1.setName --> Person.prototype.setName

person2.setName --> Person.prototype.setName

여기서 getName과 setName의 차이를 알아야한다.

getName의 경우 person1과 person2가 메모리상에서 “소유” 하고 있다. 즉, person1이 만들어질 때 getName도 같이 생성된다.

하지만 setName의 경우 Person.prototype이 가지고 있다. 즉, setName의 경우 person1과 person2가 Person.prototype.setName을 공유하고 있다는 것이다.

만약에 Person을 통해서 생성된 객체가 1억개라고 생각해보자. 그럴 때 getName은 1억개가 생성될 것이며, setName은 1개만 생성될 것이다. 즉, 메모리가 굉장히 낭비되는 것이다.

이렇게 ECMAScript에서는 prototype을 이용하여 class를 모방한다. 그 이유는 초기에 컴퓨터는 메모리 공간에 대한 여유가 없었기 때문에, 브라우저에서 작동하는 코드는 “최소한의 메모리”만 사용해야 했기 때문이다. 따라서 브라우저에서 작동하는 “javascript”는 prototype이라는 개념을 사용하여 메모리 공간을 최소한으로 사용하는 방법으로 객체지향을 모방해야할 수 밖에 없었던 것이다.

사실 프로토타입에 대해 깊게 알아보고자 하면 족히 10페이지 분량은 기본적으로 넘어갈 것이다. 굉장히 기초적인 내용만 살펴봤다. 현재는 브라우저 엔진이 지속적으로 업데이트되고, OS를 런타임으로 하는 Node.js가 출시되면서 ECMAScript는 급격하게 발전했다. 현재는 다음과 같은 Class 문법을 지원한다.

```
class Person {
  constructor (name) {
    this.setName(name)
  }
  setName (name) { this._name = name }
  getName () { return this._name }
  static getClassName () { return 'This is Person' }
}

class DankookPerson extends Person {
  constructor (name, major) {
    super(name)
    this.setMajor(major)
  }
  setMajor (major) { this._major = major }
  getMajor () { return this._major }
}

const junil = new DankookPerson('황준일', '소프트웨어공학')
junil.getName() // 황준일
junil.getMajor() // 소프트웨어공학
junil.setName('황준일 수정')
junil.setMajor('소프트웨어공학 수정')

// DankookPerson.prototype === Person
// junil.__proto__ === Person
```

이것은 ECMAScript 2015 (혹은, ECMAScript 6) 부터 지원하는 문법이며, 기존의 Prototype Chain으로 구성하던 class를 조금 더 직관적으로 구성하도록 변경되었다.

그리고 이러한 prototype의 개념 때문에 ECMAScript에서 데이터의 원형(type)은 총 7가지{String, Number, Boolean, undefined, null, Object, Symbol}등이 존재하며

그리고 배열(Array)과 함수(Function) 또한 prototype을 타고 올라가면 Object가 나온다.

즉, Array와 Function은 Object이다.

function a {} 는 var a = new Function() 과 동일하다.

라) 함수형 프로그래밍 지원

javascript에서 함수는 “일급객체”로 취급한다.

```
// 1. 함수를 변수에 담을 수 있다.
var a = function () {
  console.log('this is a')
}

// 2. 함수를 매개변수(Parameter)로 넘길 수 있다.
var b = function (f) {
  console.log('this is b'); f();
}
b(a) // this is b, this is a 출력

// 3. 함수를 반환(return)할 수 있다.
var c = function () {
  var n = 1
  return function () {
    console.log(n++)
  }
}
var d = c()
d(); // 1
d(); // 2
d(); // 3
```

그렇기 때문에 함수형 프로그래밍이 가능하다.

마) 기타

사실 javascript는 현 시점에서 python과 더불어 제일 핫한 언어이다. javascript에 type을 정의하여 사용하는 typescript 또한 javascript의 한 줄기이며, js + ts까지 포함하면 약 25%의 점유율을 가지고 있다.

Year

Quarter

2019

2

# Ranking	Programming Language	Percentage (Change)	Trend
1	JavaScript	19.922% (-2.425%)	
2	Python	17.803% (+1.519%)	
3	Java	10.482% (+0.581%)	
4	Go	7.916% (+0.289%)	
5	C++	7.253% (+0.187%)	
6	Ruby	6.296% (-0.249%)	
7	PHP	5.515% (-0.288%)	
8	TypeScript	5.415% (+0.841%)	
9	C#	4.001% (+0.858%)	
10	C	3.190% (+0.248%)	
11	Shell	2.347% (+0.000%)	
12	Scala	1.444% (+0.017%)	
13	Swift	1.103% (-0.035%)	
14	Rust	0.714% (-0.082%)	
15	Kotlin	0.856% (+0.113%)	^
16	Objective-C	0.528% (-0.113%)	v
17	Elixir	0.448% (+0.079%)	^
18	Groovy	0.443% (-0.069%)	
19	DM	0.398% (-0.123%)	v
20	Dart	0.376% (+0.059%)	^
21	Perl	0.334% (-0.013%)	^

* 사실 저 또한 javascript를 즐겨 사용하는 사람입니다.

