나만의 Windows Live 가젯 만들기
체스 기보 뷰어 만들기

Windows Live 가젯을 잘 만들기 위해서 가장 중요한 것은 자바스크립트를 정확히 이해하는 것이다. 지난 시간에 배운 기초적인 자바스크립트 지식을 활용해서 체스 기보 뷰어를 작성해 본다. 이 과정에서 기초적인 체스 규칙과 그것들을 자바스크립트로 분석해서 화면에 출력하는 방법에 관해서 다룬다.

연재 가이드
운영체제: 윈도우 2000/XP
개발도구: Editplus, IE7 or FireFox
기초지식: Javascript, HTML, CSS
응용분야: Windows Live Gadget 프로그램, AJAX 프로그램

연재 순서
2007. 02 이상한 나라의 자바스크립트
2007. 03 체스 기보 뷰어 만들기
2007. 04 Hello, World 가젯 만들기.
2007. 05 StockViewer 가젯 만들기

필자 소개

신영진 pop@jiniya.net, http://www.jiniya.net
삶에는 영원한 상승도, 하락도 없다. 단지 그것들이 조화롭게 반복될 뿐이다. 지금 일이 잘 풀린다고 경거망동 해서는 안되며, 안 좋은 일들만 벌어진다고 의기소침할 필요도 없다. 긴 시간을 두고 봤을 때 그런 일들의 합은 늘 0 이기 때문이다. 그래서 인생은 빈손으로 와서 빈손으로 간다는 말이 있는지도 모르겠다.

필자 메모

보통 소프트웨어 개발에서 설계는 무척 중요한 단계라고 말한다. 소프트웨어 공학에서도 사전 준비를 무척이나 강조한다. 코딩은 그런 만반의 준비가 끝난 다음에나 하는 일이다. 그런데 사실 필자는 프로그램을 작성할 때 설계를 잘 하지 않는다. 이유야 여러 가지가 있겠지만 아마도 프로그래밍을 혼자서 막 배워서 그런 경향이 강한 것 같다.

이번 체스 기보 뷰어도 아무 설계 없이 하나씩 만들기 시작했다. 그러자 어느 순간 생각하지 못했던 여러 가지 경우가 생기고 코드는 점점 복잡해지기 시작했다. 급기야 나중에는 작성한 사람 조차도 손대지 못하는 괴물이 되어 버렸다. 그래서 어쩔 수 없이 다음날 연습장에다 무엇을 어떻게 만들지 고민한 다음 새로 만들었다.

필자는 이번 사례로 설계의 중요함을 다시금 느꼈다. 하지만 아직도 여전히 소프트웨어 공학에서 말하는 복잡한 설계에는 반대한다. 아마 이번 사례에서도 엉망진창의 코드(http://www.jiniya.net/script/chess/chess.js 참고)를 만들어 보지 안았다면 제대로 된 체스 기보 뷰어를 설계하지 못했을 것이기 때문이다. 언제나 그렇듯 가장 중요한 것은 그 사이에서 균형을 찾는 것이다.

체스는 굉장히 논리적인 게임이다. 제한된 수를 가지고 서로 싸우기 때문에 수 읽기가 중요하고, 플레이어의 논리적인 판단 능력에 따라서 승부가 좌우 된다. 한국의 장기, 중국의 상기, 일본의 쇼기, 서양의 체스가 근원은 모두 인도로 같기 때문에 기물의 행마법과 보드 배치의 차이는 있으나 게임의 큰 흐름은 유사하다. 하지만 체스의 가장 큰 특징은 컴퓨터 엔진이 많이 개발되어 있다는 점이다. 따라서 잘 두는 상대가 옆에 없어도 배울 수 있다는 점이 매력적이라고 할 수 있다.

고수의 경기를 배우기 위해서 스타크래프트 리플레이를 보는 것처럼 바둑이나 체스와 같은 보드 게임도 경기를 분석하기 위해서 기록해 두는 관례가 있다. 이렇게 경기를 기록해 둔 것을 기보라고 한다. 이번 시간에 우리는 이러한 체스 기보를 인터넷 상에서 손쉽게 볼 수 있도록 체스 기보 뷰어를 만들어 본다.

1. 체스 기보 표기법

체스의 기보를 표기하는 방법을 간단히 살펴보자. 체스에는 총 여섯 개의 기물이 등장한다. 폰, 룩, 나이트, 비숍, 퀸, 킹이 그것이다. 각 기물의 머릿 글자를 따서 P, R, N, B, Q, K 로 표기한다. 이 중에서 폰은 기물의 개수가 많기 때문에 보통 P 기호는 생략한다.

