メソッド、関数	説　明	引数（→返値）	サンプル	その他
Prototype.js ver1.6.0.3				**UPDATE:2009/07/20**
Utility Methods				
$(id | element)	getElementById( )の代わり 複数設定可（	id:id element:要素（拡張形式にする） →要素	$('ida') →要素 $('ida', 'idb') →要素の配列	
$$(selector)	$()のCCS3セレクタ、要素名版 常に配列で返す(NodeList)	selector:セレクタ、要素名文字列 →要素の配列	$$('div') $$('.class') $$('#coment li')	CSS3表記対応
$w(str)	配列定義の簡易版 空白で分けた文字列の配列化 文字列のみ	str:配列初期値 →配列	$w('aaa bbb ccc') →['aaa', 'bbb', 'ccc']	
$A(arg)	拡張Array化 =toArray() Array.from(a|s) 文字列の配列化	arg: 配列→Prototype拡張 配列互換(argumentsなど)→配列化 文字列→配列化 オブジェクトの配列化→プロパティ値の配列（文字列のみ。数値は無視） →配列	$A(arguments) $A('abcd') →['a', 'b', 'c', 'd'] $A({a:'aa',b:'bb',c:10}) →[aa, bb]	each()を使用するのに必要
$F(fieldelt)	フォームフィールド内容の取得 $( ).valueと同じ チェック系もvalueなので注意	fieldelt:ID値 →value値		マルチリストボックスは配列を返す そのほかは単一 チェック系はチェック無しでnullを返す リストボックスでvalue無しはテキストを返す
$H([obj])	ハッシュ作成（ハッシュ化） オブジェクトのハッシュ拡張	obj:オブジェクト		new Hash(obj);
$R(start, end)	範囲作成	start:開始値 end:終了値	$A($R(1,3)) →[1,2,3] $A($R(a,c)) →['a','b','c'] $R(1,5).each(xxx)	
RegExp.escape(str)	文字列を正規表現用にエスケープする	str:文字列	RegExp.escape('hi. [1] aaa?') →'hi\. \[1\] aaa\?'	

Object				
Object.inspect(obj)	オブジェクトの種類を詳細に表示 似)toString() null,undefineなどを表示	オブジェクト →string		
Object.isArray(obj)	配列かチェック	オブジェクト →Boolean		コンストラクタ関数がArrayのコンストラクタ関数と同じかチェック
Object.isElement(obj)	nodeTypeプロパティーのチェック 要素ノードか（1）	オブジェクト →Boolean		
Object.isFunction(obj)	関数かのチェック typeof	オブジェクト →Boolean		
Object.isHash(obj)	ハッシュかのチェック instanceof	オブジェクト →Boolean		
Object.isNumber(obj)	数値かのチェック typeof	オブジェクト →Boolean		
Object.isString(obj)	文字列化のチェック typeof	オブジェクト →Boolean		
Object.isUndefine(obj)	undefineかのチェック typeof	オブジェクト →Boolean		
Object.keys(obj)	オブジェクトのプロパティー名の配列作成	オブジェクト →配列	var list = { id:1, name:”take”); Object.keys(list); →[id, name]	ハッシュ（$H()）のメソッドと別
Object.values(obj)	プロパティー値の配列作成	オブジェクト →配列	var list = { id:1, name:”take”}; Object.keys(list); →['1', 'take']	ハッシュ（$H()）のメソッドと別
Object.toQueryString(obj)	オブジェクトをURLクエリーに変換	オブジェクト →String	{key1 : value1, key2 : value2} →’key1=value1&key2=value2’	
Object.clone(obj)	オブジェクトの複製	複製するオブジェクト →複製したオブジェクト		参照（配列、オブジェクト）は複製元と共有される
Object.extend(dest, source)	オブジェクトの複製 clone()の元	dest:複製先（書き換わる） source:複製元 →dest destは双方同じオブジェクト	var myObj1 = { id:123, name:"KF" }; var myObj2 = { age:37 }; var newObj = Object.extend(myObj1, myObj2);	Element.extend()とは別