3) JSP (Java Server Pages)

java에서 servlet을 이용하여 web server를 개발시, text/html 형태의 response를 만들 때 불편함 인식하고 만들어진 언어다

가) Servlet

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public ThreeParams extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        String title = "Reading Three Request Parameters";
        String docType = "<!DOCTYPE html>\n";

        out.println(
            docType +
            "<html>\n" +
            "<head><title>" + title + "</title></head>\n" +
            "<body>\n" +
            "<h1>" + title + "</h1>\n" +
            "<ul>\n" +
            "<li>" + request.getParameter("param1") + "</li>\n" +
            "<li>" + request.getParameter("param2") + "</li>\n" +
            "<li>" + request.getParameter("param3") + "</li>\n" +
            "</ul>\n" +
            "</body></html>"
        )
    }
}
```

java를 이용하여 웹 서버를 구축할때 위에 코드처럼 servlet이라는 package를 사용한다.

servlet은 http request를 처리하고, response를 통하여 응답 코드를 만드는 데, 이 때 html 코드를 구성하는 것이 굉장히 번거롭기 때문에 JSP라는 것을 만들었고, 사용한다.

나) JSP

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<!doctype html>
<html>
<head>
<title>reading three request parameters</title>
</head>
<body>
<h1>reading three request parameters</h1>
<ul>
<li>param1 : <%= request.getParameter("param1") %>
<li>param2 : <%= request.getParameter("param2") %>
<li>param3 : <%= request.getParameter("param3") %>
</ul>
</body>
</html>
```

이렇게 JSP는 서버에서 response를 text/html 형태로 처리할 때 간편합니다.

그리고 JSP는 Apache Tomcat 같은 WAS(Web Application Server)와 같이 사용되며, WAS가 JSP구문을 해석하여 출력하는 역할을 한다. WAS에서는 기본적으로 다음과같은 객체들을 제공한다

- request: HttpServletRequest Object
- response: HttpServletResponse Object
- session: HttpSession Object
- out: PrintWriter Object
- application: ServletContext Object

그리고 `<%= expression %>` 이러한 표현식으로 변수를 출력할 수 있으며,

이는 `<% out.println(expression) %>` 이 표현식과 동일하다

그리고 `<% %>` 내부에서 기본적인 java 문법을 사용할 수 있다

이 외에도 여러가지 문법과 정의가 존재하지만, 중요한점은 java의 servlet을 간편하게 표현하기위해 만들어졌다는 것이다. JSP의 등장 이후에 MVC Pattern이라는 개념으로 인하여 servlet과 JSP를 같이 사용하였으며, Spring Framework가 MVC Pattern을 굉장히 단순하고 직관적으로 표현했다.

하지만 지금 대부분의 서비스는 MVC Pattern보단 Single Page App과 REST API 형태의 구조를 더 많이 사용하고 선호한다. JSP를 사용하는 곳 회사는 에이전시가 대부분일것이다.

4) Java Applet

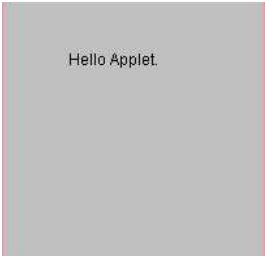
애플릿은 작은 어플리케이션이란 뜻으로 웹 브라우저를 통해 실행될 수 있는 자바클래스를 말한다. 즉, Java 기반의 **웹 브라우저 플러그인 기술**이다.

Java로 작성한 프로그램을 미리 업로드 하고 브라우저 상에서 applet 태그로 불러온다. 애플릿 태그가 실행되면 브라우저에서는 프로그램을 다운로드 한 다음 실행한다. ActiveX, 어도비 플래시 및 AIR, Silverlight와 비슷하며 JVM에서 실행된다.

애플릿을 실행하려면 JVM을 운영체제에 미리 깔아놓아야 한다. JVM위에서 실행되기 때문에 속도가 굉장히 느리다.

가) 특징

1. 애플릿 관련정보가 포함된 HTML문서를 작성해야한다.
2. 웹 브라우저를 통해 실행된다.
3. main method가 필요없다.
4. 애플릿은 java.applet.Applet을 상속하는 public 클래스이어야 한다.