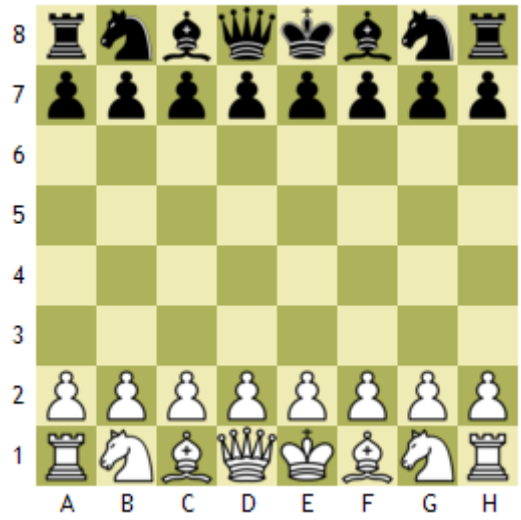


그림 1 일반적인 체스판의 배치

<그림 1>은 체스를 시작할 때 기물이 배치된 상황을 보여준다. 가로로 a 부터 h 까지의 파일이 있고, 세로로 1 부터 8 까지의 랭크가 있다. 각 자리는 파일, 랭크 순으로 읽는다. a8 은 좌측 상단에 있는 칸을 나타낸다. a2 부터 h2 까지의 기물이 폰이다 a1, h1 에 있는 기물이 룩, b1 이 나이트, c1 이 비숍, d1 이 퀸, e1 이 킹이다.

표 1 아인슈타인과 오펜하이머의 체스 기보

1.e4 e5 2.Nf3 Nc6 3.Bb5 a6 4.Ba4 b5 5.Bb3 Nf6 6.0-0 Nxe4 7.Re1 d5 8.a4 b4 9.d3 Nc5 10.Nxe5 Ne7 11.Qf3 f6 12.Qh5 g6 13.Nxg6 hxg6 14.Qxh8 Nxb3 15.cxb3 Qd6 16.Bh6 Kd7 17.Bxf8 Bb7 18.Qg7 Re8 19.Nd2 c5 20.Rad1 a5 21.Nc4 dxc4 22.dxc4 Qxd1 23.Rxd1 Kc8 24.Bxe7

<표 1>은 아인슈타인과 오펜하이머의 체스 경기 기보다. 체스 기보는 기본적으로 순번, 백이 둔 수, 흑이 둔 수 형태로 구성된다. 1. e4 e5 에서 1 은 첫 번째 수 임을, e4 는 Pe4 에서 P 가 생략된 형태로 백이 폰을 e4 로 이동시켰음을 나타낸다. e5 는 흑의 수로 폰을 e5 로 이동시켰음을 나타낸다. 다음에 나오는 2 를 다음 줄에 표시하는 것이 일반적이나 지면 관계상 모두 한 줄로 표시했다. 중간에 나오는 Nxb3 은 나이트가 b3 으로 이동하면서 상대편 기물을 잡아먹었음(x)을 나타낸다. dxc4 는 d 파일에 있는 폰(P)이 c4 로 이동하면서 상대편 기물을 잡아먹은 것을 나타낸다.

이번 연재의 내용과 코드를 이해하기 위해서는 기본적인 체스 규칙을 숙지하고 있어야 한다. 체스 규칙을 잘 모른다면 http://www.jiniya.net/tt/category/%C3%BC%BD%BA 를 참고하도록 하자.

2. 기보 분석기

기보 뷰어를 제작하기 위해서 우리가 가장 먼저 해야 할 일은 기보를 분석해서 프로그램이 이해할 수 있는 형태로 변환하는 작업이다. 프로그램에서 이해하는 각 명령어의 형태가 <리스트 1>에 나와있다. 명령 종류에는 세 가지가 있다. QSC, KSC, null 이 그것이다. QSC 는 퀸 사이드 캐슬링을, KSC 는 킹 사이드 캐슬링을, null 은 일반적인 이동 명령을 의미한다. 기물 종류는 앞서 설명했던 P, K, N, Q, B 가 저장된다. file 에는 이동 전의 기물이 있던 파일의 위치를 저장한다. to 에는 기물이 이동한 위치를 나타내는 Square 객체가 저장된다. capture 는 이동 중 상대편 기물을 잡아 먹었는지가 true, false 형태로 저장된다. promotion 에는 폰이 승격한 경우에 어떤 기물로 승격했는지가(P, K, Q, N, R, B) 저장된다.