Object.toHTML(obj)				
Funciton				
obj.func.bind(obj, arg...)	関数へオブジェクトをバインドする デフォルトで設定される引数を予め設定可能	obj:バインドするオブジェクト arg:事前設定引数 →boundfunction バインド後の関数	xobj.getName.bind(xobj)	func.applyをラップした無名関数を返す
obj.func.bindAsEventListener(obj[, arg...])	リスナーを特定のオブジェクトへバインドする thisがイベント要素ではなくてバインドされたオブジェクトになる。 無名関数なのでイベント解除のメソッド名には使用できない。	obj:バインド元オブジェクト arg:リスナーへの引数 →リスナー（無名関数）	Disp.display.bindAsEventListener(Disp)	bind()だとハンドラの引数が(arg..., event)となるがこれだと(event,arg...)となる。
func.curry(arg..)	カリー化	arg:カリー化するメソッドの引数	String.prototype.splitStr = String.prototype.split.curry(':'); '1:2:3'.splitStr(); →['1','2','3']	
func.delay(sec, arg...)	関数の遅延起動	sec:遅延秒数 arg:起動される関数の引数 →ハンドル	func.delay(5, 'arg'); 5秒後にfuncをargを引数に実行する	
func.defer()	10msの遅延 delay()のcurry		Function.prototype.defer = Function.prototype.delay.curry(0.01); func.defer(); 10ms後にfunc()実行	
func.wrap(				
func.argumentNames(				
func.methodize(arg...)	関数のメソッド化 ある関数（メソッド）を他オブジェクトのメソッドにすることができる。		$w('abs round').each(function(method){ Number.prototype[method] = Math[method].methodize(); }); n.abs(),n.round()という形態で使用可能になる。 =Math.abs(n), Math.round(n)	

Number				
num.succ()	次の値を返す（＋1）		n.succ() →n+1 (10).succ() →11	
num.times(func)	0開始ループ	func:イテレーター func引数　カウンター値（0～n-1）	(10).times(function(n) { total += n; }); total=45	
num.toColorPart()	16進変換		(255).toColorPart() →ff	
num.toPaddedString(len, radix)	ゼロサプレス	len:作成桁数 radix:基数（省略可）	(10).toPaddedString(5); →'00010'	
String				
string.inspect()	Object.inspect()と同じ			
string.strip()	文字列前後の空白、改行、タブを取り除く	→削除後文字列	'　aaa\n'.strip() →'aaa'	全角空白注意 削除するかはUAによる
string.stripScript()	<script>タグの範囲を削除	→削除後文字列	aaa <script type='xxx'> xxxx() </script> bbb → aaa bbb	
string.stripTags()	開始タグと終了タグの削除	→削除後文字列	aaa →’aaa’	
string.truncate(len, str)	文字列の省略 指定長で文字列削除、省略文字を付加する 指定長は省略文字を含む	len:指定長(30) str:省略文字列'...' →削除後文字列	'0123456789'.truncate(7, 'xx') →'01234xx'	
string.escapeHTML()	HTMLエスケープ処理　<>＆などを実体参照にする	→string	<div'.escapeHTML(); →'<div'	
string.unescapeHTML()	HTML文字列の複合化 実体参照を実体にする	→string	<div'.unescapeHTML(); →'<div'	
string.camelize()	ハイフン文字列をキャメル文字列化	→変更後文字列	'border-style'.camelize() →'borderStyle'	

string.capitalize()	文字列最初の文字のみ大文字にする	→変更後文字列	'COOL STAFF'.capitalize() →'Cool staff'	
string.dasherize()	アンダースコアーをハイフンに変換	→変更後文字列	'aaa_bbb'.dasherize() →'aaa-bbb'	
string.underscore()	キャメル文字列をアンダースコア文字列へ （大文字を小文字へ変換）	→変更後文字列	'aaaBbbCCC'.underscore() →'aaa_bbb_ccc'	
string.gsub(reg, str | func)	文字列置換 /gオプション同等 繰り返し置換 イテレーター時はリターン値が置き換え文字列になる。 Rubyテンプレート使用可能{}	reg:パターン、正規表現 str:置き換え文字列 func:イテレータ（引数：マッチ配列） →置換後文字列	“abcdefg”.gsub(/[ac]/, “-”) →”-b-defg” “abcdefg”.gsub(/[ac]/, function(match){return match[0]})	
string.sub(reg, str | func, [cnt])	文字列置換　繰り返し置換回数指定 イテレーター時はリターン値が置き換え文字列になる。	reg:パターン、正規表現 str:置き換え文字列 func:イテレータ（引数：マッチ配列） cnt:置換回数（=1） →置換後文字列	“abcdefg”.sub(/[ac]/, “-”) →”-bcdefg”	
string.interpolate(object[, pattern])	Templateオブジェクトの作成 テンプレート：Ruby形式 #{xxx}	object:置き換え：プロパティ名 置き換え値：プロパティ値	“aa#{xx}bb”.interpolate({xx:”ccc”}); →”aacccbb”	