<pre>import java.applet.*; import java.awt.*; public class Hello extends Applet { public void paint(Graphics g) { setBackground(Color.lightGray) g.drawString("Hello, Applet!", 50, 50); } }</pre>	<pre><!DOCTYPE html> <html> <head> <title>HelloWorld_example</title> </head> <body> <h1>A Java applet example</h1> <applet code="Hello.class" width="200" height="200"> </applet> </body> </html></pre>	
---	---	---

HTML코드에서 Hello.class로 사용했는데, 다운로드 시간을 최소화하기 위해 jar로 만들어서 사용하는 게 더 좋다.

`<applet archive="example.jar" code="Hello.class" width="200" height="200"></applet>`

나) 지원종료

모질라 재단에서 2015년 10월 파이어폭스에서 NPAPI 플러그인 지원을 중단하겠다는 발표를 했고, 곧이어 오라클에서는 2016년 1월 Java 9부터 애플릿을 위한 자바 플러그인 지원을 중단하겠다고 발표했다. 따라서 자바 애플릿은 Java 9 이후 역사 속으로 사라질 예정이며, 이후 자바 애플릿이 했던 역할은 유사한 기술인 Java Web Start가 대신 하게 된다.

사실 HTML5에서는 아예 폐지되었으며 대신 embed 태그를 쓴다. 아니면 그냥 canvas 태그를 사용하여 javascript로 만드는 것이 더 유익하다.

2. Markup Language

1) XML(eXtensible Markup Language)

XML은 W3C에서 개발된 다목적(특수목적) 마크업 언어이다. “마크업”이라는 뜻은, 쉽게 말해 단순한 텍스트에 의미를 부여한다고 생각할 수 있다. HTML은 미리 정의된 태그가 있는 반면, XML은 태그를 직접 정의한다.

원래 동일한 목적을 가진 Standard Generalized Markup Language(SGML) 라는 것이 인터넷 등장보다 빠른 1980년대에 등장했으며, 이것의 Profile(부분집합)에 인터넷 환경과 새롭게 변화해온 컴퓨팅 환경에 맞게 확장을 더한 것이 바로 XML 이다. XML은 바로 SGML의 파생형이라고 할 수 있으며, HTML은 완전한 파생형은 아니지만 SGML의 영향을 강하게 받았다고 최초 제안자인 팀 버너스리가 밝힌 적이 있다.

XML은 많은 종류의 데이터를 기술하는 데 사용할 수 있다. 주로 다른 종류의 시스템, 특히 인터넷에 연결된 시스템끼리 데이터를 쉽게 주고 받을 수 있게 하여 HTML의 한계를 극복할 목적으로 만들어졌다. 예를 들어 HTML에서는 CPU 2.83GHz라는 데이터를 표기할 때 어디부터가 데이터 명이고 어디부터가 실제 데이터인지 표시할 수 있는 마땅한 방법이 없다.

XML을 이용하면 {데이터 이름, 실제 데이터, 단위}에 대한 표현이 다음과 같이 가능하다.

```
<dataname>CPU</dataname>
<datavalue>2.83</datavalue>
<dataunit>GHz</dataunit>
```

기계는 인간의 언어를 읽거나 이해할 수 없는 계산기에 불과하므로 XML과 같은 구조화된 마크업 언어들은 인간의 읽고 분석하여 이해하는 능력과 컴퓨터의 단순한 계산적인 판독 능력 사이에 타협점을 만들어 줄 수 있다.

W3C는 XML 설계 목표에서 단순성과 일반성, 그리고 인터넷을 통한 사용 가능성을 강조했다. XML은 텍스트 데이터 형식으로 유니코드를 사용해 전 세계 언어를 지원한다. XML을 설계할 때는 주로 문서를 표현하는데 집중했지만, 지금은 임의의 자료구조를 나타내는 데 널리 쓰인다.

XML을 사용하는 대표적인 예로 RSS가 있다. 검색 봇이 웹 사이트를 탐색할때 사이트에 RSS Feed를 있는지 살펴본다. 그래서 웹 서비스에 RSS Feed가 구현 되어 있으면 더욱 검색이 잘 된다.

그 밖에 Java로 웹 어플리케이션을 작성할 때 web.xml을 작성하는데, 이것도 Oracle(과거 Sun Microsystems)에서 XML Schema로 형식을 규정한 JSP/Servlet Web Application 표준 설정 파일이다. C#에서는 프로그램 설정에 사용하는 App.config, WPF에서 사용하는 Xaml 파일이 있다.

XML이 초기에 널리 퍼진 이유 중 하나는, 거의 웬만한 언어에서 기본으로 지원하는 구조화된 데이터를 읽는 기본 API 보다 훨씬 복잡한 형태의 구조화된 데이터를 읽고 쓰는데 매우 편리했기 때문이다. 지금은 웹 관련 한정으로 JSON이 더 많이 쓰이고 있다.

