DevOn RichUI

최적화 가이드

Ver. 1.1

관리부서 : 모바일/UI기술그룹

개 정 이 력

버전	작성일	변경 내용	작성자	승인자
1.0	2013.07.11	최초 작성	홍준희	김순호
1.1	2013.12.02	슬라이드 II-2 제거 (RichUI의 기능 개선으로 교육의 필요성 없음)	문혁찬	장문수

목 차

1. ActiveX 기반과 웹표준 솔루션의 성능

➢ **ActiveX 기반의 솔루션과 웹표준 솔루션의 성능**

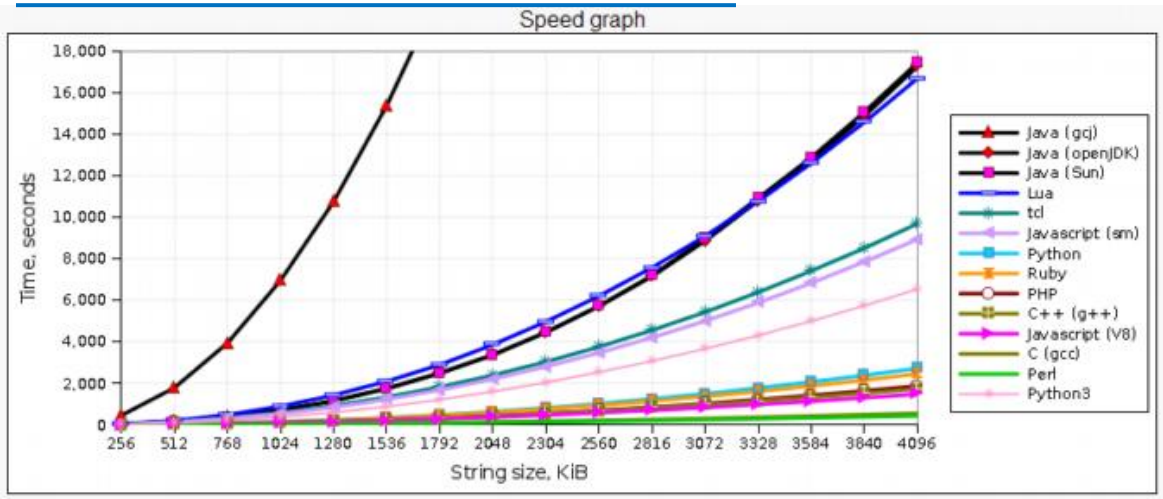
▪**ActiveX 기반의 솔루션(Native 언어)과 웹표준 솔루션(Javascript)의 구현 언어의 성능 차이를 고려해야 함**

Javascript의 성능 비교

This table shows number of seconds taken to complete every testing stage.