리스트 1 프로그램에서 사용될 명령어 구조체

```
function Square(square)
{
    if(square.length < 2)
    {
        this.x = 1;
        this.y = 1;
        return;
    }

    this.x = FileToInt(square.charAt(0));
    this.y = parseInt(square.charAt(1));
}

function Command(type, sig, file, to, capture, promotion)
{
    this.type = type; // 명령 종류
    this.sig = sig; // 기물 종류
```

```
    this.file = file; // 이동한 기물의 파일
    this.to = to; // 이동한 위치
    this.capture = capture; // 잡아먹은 기물
    this.promotion = promotion; // 승격한 기물
}
```

각 명령을 위의 형태로 분석해 주는 일은 CommandParser 의 Run 메소드가 수행한다. record 에 e4, Qxh5 등의 각 플레이어가 수행한 기보 내용을 넣어주면 거기에 대응하는 Command 구조체를 리턴해 준다. 체스 기보의 끝에는 각종 수식어(++, ??, !)가 붙을 수 있기 때문의 분석을 용이하게 하기 위해서 끝에서부터 해석한다.

```
function CommandParser() {}

CommandParser.prototype.Run = function(record)
{
    var cmd = new Command(null, null, null, null, null, null);

    this.record = record;
    index = record.length - 1;
    return this.Type(index, cmd);
}
```

체스 기보에 표기되는 내용은 앞서 설명한 것과 같이 기물, 위치 순으로 표기하는 것이 대부분이다. 하지만 이 표기 방법을 벗어나는 것이 몇 가지 있다. 이런 것은 체스 특별 규칙으로 별도의 표기법을 사용한다. <표 2>에는 이러한 특별한 표기법을 사용하는 규칙에 대한 표기법이 나와있다.

표 2 체스 특별 규칙 표기법

기보 표기	의미
0-0	킹 사이드 캐슬링
0-0-0	퀸 사이드 캐슬링
h8=Q	폰이 h8 로 이동하면서 퀸(Q)으로 승격했음

<그림 2>는 체스 기보 분석을 위한 프로그램의 상태 전이도다. 각각의 직선 동그라미는 하나의 함수를 의미한다. 점선 동그라미는 별도의 함수로 분리하진 않았으나 판단의 근거가 되는 단계를 나타낸다. 선에 표기된 내용은 전이되기 위한 입력이다. 각 상태가 어떤 작업을 처리하는 지는 <표 3>에 나와있다. 이와 관련된 코드는 <리스트 2>에 나와있다.

표 3 각 상태별 설명

함수	역할
Type	명령의 타입을 구분하는 단계다. 입력의 마지막 문자가 0 인 경우는 캐슬링을 나타내기 때문에 Castling 으로 전이한다. 마지막 앞의 문자가 =인 경우는 폰의 승격을 나타내므로 Promotion 으로 전이한다. 1 에서 8 사이의 문자가 있는 경우는 기물이 이동한 목적지이므로 Dest 로 전이한다.
Castling	킹 사이드 캐슬링과 퀸 사이드 캐슬링을 구분하는 단계다.
QSC	퀸 사이드 캐슬링
KSC	킹 사이드 캐슬링
Promotion	폰이 어떤 기물로 승격했는지를 promotion 에 저장한다.
Dest	a8, e4, e5 등의 기물이 이동한 위치를 인식한다.
Capture	x 가 포함된 경우 중간에 상대 기물을 획득한 것이므로 capture 플래그를 true 로 한다.
File	동시에 두 개의 기물이 목표 위치로 이동할 수 있는 경우 어떤 기물이 이동했는지를 알기 위해 이동한 기물의 파일을 표기한다. 파일 표기가 있는 경우 해당 파일을 file 에 저장한다.
Sig	이동한 기물의 종류를 판별한다(폰, 룩, 퀸, …).