요즘에는 웹환경이 아닌 일반 TCP/IP 네트워크 통신을 할 때에도 점점 사용빈도가 늘어나고 있는 추세이다. 디지털 기반의 네트워크 통신을 할 때 송신자와 수신자간에서는 서로 주고받는 데이터 패킷을 비트 단위로 정확하게 맞춰줘야 문제가 생기지 않는데, 이 비트단위 데이터 통신에 불확실성이 많기 때문이다. 가장 큰 불확실성은 사람이 만든 버그 또한 데이터 패킷단위 전송시 애플리케이션의 개발기반, 구동 환경에 따라 이런저런 잔처리가 많이 들어간다. 그래서 요즘은 아예 XML로 데이터를 주고받는다. 이렇게 하면 최소한 패킷 사이즈가 안 맞아서 발생하는 버그는 줄일 수 있으며 이렇게 만든 프로그램은 XML로 데이터를 주고받기 때문에 웹확장성도 갖게된다.

2) RDF(자원 기술 프레임워크, Resource Description Framework)

웹상의 자원의 정보를 표현하기 위한 규격이다. 상이한 메타데이터 간의 어의, 구문 및 구조에 대한 공통적인 규칙을 지원한다. 웹상에 존재하는 기계해독형(machine-understandable)정보를 교환하기 위하여 w3c에서 제안한 것으로, 메타데이터간의 효율적인 교환 및 상호호환을 목적으로 한다. 메타 데이터 교환을 위해서 명확하고 구조화된 의미표현을 제공해 주는 공통의 기술언어로 XML을 사용하기도 한다.

데이터모형

{ resource, property type, value } 등이 있으며 메타데이터를 정의하고 사용하기 위한 추상적이고 개념적인 구조를 제공한다.

RDF 구문

메타데이터를 교환하고 작성하기 위해서는 구체적인 구문(syntax)이 필요하다. 이를 위해서 상호호환성/확장성/검증/가독성/웹 상에서의 운용가능성/복잡한 문서의 논리적 구조 등을 표현해 줄 수 있는 XML을 사용하는 경우가 많다. 다만 문서유형정의부(DTD)를 정의하지 구조를 취한다. Namespace정의는 다음과 같은 형식이다. `<?xml:namespace name="URI" as="some-abbreviation" ?>`

RDF 스키마

특정 메타데이터에서 정의하고 있는 어휘들을 선언하기 위해서 사용된다. 어휘란 자원을 기술하기 위해 각 메타데이터 형식들에서 정의하고 있는 메타데이터 요소집합이다. 인간이 읽을 수 있고(human-readable) 기계처리가 가능한(machine-processable) 어휘들을 정형화 하는 것은 상이한 메타데이터 형식들간의 어휘 확장과 재사용, 상호교환을 가능하게 해 주는 것이며, 이러한 정형화를 위한 것이 바로 RDF 스키마이다. RDF 스키마는 RDF의 데이터 모형과 구문 명세에 의해서 표현된다. 현재 RDF 스키마 명세는 개발단계에 있으며, 더 많은 연구가 필요하다. 기본적인 RDF스키마 유형으로는 property, propertyType, instanceOf, subclassof, allowedPropertyType, Range 가 있다.

3) OWL(Web Ontology Language)

웹에서 표현되는 다양한 어휘 정보를 여러 웹 응용 프로그램들이 재활용할 수 있도록 지원하는 온톨로지 저작용 언어이다. 온톨로지의 사물(thing) 중 어휘를 대상으로 하여, 웹 형식의 문서/ 어휘 / 어휘 그룹 / 어휘 관계 등을 논리적으로 풍부하고 유연하게 기술할 수 있는 장점이 있다.

추론 시스템에 { 축적된 명제들을 정의 / 클래스 및 구성원 간의 관계 기술 / 구문적으로 정의되지 않은 사실의 논리적 유추를 가능하게 하는 클래스 및 속성 / 제약 사항 } 등으로 구성된다. OWL에서 표현 수준에 따라 세 가지 하위 언어가 있다.