string.scan(reg, func)	マッチ配列をイテレータに渡す
	置換は行わない
		reg:パターン、正規表現
		func:イテレーター
		
		→同じ文字列
			var str = [];
			“aa ab”.scan('aa', function(txt){
			str.push(txt);
			});
			→str : [aa];	

string.parseQuery(sep) string.toQueryParams()	Query文字列をオブジェクトに変換 （decodeURIComponent()でデコードされる）	sep:セパレータ(&)	“?aaa=123&bbb=take” →{aaa:”123”, bbb:”take”} =array['aaa']='123' array['bbb']='take'	エンコード hash.toQueryString()
Array				
array.include(str) (member())	arrayの中にstrがあるか検索 ==で比較	a:配列、列挙 str:検索文字列 →Boolean	$R(1,10).include(“5”) →true　5==”5”	
array.size()	配列要素数 =array.length			
array.clear()	配列オールクリアー	→空の配列	a.clear()	

array.compact()	null,undefined要素を外して新しい配列を作成	→新配列	var new_a = old_a.compact()	
array.flatten()	再帰配列を平坦な配列に変換	→新配列	[11, [aa, bb], 22] →[11, aa, bb, 22]	
array.intersect(array)	配列二つの共通値だけの配列作成	array:二つ目の配列 →配列	[1,2,3,2] [1,2] →[1,2]	
array.reduce()	要素が1つの配列から値を取り出す	→配列、値	[1].reduce() → 1 [1,2,3].reduce() → [1,2,3]	
array.uniq([sorted])	順序はそのままで重複値を取り除く	sorted:ソート済みtrue（false） →配列	[1,2,3,2,4]→[1,2,3,4]	
array.without(value, …)	指定した値を取り除く新しい配列作成	value:削除値 →配列	[1,2,3,2].without(2) →[1,3]	
array.first()	配列の一番最初の値取りだし	→値 空配列時:undefined	[1,2,3]→1	
array.last()	配列の一番最後の値取りだし	→値 空配列時:undefined	[1,2,3]→3	
array.indexOf(value)	配列内値検索　位置取得 比較が==のため変換有り注意 文字列大文字小文字区別有り	value:検索値 →位置（0～）該当無し：－1	[1,2,3].indexOf(1) →0	
array.inspect()				
Enumerable				
enum.each(func)	反復処理 配列、範囲による反復 returnで飛び越し（continue相当） throw $breakで中断	func:イテレータ イテレータ引数：要素値、インデックス →enumerable	[“one”, “two”].each(function(s, idx){}) $R(1,10).each(function(n){})	

enum.all([func])
(every())
	要素値を調べる
	false条件になると反復終了
	false等価の値が含まれていたらfalseを返す
	false等価が無い場合はtrueを返す
	func有り時はその条件に従う
		func:イテレータ
		イテレータ引数：要素値、インデックス
		
		→Boolarn
			[1, 2, null, 3].all()
			→false (nullで終了)
			[50, 80, 90].all(function(n){
			return n >=50;
			})
			→true	

enum.any([func]) (some())	要素値を調べる true条件になると反復終了 true等価の値が含まれていたらtrueを返す true等価が無い場合はfalseを返す func有り時はその条件に従う	func:イテレータ イテレータ引数:要素値、インデックス →Boolarn	[1, 2, null, 3].any() →true (1で終了) [10, 51, 90].any(function(n){ return n >=50; (51で終了) }) →true	

enum.find(func)
(detect())
	イテレータがtrueを返す要素値を返す	func:イテレータ
		イテレータ引数:要素値、インデックス
		
		→要素値
		無い場合:undefined
			[“row”, “is”].find(function(n){
			return n.length == 2;
			})
			→”is”	

enum.findall(func)
(filter())
	イテレータがtrueを返す要素値を全部返す	func:イテレータ
		イテレータ引数:要素値、インデックス
		