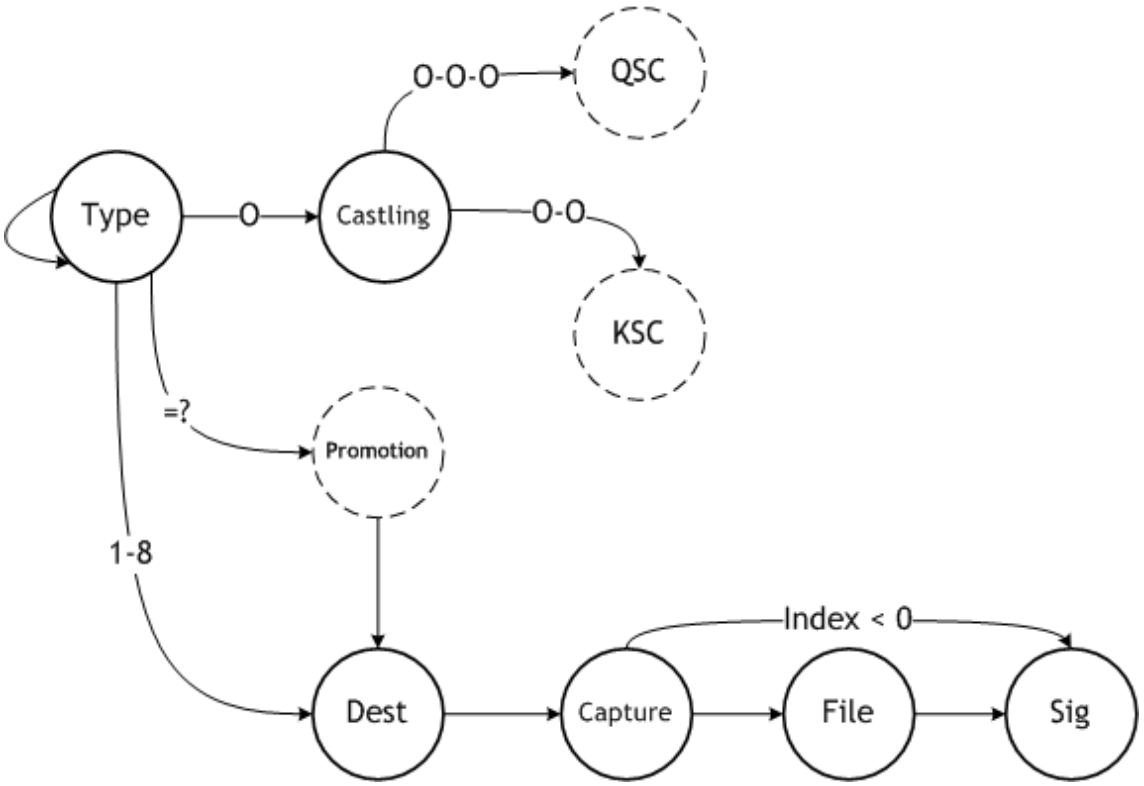


그림 2 체스 기보 분석 상태 전이도

리스트 2 기보 분석과 관련된 함수들

```
CommandParser.prototype.Sig = function(index, cmd)
{
    if(index < 0)
    {
        cmd.sig = 'P';
        return cmd;
    }

    var ch = this.record.charAt(index);
    switch(ch)
    {
    case 'Q': case 'K': case 'N': case 'B': case 'R': case 'P':
        cmd.sig = ch;
        return cmd;
```

```
    }

    return null;
}

CommandParser.prototype.File = function(index, cmd)
{
    if(index < 0)
        return this.Sig(index, cmd);

    var file = this.record.charAt(index);
    if(file < 'a' || file > 'h')
        return this.Sig(index, cmd);

    cmd.file = FileToInt(file);
    return this.Sig(index-1, cmd);
}

CommandParser.prototype.Capture = function(index, cmd)
{
    if(index < 0)
        return this.Sig(index, cmd);

    if(this.record.charAt(index) == 'x')
    {
        cmd.capture = true;
        --index;
    }

    return this.File(index, cmd);
}

CommandParser.prototype.Dest = function(index, cmd)
{
```

```
    if(index < 1)
        return null;

    var rank = this.record.charAt(index);
    if(rank < '1' || rank > '8')
        return null;

    var file = this.record.charAt(index-1);
    if(file < 'a' || file > 'h')
        return null;

    cmd.to = new Square(file + rank);
    index -= 2;

    return this.Capture(index, cmd);
}

CommandParser.prototype.Castling = function(index, cmd)
{
    var ch = this.record.charAt(index);
    if(ch != 'O')
        return null;

    if(index == 0)
    {
        cmd.type = "KSC";
        return cmd;
    }

    cmd.type = "QSC";
    return cmd;
}

CommandParser.prototype.Type = function(index, cmd)
```

```
{
    var ch = this.record.charAt(index);

    switch(ch)
    {
    case '0':
        if(this.record.charAt(index-1) != '-')
            return null;

        return this.Castling(index-2, cmd);

    case 'Q': case 'N': case 'R': case 'B':
        if(this.record.charAt(index-1) != '=')
            return null;

        cmd.promotion = ch;
        index -= 2;
        break;

    case '1': case '2': case '3': case '4': case '5':
    case '6': case '7': case '8':
        break;

    default:
        return this.Type(index-1, cmd);
    }

    return this.Dest(index, cmd);
}
```

3. 기물 클래스

체스에 등장하는 기물들은 모두 이동 규칙이 조금씩 다르기 때문에 기물들을 클래스로 만들어 두는 것이 편리하다. 우리는 각 기물을 공통된 인터페이스를 사용해서 추상화

시켜서 만들 것이다. 우리가 기물에 표시해야 하는 정보는 두 가지다. 어느 진영의 기물인지, 어떤 종류의 기물인지가 그것이다. isWhite 에는 백 진영의 기물인지가 저장된다. isWhite 가 true 라면 백의 기물이고, 아니라면 흑의 기물이다. Signature()는 기물의 종류를 리턴 한다. 앞서 살펴 보았던 기보 표기에 등장하는 기호와 동일하다. CanMove()는 해당 기물의 이동 경로를 추적하는데 사용하는 함수다. 넘어온 위치로 이동할 수 있는지 없는지를 나타낸다. 체스 기보를 표기할 때 단지 이동된 결과면 적기 때문에 어떤 기물이 이동했는지 추적할 필요가 있다.