OWL Lite

클래스 계층 구조와 단순 제약 사항 정도만 표현하는 경우를 대상

OWL DL(Description Logic)

OWL의 모든 어휘를 포함, 어휘의 속성이나 관계 표현에 대한 제약, 기술 논리 비즈니스 분야를 지원하고 추론 시스템에 적합한 계산 속성

OWL Full

계산 결과나 시간 등에 대한 보장이 없이 최대 표현력과 RDF의 유연한 문법을 모두 지원, 어휘 조합을 통해 온톨로지를 확장

예시

```
<owl:class rdf:id="Wine">
  <rdfs:subclassof rdf:resource="#food:PotableLiquid">
    <rdfs:subclassof>
      <owl:restriction>
        <owl:onproperty rdf:resource="#hasMaker">
          <owl:allvaluesfrom rdf:resource="#Winery"></owl:allvaluesfrom>
        </owl:onproperty>
      </owl:restriction>
    </rdfs:subclassof>
  </rdfs:subclassof>
</owl:class>
```

4) Ontology

사람들이 세상에 대하여 보고 듣고 느끼고 생각하는 것에 대하여 서로 간의 토론을 통하여 합의를 이룬 바를, 개념적이고 컴퓨터에서 다룰 수 있는 형태로 표현한 모델로, 개념의 타입이나 사용상의 제약조건들을 명시적으로 정의한 기술이다. 온톨로지는 일종의 지식표현(knowledge representation)으로, 컴퓨터는 온톨로지로 표현된 개념을 이해하고 지식처리를 할 수 있게 된다.

온톨로지는 프로그램과 인간이 지식을 공유하는데 도움을 주는 역할을 해주며 정보시스템의 대상이 되는 자원의 개념을 명확하게 정의하고 상세하게 기술하여 보다 정확한 정보를 찾을 수 있도록 한다. 온톨로지는 시맨틱 웹을 구현할 수 있으며 지식개념을 의미적으로 연결할 수 있는 도구로서 RDF, OWL, SWRL 등의 언어를 이용해 표현한다.

온톨로지는 일단 합의된 지식을 나타내므로 어느 개인에게 국한되는 것이 아니라 그룹 구성원이 모두 동의하는 개념이다. 그리고 프로그램이 이해할 수 있어야 하므로 여러 가지 정형화가 존재한다.

전산/정보과학에서 온톨로지란 도메인을 기술하는 데이터 모델로서, 이에 속한 개념과 개념 사이의 관계를 기술하는 정형(formal) 어휘의 집합으로 이루어진다. 예를 들어 "종-속-과-목-강-문-계"로 분류되는 생물과 생물 사이의 분류학적 관계나, 혹은 영어 단어 사이의 관계를 정형 어휘로 기술하면 각각 온톨로지라고 할 수 있다. 정형 언어(formal language)로 기술된 어휘의 집합인 온톨로지는 연역과 추론에 사용된다.

웹 정보 검색은 웹을 통해 접근할 수 있는 모든 전자자원을 대상으로 하는 검색을 가능하게 하였다. 웹의 급속한 발달로 인해 검색 대상 범위가 확대됨에 따라 보다 정교한 검색을 필요로 하게 되었으며, 지능화된 정보 검색 시스템 개발을 촉진하는 계기가 되었다. 이런 계기를 바탕으로 **웹 자원을 효과적으로 관리할 수 있는 정보 검색의 새로운 도구의 필요성**이 대두되었고, 온톨로지가 각광을 받게 되었다.

온톨로지는 자연어의 기계 번역과 **인공지능 분야에서 활용**되며, 최근에는 특정 분야의 인터넷 자원과 그 사이의 관계를 기술하는 온톨로지를 사용하는 시맨틱 웹과 이것에서 파생된 시맨틱 웹 서비스 등의 핵심 요소로서 주목받고 있다.

온톨로지의 구성 요소는 클래스(class), 인스턴스(instance), 관계(relation), 속성(property)으로 구분할 수 있다.

클래스(Class)

일반적으로 우리가 사물이나 개념 등에 붙이는 이름을 말한다고 설명할 수 있다. "키보드", "모니터", "사랑"과 같은 것은 모두 클래스라고 할 수 있다.

인스턴스(Instance)

사물이나 개념의 구체물이나 사건 등의 실질적인 형태로 나타난 그 자체를 의미한다. 즉, "LG전자 ST-500 울트라슬림 키보드", "삼성 싱크마스터 Wide LCD 모니터", "로미오와 줄리엣의 사랑"은 일반적으로 인스턴스라 볼 수 있다.

속성(Property)

클래스나 인스턴스의 특정한 성질, 성향 등을 나타내기 위하여 클래스나 인스턴스를 특정한 값(value)과 연결시킨 것이다. 예를 들어, "삼성 싱크마스터 Wide LCD 모니터는 XX인치이다."라는 것을 표현하기 위하여, hasSize와 같은 속성을 정의할 수 있다.

관계(Relation)

클래스, 인스턴스 간에 존재하는 관계들을 칭하며, taxonomic relation과 non-taxonomic relation으로 구분할 수 있다.