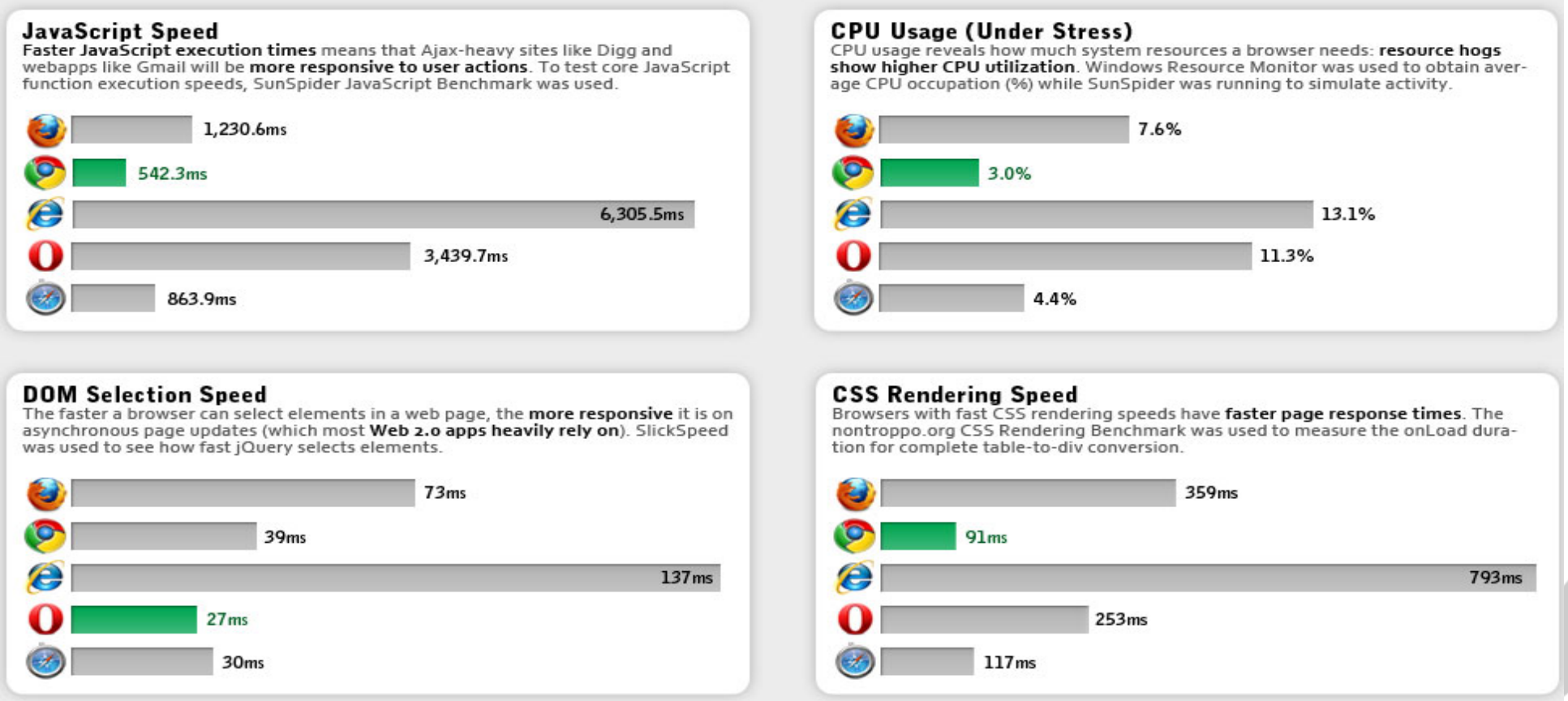
Line size Kb	Perl5	PHP	Ruby	Python	C++ (g++)	C (gcc)	Javascript (V8)	Javascript (sm)	Python3	tcl	Lua	Java (openJDK)	Java (Sun)	Java (gcj)
256	2	6	7	7	7	2	3	30	17	33	49	39	38	451
512	7	23	29	32	26	8	21	131	81	141	203	162	157	1783
768	16	54	75	78	60	19	51	300	201	324	480	381	371	3937
1024	27	96	141	144	107	34	91	535	373	583	886	711	696	6952
1280	43	153	225	232	167	53	144	842	598	921	1423	1161	1145	10744
1536	62	227	328	342	242	76	208	1220	877	1334	2090	1751	1739	15372
1792	84	318	452	476	329	104	283	1672	1211	1823	2886	2489	2478	20819
2048	109	424	597	634	431	136	370	2203	1598	2387	3856	3370	3358	27132

Javascript의 성능 비교 그래프



2. 브라우저의 성능 차이

➢ IE와 타 브라우저의 심각한 성능 차이



브라우저를 바꿀 수 있다면 사용자 체감 성능이 많이 높아진다.

3. 일반적인 개발자의 오해

◆ 그냥 ActiveX가 빠르고 편한데...?
- ➢ 전세계적으로 현재 ActiveX는 빠르게 사라지는 추세이다. (향후 3년 후를 고려해보라)
- ➢ 플랫폼(OS, 브라우저...)이 점차 다양화 되고 있다.
- ➢ ActiveX에 맞춰 개발한 시스템을 글로벌 시스템으로 전환할 경우 추가 개발해야 한다.

◆ 우리 프로젝트는 IE 8이 표준이다?
- ➢ 표준이 IE 8일 뿐이지 IE 8만 지원하라는 시스템이 아니다.

◆ 다른 ActiveX(파일 업로드, 리포팅 툴 등...) 때문에 IE만 사용할 수 있다?
- ➢ 다른 ActiveX 솔루션들도 최근 Cross Browser를 지원한다.(크롬, 사파리, 파이어폭스 등...)

◆개발하기 바쁜데 다른 브라우저는 모르겠다?
- ➢ IE에서 개발한 후 타 브라우져를 지원하는 것보다 크롬과 같은 브라우저로 개발한 후 IE로 전환하여 테스트 하면 개발 생산성이 더 좋아진다.