리스트 3 퀸 클래스

```
function Queen(isWhite)
{
    this.isWhite = isWhite;
}

Queen.prototype.Signature = function()
{
    return "Q";
}

Queen.prototype.CanMove = function(board, from, to, c)
{
    return true;
}
```

<리스트 3>에는 가장 간단한 퀸 클래스가 나타나 있다. 이동 경로가 가장 복잡한 퀸의 CanMove 가 간단한 이유는 퀸은 하나 밖에 없기 때문이다. 우리는 합법적인 기보만 취급할 것이기 때문에 퀸의 이동은 무조건 true 다. 킹도 마찬가지다.

리스트 4 나이트 클래스

```
function Knight(isWhite)
{
    this.isWhite = isWhite;
}
```

```
Knight.prototype.Signature = function()
{
    return "N";
}

Knight.prototype.CanMove = function(board, from, to, c)
{
    var xMoves = new Array(2, 1, -1, -2, -2, -1,  1,  2);
    var yMoves = new Array(1, 2,  2,  1, -1, -2, -2, -1);

    var x, y;

    for(var i=0; i<8; ++i)
    {
        x = from.x + xMoves[i];
        if(x < 1 || x > 8)
            continue;

        y = from.y + yMoves[i];
        if(y < 1 || y > 8)
            continue;

        if(y != to.y || x != to.x)
            continue;

        if(board[y][x] == null)
            return true;

        if(c && board[y][x] != null)
            return true;
    }

    return false;
}
```

<리스트 4>에는 나이트 클래스가 나와있다. 나이트는 다른 기물을 뛰어 넘어 여덟 방향으로 이동할 수 있다. 이 이동 방향이 자신의 위치에서 상대적이기 때문에 차이만 저장해 두었다 계산하면 편리하다. xMoves 와 yMoves 는 이러한 상대적인 차이를 저장하고 있는 배열이다. 새로운 위치가 합법적인지 체크하고 비어있는 경우라면 이동이 가능한 것으로 판단한다.

리스트 5 비숍 클래스

```
function Bishop(isWhite)
{
    this.isWhite = isWhite;
}

Bishop.prototype.Signature = function()
{
    return "B";
}

Bishop.prototype.CanMove = function(board, from, to, c)
{
    if(from.x + from.y == to.x + to.y
        || Math.abs(from.x - from.y) == Math.abs(to.x - to.y))
    {
        return true;
    }

    return false;
}
```

<리스트 5>에 비숍 클래스가 나와있다. 비숍의 경우 대각선 체크만 해주면 된다. 대각선이 같은지는 x, y 좌표의 합과 차가 일치하는지를 검사하면 된다. 두 개의 비숍은 서로 다른 길만 다니기 때문에 다른 체크는 필요가 없다. 폰과 룩에 대한 코드는 이달의 디스크를 참고하도록 하자. 폰의 경우 특별 규칙이 많기 때문에 코드를 이해하기 위해서는 체스 규칙을 잘 알고 있어야 한다. 룩의 경우는 둘 다 동시에 이동할 수 있는 경우가 많기 때문에 사이에 방해 기물이 없는지를 모두 체크해야 한다.

4. ChessBoard 클래스

체스를 즐기기 위해서는 체스 기물과 체스 판이 모두 있어야 한다. 우리가 에뮬레이팅할 가상 세계도 마찬가지다. 앞서 만든 기물 클래스를 담고 있는 체스 판 클래스를 만들어야 한다. <리스트 6>에는 이러한 역할을 하는 ChessBoard 클래스가 나와있다. ChessBoard 클래스는 실제 세계의 체스 보드와 마찬가지로 64 개의 칸에 해당하는 배열이(this.board) 있고, 그 속에는 각각의 기물 클래스가 저장된다.

리스트 6 ChessBoard 클래스

```
function ChessBoard()
{
    this.Initialize();
}

ChessBoard.prototype.Initialize = function()
{
    this.board = new Array(9);
    for(var i=0; i<9; ++i)
    {
        this.board[i] = new Array(9);
        for(var j=0; j<9; ++j)
            this.board[i][j] = null;
    }

    var pieces = new Array();
    pieces = [ { pos:new Point(1,2), obj:new Pawn(true) }
               , { pos:new Point(2,2), obj:new Pawn(true) }
               , { pos:new Point(3,2), obj:new Pawn(true) }
               , { pos:new Point(4,2), obj:new Pawn(true) }
               // … 중략 …
               , { pos:new Point(7,8), obj:new Knight(false) }
               , { pos:new Point(8,8), obj:new Rook(false) } ];

    this.AddPieces(pieces);
```

```
}
```

5. 렌더러 클래스

내부적인 데이터를 HTML 형태로 사용자에게 출력하는 일을 Renderer 클래스가 한다. 이렇게 Renderer 클래스를 분리하는 이유는 화면 출력과 관련된 코드가 다른 코드와 섞여있지 않게 함으로써 유지 보수를 용이하게 하기 위해서다. 추후에 화면 표시 부분을 변경하고 싶다면 이 클래스만 변경하면 된다.