		→要素の配列
		無い場合:[]
			[“row”, “is”, “a”].find(function(n){
			return n.length >= 2;
			})
			→[“row”, ”is”]	

enum.grep(pattern [, func])	正規表現に一致する要素を返す 一致する値全部	pattern:正規表現パターン func:一致した値を渡すイテレータ 返り値が値となる。 イテレータ引数:要素値、インデックス	$R(1,20).grep(/[05]$/); →[5,10,15,20]	

enum.reject(func)	入れテータがfalseを返す要素値を全部返す
	findallの逆
		func:イテレータ
		イテレータ引数:要素値、インデックス
		
		→要素の配列
		無い場合:[]
			[“row”, “is”, “a”].reject(function(n){
			return n.length >= 2;
			})
			→[”a”]	

enum.partition([func])	要素を二つのテーブルに分ける イテレータが無い場合はブール値により分ける。 イテレータがある場合は返り値のブール値により分ける。	func:イテレータ イテレータ引数:要素値、インデックス	[1, 2, null, 3, false].partition() →[[1,2,3], [null, false]]	

enum.eachSlice(size [,func])	要素をsizeで指定されたサイズに分割する。→配列の配列へ
	sizeに足りない分は要素無しのまま。
		size:分割要素数
		func:イテレータ
		イテレータには分割後の1配列が渡される。
		
		→配列の配列
			$R(1,7).eachSlice(4, function(e){
			// [1, 2, 3, 4]
			// [5, 6, 7]の2回くる
			return e;
			}
			→[[1, 2, 3, 4],[5,6,7]]	

enum.inGroupOf(size [,filler])	要素をsizeで指定されたサイズに分割する →配列の配列へ sizeに足りない分はfillerで埋める	size:分割要素数 filler:埋める値(null) →配列の配列	$R(1,7).inGroupOf(4) →[[1,2,3,4],[5,6,7,null]]	
enum.zip(sequence... [,func])	複数の配列の要素毎の結合	sequence:	$w('aa bb cc').zip($R(1,3)) →[['aa', 1],['bb', 2],['cc, 3]]	

enum.inject(accu, func)	全ての要素を結合する アキュムレータに値が積算される。	accu:アキュムレータ初期値（オブジェクト参照可） func:イテレータ イテレータ引数：アキュムレータ変数、要素値、インデックス	$R(1,10).inject(0,function(acc,n) {return acc+n;}) →55	
enum.collect(func) map()	イテレータの処理結果を配列とする	func:イテレータ イテレータ引数：要素値、インデックス	$R(1,3).map(function(n){ return n*n;}) →[1, 4, 9]	
enum.invoke(func [,arg])	全ての要素に対して指定メソッドで処理を行う	func:メソッド arg:メソッドの引数	[“aaa”, “bbb”].invoke("toUpperCase") →[“AAA”, “BBB”] $(“navbar”, “adsbar”).invoke(“hide”)	
enum.pluck(prname)	要素のプロパティー値の取得	prname:プロパティー名	[“hello”, “way”].pluck(“length”) →[5, 3] $$(“.cool”).pluck(“tagName”)	
enum.max([func])	要素内の最大値を取得 イテレータ有り時は処理後結果(返値）の最大値	func:イテレータ イテレータ引数：要素値、インデックス	[1,2,4,10,5,3].max(function(value, index){ return -value; }); →－1	
enum.min([func])	要素内の最小値を取得 イテレータ有り時は処理後結果(返値）の最小値	func:イテレータ イテレータ引数：要素値、インデックス	[1,2,4,10,5,3].min(function(value, index){ return -value; }); →－10	
enum.sortBy(func)	イテレータの処理結果により昇順にソートする。	func:イテレータ イテレータ引数：要素値、インデックス	[1,2,4,10,5,3,9,23].sortBy(function(value, index){ return -value;}); →[23,10,9,5,4,3,2,1]	
enum.inspect()			→'#<Enumerable:[1,2]>'	
Event				
event.stop()	バブリング停止 イベントデフォルト動作キャンセル	event:イベントオブジェクト	event.stop()	
event.stopPropagation()	バブリング停止 イベントデフォルト動作そのまま	event:イベントオブジェクト		
event.preventDefault()	イベントデフォルト動作キャンセル クロス化	event:イベントオブジェクト		