◆ 개발한 후 성능을 개선해야지?
- ➢ 최근 UI는 점점 복잡도가 높으므로 개발을 완료한 후 성능을 개선하고자 한다면 제약사항이 많아 개선의 여지가 적어진다.

1. 동기식(sync) Ajax 사용시의 성능 저하 #1

◆ sync방식으로 Ajax 처리시 IE의 성능저하

비동기(async)방식이 아닌 동기(sync)방식으로 요청 시 IE는 서버로부터 응답을 받을 때 까지 모든 프로세스가 점유 되어 대기 하게 된다.
결국 병행 처리되어야 할 모든 작업들이 순차처리가 되어 느려지는 것이다.
특히 네트워크 환경이 좋지 않은 환경, 해외나 불안정한 WIFI 환경(ping loss)이라면 더욱 심각하다.

방안 1 async를 사용하라!

Callback 사용

- async로 호출 후 callback 메소드로 처리

```
Rui.ajax({
    url: 'test.ajax',
    success: function(e) {
        //서버 처리가 완료되면 call back 메소드 호출
    }
});
```

Event 처리

- async로 호출 후 event로 처리

```
dataSet.on('load', function(e) {
    // 서버 처리가 완료되면 이벤트 호출
    .....
});
```

1. 동기식(sync) Ajax 사용시의 성능 저하 #2

◆가능하다면 서버 측에서 데이터를 준비하라.

화면에 필요한 고정된 값을 Ajax를 이용하여 서버로부터 가져오는 것 보다 가능하다면 서버에서 HTML을 만들어낼 때 필요한 데이터를 HTML에 포함시키는 것이 성능에 좋다.

방안 2 가급적 서버 측 코딩을 사용하라!

JSP 코딩 예

- JSP 코딩에 DataSet에 로드될 값을 미리 포함시킨다.

```
<%
    String data = "{ records: [ { code: 'H', value: 'Hello' }, { code: 'W', value: 'World' } ] }";
%>
dataSet.loadData(<%=data%>);
```

2. 불필요한 DataSet 보다 Rui.ajax 메소드를 사용 #1

◆DataSet을 단순조회용(서버 처리)으로 사용 하게 되면 DataSet의 데이터 관리 소스로 인하여 성능이 저하되고 한번 생성된 DataSet 은 페이지가 갱신되기 전까지 남아 있으므로 불필요한 메모리를 증가시킨다. (DataSet은 메모리 Cost가 높음)

화면에 DataSet이 15개 이상 생성(Combo 포함)될 경우 화면 최적화를 고려해야 한다.

문제 불필요한 DataSet 생성으로 성능이 저하되고 메모리 사용량이 증가

DataSet 사용시

- 콤포넌트와 연동되지 않는 단순한 row의 값을 조회/처리하기 위해 DataSet을 생성했다. (성능 저하/메모리 증가)

```
var dataSet = new Rui.data.LJsonDataSet({
    id: 'dataSet',
    fields: [
        { id: 'col1' }
    ]
});

dataSet.on('load', function(e) {
    dataSet.getNameValue(0, 'col1');
});

dataSet.load({
    url: '/test.ajax'
});
```

처리 완료 후에도 DataSet은 메모리에 남아 있다.

2. 불필요한 DataSet 보다 Rui.ajax 메소드를 사용 #2

◆Rui.ajax는 서버에 데이터를 요청하는 처리만 포함되므로 가볍고 빠르다. (Rui.ajax은 Cost가 낮음)

콤포넌트와 연결되지 않은 단건 row의 데이터는 Rui.ajax를 사용해야 한다.

방안 서버처리를 위한 단순조회용으로는 가급적 Rui.ajax을 권장한다.

Rui.ajax 사용시

- DataSet 없이 Rui.ajax로 서버 처리를 했다. (성능 향상/메모리는 처리 후 자동 삭제됨.)

```
Rui.ajax({
    url: '/test.ajax',
    success: function(e) {
        var obj = eval(e.responseText);
        obj.name;
        obj.jobCode;
    }
});
```

처리 완료 후에는 메모리가 초기화 된다.

3. Grid의 성능 저하 메소드들

◆Grid에서 Height, Width, ColumnModel의 규칙이 바뀔 경우 render가 반복 수행되어 성능 느려진다.

Grid의 전체 랜더링은 브라우저에서 cost가 높은 작업이다.

High cost 메소드는?