<리스트 7>에 렌더러 클래스의 생성자가 나와있다. DOM 구조를 생성하고 이미지를 로딩하는 것이 주된 일이다. 한 가지 주의해야 할 점은 반드시 프리 로딩 이미지를 사용해야 한다는 점이다. 사용하지 않으면 애니메이션 이미지가 바로 변경되지 않는다.

리스트 7 렌더러 클래스 생성자

```
// 렌더러 클래스 생성자
//
// 파라미터
//      tag - <입력> 노드 ID의 고유 식별자
//      chess - <입력> 렌더링될 체스 클래스
//
// 리턴 값
//      - 없음
function Renderer(tag, chess)
{
    this.tag = tag;

    // obj는 생성된 체스 노드가 최종적으로 추가될 부모 노드다
    var obj = FNode(tag);
    this.Show(false);

    // obj의 자식으로 생성된 체스 노드를 추가한다
    obj.appendChild(this.MakeDOMElements(tag, chess));
```

```
    // 애니메이션에 사용될 이미지 노드
    var action = Node("img");
    action.id = tag + "action";
    action.src = this.GetImageUrl(null);
    action.style.visibility = "hidden";
    action.style.position = "absolute";
    obj.appendChild(action);

    // 이미지를 미리 로드하지 않을 경우 애니메이션 이미지가 바로 변경되지 않는다
    this.preloads = new Array(new Image(),new Image(),new Image(),new Image(),new
Image()

    // … 중략 …

    this.preloads[0].src = "http://www.jiniya.net/script/chess/img/pl.png";
    this.preloads[1].src = "http://www.jiniya.net/script/chess/img/ql.png";

    // … 중략 …
}
```

<리스트 8>에는 DOM 구조를 생성하는 함수인 MakeDOMElement 가 나와있다. 체스 보드 테이블을 생성하고 이미지 노드를 만들고, 버튼을 추가하는 것이 주된 코드다. 노드를 생성하고 서로 연결하고, 속성을 지정하는 방법을 자세히 살펴 보도록 하자.

리스트 8 체스 보드 DOM 구조를 생성하는 함수

```
// 체스 보드의 DOM 구조를 생성한다
Renderer.prototype.MakeDOMElements = function(tag, chess)
{
    // 테이블을 생성한다
    var board = Node("table");
    board.className = "chess_board";
    board.cellSpacing = "0px";
    board.cellPadding = "0px";
```

```
    var tr, td, img;
    var last = 0;
    var squareStyle = new Array("chess_squareb", "chess_squarew");
    var rank;

    // 64 개의 체스 판을 생성한다
    for(var i=0; i<8; ++i)
    {
        rank = (8-i).toString();

        tr = board.insertRow(i);
        td = tr.insertCell(0);
        td.className = "chess_rank";
        td.id = tag + rank;
        td.appendChild(TNode(rank));

        for(var j=1; j<9; ++j)
        {
            td = tr.insertCell(j);
            td.className = squareStyle[last];

            // 체스 기물을 표시할 이미지 노드를 만든다
            img = Node("img");
            img.src = this.GetImageUrl(chess.board.board[8-i][j]);
            img.id = tag + j.toString() + rank;
            td.appendChild(img);

            last = 1-last;
        }

        last = 1-last;
    }

    // 파일을 표시하는 줄을 생성한다
```

```
    var files = " abcdefgh";
    tr = board.insertRow(8);
    td = tr.insertCell(0);
    for(var j=1; j<9; ++j)
    {
        td = tr.insertCell(j);
        td.className = "chess_file";
        td.appendChild(TNode(files.charAt(j)));
    }

    // 각종 버튼을 추가한다
    var button;
    var center = Node("center");

    button = Node("input");
    button.type = "button";
    button.value = "처음";
    button.id = tag + "first";
    button.onclick = function() { chess.MoveFirst(); };
    center.appendChild(button);

    // … 중략 …

    tr = board.insertRow(9);
    td = tr.insertCell(0);
    td.colSpan = "9";
    td.appendChild(center);

    return board;
}
```

우리가 데이터와 뷰를 분리함으로써 얻는 가장 큰 효과를 보여주는 코드가 <리스트 9>에 나와있다. <리스트 9>에 나온 Revert 함수가 하는 일은 화면 상의 흑과 백의 위치를 변경하는 함수다.