event.element()	イベント発生源の要素	event:イベントオブジェクト →要素オブジェクト(HTMLElement)	e.element().tagName	.targetがテキストノードを返すことがあるがこれは無い
event.findElement(tagname)	直近（祖先）の指定された要素を見つける それが自分と同じだと自分になる	tagname:検索タグ名 →要素オブジェクト（HTMLElement)	ele = e.element('p')	
event.relatedTarget	mouseoverイベント：どの要素から来たか mouseoutイベント：どの要素へ行くか の要素を取得 クロス化	→要素オブジェクト(HTMLElement)		
event.pointerX()	ページ座標の取得　X座標 スクロールに影響を受けない1ページ全体の座標	→Number		クライアントペース座標取得には元からのClient?()を使用 見える描画域のみの座標を取得
event.pointerY()	ページ座標の取得　Y座標 スクロールに影響を受けない1ページ全体の座標	→Number		
event.isLeftClick()	マウス左クリックがされたかチェック	→Boolean on:true		
event.isMiddleClick()	マウス真ん中クリックがされたかチェック	→Boolean on:true		
event.isRightClick()	マウス右クリックがされたかチェック	→Boolean on:true		
event.ctrlKey	CTRLキー押下判断	→Boolean on:true		
event.altKey	ALTキー押下判断	→Boolean on:true		
event.shiftKey	SHIFTキー押下判断	→Boolean on:true		
event.keyCode				
element.fire(eventname, [,memo])	独自イベントの起動 リスナー登録されたものを起動する 通常のイベントと同じ処理になる			
new Form.EventObserver(form, callback)	フォームの内容変更ハンドラ	form:name | id callback:変更時に呼び出される関数 callback引数：form要素 　フィールド名と値 　name=value&name=valueというようにフォーム内全フィールド 　変更フィールドだけではない		フォーカスが移動したら発生

new Form.Element.EventObserver(field, callback)	指定フィールドの内容変更ハンドラ	field:id （name不可） callback:変更時に呼び出される関数 callback引数：フィールド要素 value:値		フォーカスが移動したら発生
DOM				
Element.extend(element)	要素をprototype拡張にする $()を使ってない要素を拡張 各種拡張関数が使えるようになる。 $()でも同じく拡張される	element: 要素	Element.extend(elem);	
new Element(elem [,{atr}])	要素の作成	elem:要素名（文字列） atr:属性名:属性値（オブジェクト）	a = new Element('h1', {id:'main'});	
element.observe(eventName, observer)	リスナーの登録	eventName:イベント名 on無し observe:ハンドラ	$('clickId').observe('click', click_func)	
element.stopObserve([eventName, [observe]])	イベント解除	eventName:イベント名 on無し observe:ハンドラ それぞれ省略可	$('clickId').stopObserve()	
element.up([selector][, index])	index段階上（親）の一番近い要素を取得する 親の兄弟要素は「考慮されない」	selecotr:要素名 'p'等 index:0(=1段階）～ →要素	$('idaa').up('ul', 0) $('idaa').up(2)	親要素の兄弟は取得できない。 その場合、.previous()や.next()を使用する
element.down([selector][, index])	子要素へ向かってindex番目の一番近い要素を取得する 子の兄弟要素は「考慮される」	selecotr:要素名 'p'等 index:0(=1番目）～ →要素	$('idaa').down('p', 0) $('idaa').down(2)	
element.firstDescendant()	最初の子要素を返す element.firstChildの概念だが、W3C対応ブラウザの改行や空白のノードは考慮されない（IEタイプ） ただしテキストノードは取得できない。	→要素	$('idaa').firstDescendant().firstChild.nodeValue; →'text' <ul id='idaa'> text	
element.next([selector, index])	index番次（兄弟）の一番近い要素を取得する	selecotr:要素名 'p'等 index:0(=1段階）～ →要素	$('idaaa').next('li', 0)	
element.previous([selector, index])	index番前（兄弟）の一番近い要素を取得する	selecotr:要素名 'p'等 index:0(=1段階）～ →要素	$('idaaa').previous('li', 0)	