- setHeight(int)
- setWidth(int)
-setHidden(true or false)

그리드의 width나 height가 자주 바뀌거나 컬럼을 보였다 안 보였다 할 경우 Grid는 전체 랜더링 되므로 성능이 저하된다.

-autoWidth

table안의 Grid의 경우 부모의 width가 계속 바껴 반복 수행 될 수 있음.

또한 위 메소드들은 화면 복잡도에 따라 사용 시 성능이 심하게 저하될 가능성이 있다.

➔ IE 6,7,8에서 성능 저하가 심하게 발생한다.

그렇다면?

setHeight, setWidth의 경우 불필요하게 사용하지 않길 권장한다.

autoWidth를 감싸고 있는 부모 DOM의 max-width를 적용하여 반복 수행을 최소화해야 한다.

setHidden 메소드를 많이 사용하기 보다 Grid를 두 개 생성하여 show/hide로 처리한다.
(hidden 컬럼이 4개 이상일 경우)

➔ 개발 시 필요하다면, 시스템 사용자들의 성능이 낮은 PC를 고려해야 한다.

4. 단순 출력 Grid에서의 date 출력 방법

◆서버에서 받은 date 값은 renderer를 통해서 변환하면 성능이 느리다.

◆단순 조회용으로 출력하는 경우 서버에서 출력할 문자열을 생성하여 출력한다.

Grid의 renderer는 브라우저에서 cost가 높은 작업이다.

문제 date 객체로 처리하여 성능이 낮다.

Date 객체 사용시

```
var dataSet = new Rui.data.LJsonData({
    fields: [ { id: 'col1', type: 'date' } ]
});

var columnModel = new
    Rui.ui.grid.LColumnModel({
    { field: 'col1', renderer: 'date' }
});
```

renderer를 사용하여 출력했다.

방안 renderer 없이 처리하여 성능이 높다.

문자열 출력시

```
var dataSet = new Rui.data.LJsonData({
    fields: [ { id: 'col1' } ]
});

var columnModel = new
    Rui.ui.grid.LColumnModel({
    { field: 'col1' }
});
```

renderer를 사용하지 않고 출력했다.

5. Tab 안에 Grid 및 콤포넌트 처리 방법

◆Tab 안에 보이지 않는 Grid 및 타 콤포넌트들을 렌더링하면 성능 느리다.

사용하지 않는 Grid를 렌더링하면 DataSet 변경에 따른 HTML이 변경되므로 브라우저가 불필요한 작업을 많이 하게 된다.

방안 Tab이 처음 선택될 경우 Grid를 랜더링한다.

isFirst 속성 사용

```
var grid1 = new Rui.ui.grid.LGridPanel({
    ……
});
var grid2 = new Rui.ui.grid.LGridPanel({
    ……
});
var tabview = new Rui.ui.tab.LTabView({
    ……
});
tabview.render('tab');

tabview.on('activeIndexChange', function(e){
    if(e.activeIndex == 1 && e.isFirst) {
        grid1.render('grid1');
    } else if(e.activeIndex == 1 && e.isFirst) {
        grid2.render('grid1');
    }
});
```

Tab이 처음 호출되면 isFirst가 true가 되고 이때 Grid를 render 한다.

6. Tree와 TreeGrid의 장단점

◆Tree를 사용할 경우 대량 건 데이터를 로딩하면 성능이 느리다.

TreeGrid는 Buffer 개념을 적용하여 Grid에 보이는 row만 생성하므로
메모리 점유율도 낮고 성능도 높다.

Tree의 편리성?

Tree의 장점
- 데이터의 로딩이 쉽다.
- DataSet의 데이터 변경이 쉽다.
- 계층별 동적 로딩을 통해 보이는 영역만 로딩할 수 있다.
- 정렬이 쉽다. (데이터 순서 컬럼 값에 따라 자동 정렬)

Tree의 단점
- 대량 건 데이터 처리시 성능 저하 발생
- 한 노드에 두 개의 컬럼을 출력할 수 없다.

➔ Leaf 노드 기준, 1000건 이하일 경우 Tree를 사용한다.

TreeGrid의 성능?

TreeGrid의 장점
- 성능이 빠르다.
- 대량 건 데이터 처리도 가능하다.
- 기본적으로 Grid이므로 여러 컬럼을 나열하여 출력되고 Grid의 장점을 가질 수 있다.