리스트 9 흑/백 위치를 변경하는 함수

```
// 흑/백 위치를 변경하는 함수
Renderer.prototype.Revert = function()
{
    var id;
    var img = new Array(9);
    var rankNodes = new Array(9);
    for(var i=1; i<9; ++i)
    {
        img[i] = new Array(9);
        for(var j=1; j<9; ++j)
        {
            img[i][j] = this.GetSquareNode(new Point(j, i));
        }

        rankNodes[i] = FNode(this.tag + i.toString());
    }

    for(var i=1; i<9; ++i)
    {
        for(var j=1; j<9; ++j)
        {
            img[i][j].id = this.tag + j.toString() + (8-i+1).toString();
        }

        rankNodes[i].id = this.tag + (8-i+1).toString();
        rankNodes[i].childNodes[0].nodeValue = (8-i+1).toString();
    }
}
```

6. 체스 클래스

결국 우리가 이제까지 작성한 모든 클래스는 기보 내용을 플레이하기 위함이다. 이러한 모든 클래스를 포함하고 관리하는 것이 Chess 클래스다. 지면 관계상 Chess 클래스의 가장 중요한 부분인 다음 기보로 이동하는 MoveNext 함수만 살펴본다.

<리스트 10>에 MoveNext 함수의 코드가 나와있다. this.current 는 현재 플레이되는 기보의 번호를 나타낸다. this.records 는 전체 기보를 문자열 형태로 가지고 있는 배열이다. 가장 먼저 하는 일은 현재 기보 내용을 분석하는 것이다. 분석된 내용은 cmd 에 담긴다.
다음으로 할 일은 cmd 구조체를 기반으로 실제로 이동한 기물을 찾는 일이다. 기물을 찾는 방법은 간단하다. 모든 기물을 검사해서 이동할 수 있는지를 체크한다. 이동 가능 하다면 해당 기물을 움직인다. 캐슬링과 프로모션에 대한 특별 규칙은 별도로 처리해 준다.

리스트 10 MoveNext 함수

```
// 다음으로 넘어 간다.
// 파라미터
//      ani - <입력> 애니메이션 여부
//
// 리턴 값
//      - 마지막인 경우 false, 아닌 경우 true
Chess.prototype.MoveNext = function(ani)
{
    // 마지막까지 재생한 경우 false 를 리턴 한다.
    if(this.current >= this.records.length)
        return false;

    // 커맨드를 분석 한다.
    var cmd = this.cp.Run(this.records[this.current]);
    var board = this.board;
    var renderer = this.renderer;
    var mov = new Move(null, null, null, null, null, null);
    var complete = false;

    // 타입이 없는 경우, 일반적인 이동이다.
```

```
    if(cmd.type == null)
    {
        // 움직일 수 있는 기물을 찾는다.
        for(var i=1; i<9 && !complete; ++i)
        {
            for(var j=1; j<9 && !complete; ++j)
            {
                obj = board.board[i][j];

                // 빈 칸인 경우 다음 칸으로 이동
                if(obj == null)
                    continue;

                // 움직일 차례인 사람의 기물이 아닌 경우 다음 칸으로 이동
                if(obj.isWhite != this.isWhiteTurn)
                    continue;

                // 기물의 종류가 다른 경우 다음 칸으로 이동
                if(obj.Signature() != cmd.sig)
                    continue;

                // 해당 위치의 기물이 움직일 수 없는 위치인 경우 다음 칸으로 이동
                if(!obj.CanMove(board.board, new Point(j,i), cmd.to, cmd.capture))
                    continue;

                // 기보 내용의 파일(세로 줄)가 일치하지 않는 경우 다음 칸으로 이동
                if(cmd.file > 0 && j != cmd.file)
                    continue;

                // 이동 위치 포인트 생성
                mov.from = new Point(j,i);
                mov.to = cmd.to;

                // 상대편 기물을 잡아 먹은 경우 잡힌 기물 저장
```

```
                if(cmd.capture)
                    mov.capture = board.board[mov.to.y][mov.to.x];

                // 보드상에서 기물 이동
                board.Move(mov.from, mov.to);

                // 폰의 승격인 경우
                if(cmd.promotion != null)
                {
                    // 새로 보드에 추가될 기물 생성
                    mov.promotion = GetPiece(cmd.promotion, this.isWhiteTurn);

                    // 해당 기물을 보드에 추가
                    board.AddPiece(mov.to, mov.promotion);

                    if(ani)
                        renderer.Move(mov.from, mov.to, ani, mov.promotion);
                    else
                    {
                        // 애니메이션 기능을 사용하지 않는 경우 기존 기물 삭제후 추가
                        renderer.DeletePiece(mov.from);
                        renderer.AddPiece(mov.to, mov.promotion);
                    }
                }
                // 승격이 아닌 경우 기물 이동
                else
                    renderer.Move(mov.from, mov.to, ani, null);

                // 작업 완료
                complete = true;
                break;
            }
        }
    }
```

```
    else
    {
        // 진영에 따른 킹의 기본 랭크(가로 줄)
        var rank = this.isWhiteTurn ? 1 : 8;

        // 킹 사이드 캐슬링인 경우
        if(cmd.type == "KSC")
        {
            // 두 개의 기물을 이동 시킨다.
            board.Move(new Point(5, rank), new Point(7, rank));
            board.Move(new Point(8, rank), new Point(6, rank));

            // 동시에 하나만 애니메이션 할 수 있기 때문에 룩의 이동은 애니메이션
시키지 않는다.
            renderer.Move(new Point(5, rank), new Point(7, rank), ani);
            renderer.Move(new Point(8, rank), new Point(6, rank), false);

        }
        // 퀸 사이드 캐슬링인 경우
        else if(cmd.type == "QSC")
        {
            board.Move(new Point(5, rank), new Point(3, rank));
            board.Move(new Point(1, rank), new Point(4, rank));

            renderer.Move(new Point(5, rank), new Point(3, rank), ani);
            renderer.Move(new Point(1, rank), new Point(4, rank), false);

        }
        mov.type = cmd.type;
    }

    // 이동 경로에 현재 움직임을 저장한다.
    this.moveList[this.current] = mov;
```

```
    // 다음 저장 위치로 이동
    this.current += 1;

    // 턴 변경
    this.isWhiteTurn = !this.isWhiteTurn;
    return true;
}
```