Taxonomic Relation은 클래스, 인스턴스들의 개념분류를 위하여 보다 폭넓은 개념과 구체적인 개념들로 구분하여 계층적으로 표현하는 관계이다. 예를 들어, "사람은 동물이다"와 같은 개념간 포함관계를 나타내기 위한 "isA" 관계가 그것이다.

Non-taxonomic relation은 Taxonomic Relation이 아닌 관계를 말한다. 예를 들어, "운동으로 인해 건강해진다"는 것은 "cause" 관계(인과관계)를 이용하여 표현한다.

언어 온톨로지(Linguistic Ontology)

텍스트에서 추출된 데이터나 정보에 대한 자연어 인터페이스를 지원한다. (예: CYC 온톨로지, EDR, WordNet)

공리 온톨로지(Axiomatized Ontology)

정보의 이해를 표현하는데 이용되는 규칙, 이론, 제한점 등을 자동으로 생성시키는 것을 지원한다. (예: 웹 온톨로지)

인공지능 온톨로지 : 언어 + 공리

3. 기타언어

1) R

통계 계산과 그래픽을 위한 프로그래밍 언어이자 소프트웨어 환경이다. R는 GPL 하에 배포되는 S 프로그래밍 언어의 구현으로 GNU S라고도 한다. R는 통계 소프트웨어 개발과 자료 분석에 널리 사용되고 있으며, 패키지 개발이 용이해 통계 소프트웨어 개발에 많이 쓰이고 있다. R의 문법과 통계처리 부분은 S를 참고했고, 데이터 처리부분은 스킴으로부터 영향을 받았다.

특징

- 다양한 통계 기법과 수치 해석 기법을 지원한다.
- 사용자가 제작한 패키지를 추가하여 기능을 확장할 수 있다. 핵심적인 패키지는 R와 함께 설치되며, CRAN(the Comprehensive R Archive Network)을 10,300개 이상의 패키지를 내려 받을 수 있다.
- 수학 기호를 포함할 수 있는 출판물 수준의 그래프를 제공한다.
- 통계 계산과 소프트웨어 개발을 위한 환경이 필요한 통계학자와 연구자들 뿐만 아니라, 행렬 계산을 위한 도구로서도 사용될 수 있다.
- 윈도, 맥 OS 및 리눅스를 포함한 UNIX 플랫폼에서 이용 가능하다.
- 계산 작업의 경우 C, C++, Fortran 코드를 런타임에 링크하고 호출할 수 있다.
- R 객체를 C, C ++, Java, .NET, Python 코드로 작성할 수 있다
- S의 전통으로 인해 R는 대부분의 통계 컴퓨팅 언어보다 강력한 객체 지향 프로그래밍 기능을 제공한다.

2) Haskell

순수 함수형 프로그래밍 언어. I/O와 같이 필요한 경우가 아니면 Side Effect가 없는 순수 함수로만 만들어졌다. 함수형 언어와 컴퓨터 아키텍처 연구 학술회에서 1987년부터 설계를 시작했으며 3년 뒤인 1990년에 1.0 버전의 보고서가 발표되었고, 1.1~1.4 버전을 거쳐, 1999년에 가장 널리 알려진 버전인 하스켈 98 보고서가 나왔다. 현 시점의 최신 정의는 하스켈 2010이다.

특징

-

- 패턴 맞춤 / 커링 / 조건제시법 / 가드 / 연산자 정의
- 재귀 함수 / 대수적 자료형 / 느긋한 계산법
- 단일체, 타입 클래스 등은 하스켈만의 독창적인 개념이며 이러한 특징들은 절차적인 프로그래밍 언어에서 매우 힘들었던 함수 정의를 손쉽게 만들어 버린다.
- 수학의 한 특이 분야인 범주론의 개념들, 특히 함수의 개념의 추상화된 형태인 사상과 모나드(monad)를 차용하여 가져온 언어인데, 이를 통해 함수를 대상으로써 다룸에 있어서 명확성을 가질 수 있다.
- 매사추세츠 공과대학교와 글래스고 대학교가 개발한 버전은 병렬화가 가능하기 때문에 '병렬 하스켈'이라고 불린다. 병렬화와 분산 처리를 더욱 강화한 '분산 하스켈'과 에덴 프로그래밍 언어가 나왔고, 느긋한 계산법 대신 적극적인 계산법을 쓰는 '적극적 하스켈'이 있으며 하스켈에 객체 지향 개념을 도입한 버전으로 '하스켈++', '오하스켈', 몬드리안 프로그래밍 언어 등등이 있다.
- 하스켈과 비슷한 언어로 그래픽 사용자 인터페이스 개발에 새로운 방법을 도입한 클린이 있다. 이 언어와 하스켈의 가장 큰 차이점은 입출력을 위해 단일체 대신에 유일형을 사용한다는 것이다.