TreeGrid의 단점
- 데이터의 로딩 시 서버에서 데이터를 정렬해서 처리해야 한다.
- DataSet의 데이터 변경이 어렵다.
  정렬이 어렵다. (DataSet의 데이터 순서를 직접 변경)

➔ Leaf 노드 기준, 1000건 이상일 경우 TreeGrid를 사용한다.

7. 반복적인 innerHTML 사용 자제

◆IE 6/7/8에서는 innerHTML을 반복 호출하는 경우 성능이 느리다.

◆사용하려면 문자열을 한번에 만들어서 처리해야 한다.

innerHTML은 IE 브라우저에서 반복적으로 렌더링을 수행하게 되어 성능이 느려진다.

문제 innerHTML을 반복 수행하여 성능이 낮다.

innerHTML 반복 수행시

```
var dom = document.getElementById('aaa');
dom.innnerHTML = '<a>test1</a>';
dom.innnerHTML += '<a>test2</a>';
dom.innnerHTML += '<a>test2</a>';
```

매번 렌더링

방안 innerHTML을 한번에 수행하여 성능이 높다.

innerHTML을 한번에 수행시

```
var dom = document.getElementById('aaa');
var html = '<a>test1</a>';
html += '<a>test2</a>';
html += '<a>test2</a>';
dom.innerHTML = html;
```

한번만 렌더링

◆두 개 이상의 프레임워크를 사용 시 DOM에 이벤트를 탑재한 후 해당 DOM보다 상위 DOM을 다른 프레임워크로 제거하면 memory leak 발생한다.

브라우저는 하위 DOM을 삭제할 때 이를 다른 프레임워크에서 사용중으로 인식하므로 브라우저의 Garbage Collection 대상에서 제외된다.

문제 이벤트를 탑재한 후 remove 메소드를 사용하여 DOM을 제거했다.

jquery의 remove() 사용 시

* DOM remove시 memory leak 예

```
<div id="parent"><div id="child"></div></div>

Rui.get('child').on('click', function(e) {     .....     });

.....

$('#parent').remove();
```

Rui로 탑재된 child가 jquery로 parent에 대해서 remove 메소드를 통해 삭제되면 Rui에 탑재된 child 의 function 에 대 해 memory leak 발생한다.

◆DOM을 삭제하기 전에는 DOM에 탑재했던 이벤트를 제거하라.

DOM을 제거하기 전에 이벤트를 삭제하여 Garbage Collection 대상으로 포함시킨다.

방안 탑재된 이벤트를 제거한 후 상위 DOM을 제거했다.

이벤트 제거

DOM remove시 메모리릭 예

```
<div id="parent"><div id="child"></div></div>

var onClick = function(e) {     .....      };
Rui.get('child').on('click', onClick);

.....
Rui.get('child').unOnAll();
$('#parent').remove();
```

unOnAll 메소드를 통해 해당 이벤트들을 모두 제거하고 jquery로 child를 제거했다.

2. 지원되지 않는 이벤트에 대한 memory leak

◆이벤트를 사용할 경우 브라우저가 지원하지 않는 DOM 이벤트를 사용하면 memory leak이 발생한다.

브라우저 기본 DOM 이벤트가 아닌 경우 memory leak이 발생한다.

방안 **IE와 webkit 계열 브라우저의 mousewheel 이벤트는 서로 다르다.**

이벤트를 브라우저 별로 체크

IE에서 조건 없이 DOMMouseScroll를 addListener를 통해 추가할 경우 memory leak이 발생한다.

```
DU.util.LEvent.removeListener(this.scrollerEl.dom, "mousewheel", this.onWheel);
DU.util.LEvent.addListener(this.scrollerEl.dom, "mousewheel", this.onWheel, this, true);

//firefox only
if(DU.browser.mozilla) {
    DU.util.LEvent.removeListener(this.scrollerEl.dom, "DOMMouseScroll", this.onWheel);
    DU.util.LEvent.addListener(this.scrollerEl.dom, "DOMMouseScroll", this.onWheel, this, true);
}
```

DOMMouseScroll 이벤트 같은 경우 ie에서는 인식되지 않는 이벤트로 memory leak이 발생하므로 반드시 조건을 부여해야 한다.

※ 브라우저가 지원하는 기본 DOM 이벤트만 사용할 경우는 memory leak이 발생하지 않는다.

3. documentFragment 사용 시 놓치기 쉬운 memory leak

◆Document에 생성된 DOM을 부득이 documentFragment로 이동하여야 할 경우 memory leak의 발생할 수 있다.

documentFragment에 들어 있는 DOM객체는 삭제한 후 innerHTML 처리해야 memory leak을 방지할 수 있다.