7. 체스 기보 뷰어

이제까지 작성한 체스 기보 뷰어를 테스트하는 방법은 간단하다. <리스트 11>과 같은 간단한 HTML 페이지를 작성해서 브라우저에서 불러오면 된다. <화면 1>은 FireFox 에서 페이지를 불러들여서 실행한 화면이다. Internet Explorer 6.0 의 경우 png 이미지를 정상적으로 지원하지 않아 그림의 배경이 깨진 이미지가 표시된다.

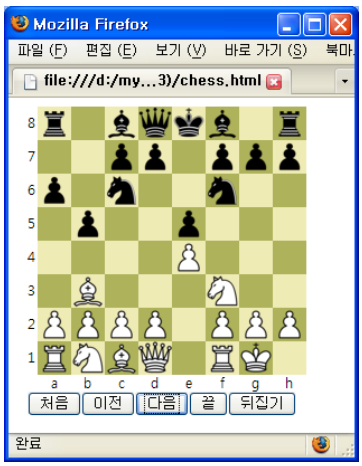
리스트 11 체스 기보 뷰어 테스트 html 페이지

```
<HTML>
    <HEAD>
        <link href="chess.css" rel="stylesheet" type="text/css">
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    </HEAD>
    <BODY>
        <script type="text/javascript" src="chess.js"></script>

        <div id='chess1394' style='display:none;'></div>
        <script>
            var chess = new Chess('chess1394'
                                  , '1.e4 e5 2.Nf3 Nc6 3.Bb5 a6 4.Ba4 b5 5.Bb3
'
                                  + 'Nf6 6.O-O Nxe4 7.Re1 d5 8.a4 b4 9.d3 Nc5 '
                                  + '10.Nxe5 Ne7 11.Qf3 f6 12.Qh5 g6 13.Nxg6 '
                                  + 'hxg6 14.Qxh8 Nxb3 15.cxb3 Qd6 16.Bh6 Kd7 '
                                  + '17.Bxf8 Bb7 18.Qg7 Re8 19.Nd2 c5 20.Rad1 '
                                  + 'a5 21.Nc4 dxc4 22.dxc4 Qxd1 23.Rxd1 Kc8
```

```
24.Bxe7');

            chess.Show(true);
        </script>
    </BODY>
</HTML>
```



화면 1 FireFox 에서 실행한 체스 뷰어

8. 도전 과제

이번 달 코드를 바탕으로 자바스크립트로 온라인 체스 게임을 작성해 보자. 엔진을 포팅하는 것은 쉽지 않은 일이기 때문에 단순하게 사람끼리 둘 수 있도록 구현해 본다. Chess 클래스를 수정해서 MoveNext, MovePrev 등의 함수를 제거하고 실제 플레이를 위한 함수를 추가해야 한다. 각 기물 클래스를 수정해서 특정 기물을 선택했을 때 이동 가능한 경로를 보여주도록 만들어 본다.

9. 참고자료

참고자료 1. Dave Crane 외, <<AJAX 인 액션>> 에이콘

참고자료 2. ECMAScript 표준안(ECMA-262 3 차 개정판) - http://www.ecma-international.org/publications/standards/Ecma-262.htm

참고자료 3. AJAX 체스 엔진 - http://ajax.phpmagazine.net/2006/04/first_implementation_of_a_ches.html