* 부수효과(side effect)

같은 입력임에도 불구하고 출력이 달라지는 것을 말한다. 순수 함수형 언어에서 함수의 결과값은 파라미터(parameter)로 넘겨진 입력값에 의해서만 결정된다. 순수한 함수들로 구성된 프로그램은 부수효과가 없어 실행 순서의 영향에서 자유롭기 때문에 병행성을 가지기 좋다. 또한 루프문이 없고 제어문도 아주 적으며, 대부분의 경우 제어문을 사용할 필요가 없다. 대부분의 분기는 패턴 매칭과 가드에 의해 만들어진다.

* 느긋한 계산

느긋한 계산(Lazy Evaluation)은 계산이 필요한 순간까지 계산을 미루어 둔다는 의미다. 따라서 그때 그때 필요한 만큼만 계산하는 것도 가능하다. 이 느긋한 계산의 개념으로 인하여 하스켈은 프로그램에 무한의 개념을 쉽게 적용할 수 있다. 예를 들어 어떤 길이의 정수 제공 리스트가 필요하다면 아래처럼 무한한 리스트를 만들게 해도 상관 없다.

* 예제 : 계승 구현

```
fac n = product [n..1]
```

4. 자연어처리(번역기)

1) 인공지능망 기계번역(NMT)

인공지능(AI)의 기반이 되는 머신러닝, 딥러닝 등 기술이 등장하기 시작하면서 사람이 번역하는 것에 비해 부자연스러운 점이 많았던 번역기의 품질이 개선되기 시작했다.

2016년, 구글과 네이버가 '인공지능망 기계번역(NMT)' 엔진을 내놓으면서 더 정교한 번역 서비스를 제공할 수 있게 됐다. 통계기반 번역(SMT)과 비교해 훨씬 자연스럽게 번역이 이뤄진다. NMT는 사람의 뇌가 학습하는 과정을 본딴 기술을 번역에 적용한 것이다.

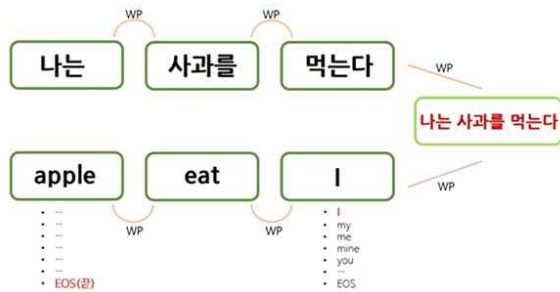
기존의 구글/네이버 번역은 통계 기반 기계번역(Statistical Machine Translation, SMT) 혹은 구문 기반 기계번역(Phrase Based Machine Translation, PBMT)이라고 부르는 방식을 썼다. 한국어-영어 번역 과정을 예로들면 SMT, PBMT, RBMT는 단어나 구문이 가진 여러가지 의미를 저장해 놓는다. 일종의 번역사전을 만들어 놓는 것이다. 다음으로 사용자가 문장을 입력하면 이를 단어나 구문 단위로 쪼개 뒤 통계적으로 가장 본래 의미에 가깝다고 판단되는 번역결과를 제시한다. SMT는 '말뭉치(corpus)'를 미리 번역 엔진에 입력해 놓은 뒤 통계적으로 봤을 때 번역을 요청한 문장 내에 단어나 구문과 가장 비슷하다고 판단한 결과를 내놓는다는 설명이다. 말뭉치는 사람이 읽을 수 있는 텍스트를 컴퓨터도 이해할 수 있는 형태로 모아 놓은 자료를 말한다.

이처럼 통계적인 방법을 쓰게 되면 우리나라말로 '밤'이 낮의 반대말인 밤(night)을 뜻하는지, 먹는 밤(chestnut)을 말하는 것인지를 분간하기 힘들다. 더구나 한국어-영어 번역의 경우 주어와 서술어 등 어순이 다르다는 점도 반영해야하지만 SMT는 전체 문장이 흘러가는 맥락(context awareness)에 대한 이해가 부족했다.

NMT는 이러한 점을 보완하는 번역 기술이다. 가장 큰 차이점은 단어나 구문 단위로 쪼개는 것이 아니라 **문장 단위로 번역한 결과**를 보여준다는 점이다. 머신러닝 기술이 적용된 엔진을 통해 전체 문맥을 파악한 다음 문장 내에 단어, 순서, 의미, 문맥에서의 의미차이 등을 반영한다는 설명이다.