방안 documentFragment에 넣었던 DOM은 document가 unload될 때 반드시 document에 다시 append되어 있어야 한다.

documentFragment 사용시

```
var foo = document.createElement('input');
foo.onclick = function(){ ...        };
document.body.appendChild(foo);
document.createDocumentFragment().appendChild(foo);
document.body.appendChild(foo);
```

body.appendChild 구문이 존재할 경우 leak이 발생 안 한다.

4. Iframe의 memory leak이 발생할 가능성

◆iframe 사용 중 iframe의 상위 DOM을 삭제할 경우 iframe 내부 컨텐츠에 의해 memory leak 이 발생할 가능성 높다.

iframe 내부의 컨텐츠에는 leak 가능성이 있는 소스가 포함되어 있을 수 있다.

방안 Iframe은 삭제하기 전에 src를 about:blank로 clear해야 하며, iframe 내부에 iframe이 있는 경우에 하위 iframe들도 같이 clear 해야 한다.

iframe 사용시

```
<div id="parent">
<iframe id="childFrame" src="http://...."></iframe>
</div>
//DU.get('childFrame').dom.src = 'about:blank';
DU.get('parent').remove();
```

iframe의 src url의 페이지에 memory leak 가능성이 존재할 경우

※ 이 증상은 프로젝트에서 발견되었으며, 메뉴 이동에 따라 body영역의 iframe 페이지를 생성, 삭제 하는 중 발생되었다.
Iframe의 내부 컨텐츠에서 leak 가능성이 있었던 것으로 판단되며, iframe 삭제 전에 iframe의 src를 about:blank로 clear하여 해결함.
또한 iframe 내 컨텐츠에도 iframe이 존재할 수 있으며 이 또한 leak 가능성을 예상하여, 내부 컨텐츠의 iframe들도 clear할 수 있도록 조치하길 권장한다.

5. window 팝업의 LDataSet 공유에 따른 memory leak

◆window 팝업에서 returnValue를 통해 DataSet을 공유하거나 opener/top을 통해서 접근하는 DataSet 사용시 memory leak 발생 한다.

window창과 팝업간의 레퍼런스가 되는 Object객체를 전달하면 window창에서 참조되는 모든 스크립트들은 브라우저에서 memory leak 발생한다.

문제 DataSet 전체를 부모창의 function으로 호출하면서 인수로 전달했다.

window 팝업의 memory leak

```
opener.fn(dataSet);
```

← opener Function에 데이터셋 전달 시 memory leak

해결 새로운 객체를 생성하여 부모창의 function으로 호출하면서 인수로 전달했다.

window 팝업의 memory leak 방지 방법

```
Var newObject = dataSet.getAt(0).getValues();
opener.fn(newObject);
```

← 새로운 객체를 생성하여 참조 정보를 끊는다.

6. LDataSet/LDataSetManager의 loadCache

◆setTimeout/setInterval을 통한 반복적인 데이터 갱신 시 IE의 경우 json(eval) 방식으로 서버에서 데이터를 로딩하면 소량의 memory leak이 발생한다.

◆loadCache 생성자 속성을 사용할 경우 서버에서 데이터가 같으면 같은 레코드로 처리하여 성능 및 memory leak 줄임 수 있다.

IE 브라우저는 eval 메소드에서 소량의 memory leak이 발생한다.

방안 초당/분당 서버에 호출 시에는 loadCache 속성을 적용한다.

DataSet 사용시

```
var dataSet = new Rui.data.LJsonDataSet({
    loadCache: true,
    fields: [
        { id: 'col1' }
    ]
});

dataSet.load({
    url: 'test.ajax'
});
```

loadCache 생성자 속성 적용

DataSetManager 사용시

```
var dm = new Rui.data.LDataSetManager({
    loadCache: true
});

dm.loadDataSet({
    dataSets: [ dataSet ],
    url: 'test.ajax'
});
```

loadCache 생성자 속성 적용

7. Memory leak 참고자료

참고자료

참고 자료)
blog.stchur.com/2007/05/16/ie-innerhtml-memory-leak/

기타 자료

http://www.ibm.com/developerworks/web/library/wa-memleak/
http://www.codeproject.com/Articles/12231/Memory-Leakage-in-Internet-Explorer-revisited