신경망 기계번역 방식을 보다 쉽게 설명하자면 다음과 같다. 입체 공간이 있다고 가정하자. 먼저 '먹다'라는 단어를 공간에 띄운다. 그리고 그 근처에 '먹었다', '먹을 거다', '먹고 싶다' 등 '먹다'라는 단어와 관계가 있는 단어들을 유사한 공간에 둔다. 이 '먹다'라는 단어에는 다양한 차원이 있을 수 있다. 이 차원에 따라 또 다른 단어들과 관계를 맺을 수 있다. 예컨대 치킨, 피자, 케이크 등 '먹다'와 함께 쓰일 수 있는 단어들이 또 '먹다'와 나름의

관계를 맺고 공간상에 위치할 수 있다. 이렇게 단어나 구 등이 공간에서 관계를 맺으며 맵핑된다. 이때 가지는 벡터값을 '단어 표현'이라고 한다. 번역기에 사용되는 단어는 200차원의 단어 표현 값으로 변환된다.



'나는', '사과를', '먹는다', 'I', 'eat', 'apple'은 각각 단어 표현 값으로 변환된다. 그리고 이 단어 표현들을 이어가며 번역하려는 문장에서 결과 문장으로 이어주는 최적의 가중치(Weight parameter)들을 찾아 행렬 곱으로 이어가 벡터를 구해가는 방식이다.

여기서 번역하려는 문장과 결과 문장을 컴퓨터에 주고, 결과 문장이 나오게 하는 값을 찾아내는 최적의 가중치(WP)를 반복적인 기계학습을 통해 자동으로 컴퓨터가 학습한다. 번역은 EOS(문장의 끝, End Of Sentence)값이 가장 높아지면 끝난다. 번역 언어가 달라질 때마다 가중치 값이 바뀐다. 이처럼 인공지능망 기계번역은 입력 문장과 출력 문장을 하나의 쌍으로 두고, 최적의 답을 찾는 중간 값을 학습한다.

인공지능망 기계번역 방식은 통계적 기계번역보다 번역 시스템이 단순하다는 장점을 가진다. 입력 문장과 출력 문장만 있으면 알아서 학습하게끔 유도하기 때문에 구조 자체가 그렇게 어렵지 않기 때문이다.

인공지능망 기계번역은 확장하기 쉽고 다양한 구조를 채택할 수 있다는 것도 장점이다. 다만 학습 시간이 다소 오래 걸릴 수는 있다. 이 문제를 해결하기 위해 병렬처리 등의 방식을 사용한다. 그러나 인공지능망 기계번역은 아직 초창기이기 때문에 많은 문제점과 가능성이 병존하고 있다.

2) 구글번역기

구글의 경우 지난 10년 간 적용했던 PBMT 방식과 비교해 NMT를 적용해 훨씬 좋은 품질의 번역결과를 제공한다. 버락 투로프스키 구글 번역 프로젝트 매니저먼트 총괄은 "NMT 기술 덕분에 구글 번역은 위키피디아 및 뉴스매체의 샘플문장을 기준으로 주요 언어 조합을 평가대상으로 했을 때 번역 오류가 55%에서 85% 가량 현저히 감소하는 등 지난 10년 간 쌓아온 발전 이상의 결과를 단번에 이룰 수 있었다"고 했다.

구글은 아직까지 NMT에서 개선해야할 과제들이 있다고 설명했다. "사람 번역가라면 빼놓지 않았을 고유명사, 희귀용어를 오역하거나 문단 또는 문맥을 고려하지 않은 문장 번역 등과 같은 오류가 발생한다"는 것이다.

3) 파파고 - 네이버 번역기

네이버 파파고의 경우 네이버랩스가 개발한 네이버의 NMT 기술인 'N2MT'를 적용했다. 이 기술은 기존 SMT에 일부 딥러닝을 적용하는 방법과 달리 전체 번역 프레임워크를 딥러닝 방식으로 구축해 번역 정확도를 2배 이상 높여 더 자연스러운 번역결과를 체험할 수 있게 한다.

네이버랩스가 발표한 '문자 단위의 Neural Machine Translation'이라는 논문에 따르면 NMT가 SMT와 구분되는 특징은 최소한의 전문지식만 필요하다는 점이다. 번역을 위해 필요한 신경망의 구조만 잘 결정해 주면 알아서 머신러닝 혹은 딥러닝 과정을 거쳐 입력된 데이터가 많을수록 더 자연스러운 번역이 가능해진다고 한다.

네이버 관계자는 "파파고를 활용한 NMT 기반 번역이 이전 방식인 SMT와 비교해 특히 한국어-중국어 간 번역에서 정확도가 높다"고 말했다. 자체 테스트 결과 한국어를 중국어로 번역할 경우 평균 점수 대비 160% 높은 정확도를 보였고, 중국어를 한국어로 번역할 경우에는 233%가 높았다는 것이다.