element.clearWhitespace()	子要素がテキストノードで全て空白の場合に子要素を削除する(removeChild())	→要素	$('ida').cleanWhitespace(); →要素のテキストノードを削除 	
element.remove()	指定要素の削除		$('idaaa').remove();	
element.replace(content)	指定された要素内の要素をcontentで置き換える <a>ならば部を置き換え	content:置き換えコンテンツ →要素	$('idaaa').replace(' text ')	
element.update(content)	指定された要素をcontentで置き換える <a>ならば<a>の全てを置き換え	content:置き換えコンテンツ →要素	$('idaaa').update(' text ')	
element.addClassName(className)	要素にcssクラス名を追加する	className:クラス名 →要素	$('idaaa').addClassName('cshide')	
element.hasClassName(className)	要素に指定のcssクラス名が存在するか調査	className:クラス名 →Boolean true:有り	if ($('idaaa').hasClassName('cshide'))	
element.removeClassName(classNname)	要素からcssクラス名を削除する	className：クラス名 →要素	$('idaaa').removeClassName('cshide')	
element.toggleClassName(classNname)	要素のcssクラス名の存在をトグル クラス名有り→削除 クラス名無し→追加	classNmae:クラス名 →要素	$('idaaa').toggleClassName('cshide')	
element.hide()	要素を隠す(display使用）	→要素		
element.show()	要素を表示	→要素		
element.toggle()	要素の状態を切り替え(hide/show)	→要素		
element.visible()	要素の表示状態を取得	Boolean 表示中:true		
element.insert(contents)	要素の指定位置にコンテンツを挿入する 位置指定が無いときは子要素の最後へ追加される	コンテンツ（文字列形式） 位置指定オブジェクト 要素の前（兄弟）:before 要素の後（兄弟）:after 要素の子要素の先頭:top 要素の子要素の最後:bottom（デフォ）	$('idaa').insert({top:' text '}) $('idaa').insert(' text ') （bottomと同）	
element.wrap(elem [,atr])	要素を指定要素で囲む	elem:作成要素名（文字列） atr:要素の属性、属性値（オブジェクト）	$('idaa').wrap('div', {className:'elm'})	囲えるのは1要素のみ

element.adjacent([selector])	指定のセレクターに一致する兄弟要素全ての取得（その子要素も）	selector:cssセレクタ →要素の配列	$('nyc').adjacent('li.us'); →[<li#chi, li#la, li#aus>]	selector#findChildElements()と同じ動作
element.ancestors()	上（親）方向に要素を取得する	→要素の配列 （自分の次からhtmlまでの順番）	<html> <body> $('kid').ancestors(); →[body, html] （自分の次からhtmlまでの順番）	同レベルに要素複数ある場合も全て
element.descendants()	下（子）方向に要素を取得する	→要素の配列 （自分の次から下方向の順番）	$('australopithecus').descendants();	
element.previousSiblings()	前（兄）方向に要素を取得する	→要素の配列 （自分の前から遠方の順番）	$('mcintosh').previousSiblings();	
element.nextsiblings()	後（弟）方向に要素を取得する	→要素の配列 （自分の後から遠方の順番）	$('mutsu').nextSiblings();	
element.childElements()	一段下（子）の要素を全て取得する	→要素の配列	$('homo-erectus').childElements();	
element.descendantOf(ancestor)	ancester（先祖）の子孫かチェック	→Boolean true:子孫	 $('homo-sapiens').descendantOf('australopithecus'); →true	
element.siblings()	兄弟要素全ての取得	→要素の配列	<li id="golden-delicious">Golden Delicious <li id="mutsu">Mutsu <li id="mcintosh">McIntosh $('mutsu').siblings(); →[li#golden-delicious, li#mcintosh]	
element.empty()	要素内テキストノードが空白はチェック	→Boolean true:空白	 $('wallet').empty(); →true	
element.getStyle(propaertyName)	CSSプロパティ値の取得 インラインではないプロパティも取得可能（計算値）	propertyName:プロパティ名 キャメル形状orCSS形状（速度遅）	$('test').getStyle('margin-left'); →'12px'	取得値はブラウザによる safariはインラインで無いものはnull

element.setStyle({name:value,...})	CSSプロパティ値の設定	name:プロパティ名
		value:設定値
		ハッシュ形式or文字列
		ハッシュ形式のときはプロパティ名はキャメル形状のみ
			$(element).setStyle({
			  cssFloat: 'left',
			  opacity: 0.5
			});
			$(element).setStyle('font-size: 12px;');
				IE6は？

element.cumulativeOffset()	位置取得 スクロールに無関係なページ上の位置	→PositionInfo 配列 [x, y] プロパティ:left, top 配列でアクセスできプロパティーをもつ	elt.cumulativeOffset().left	
element.cumulativeScrollOffset( )	位置取得 コンテナのスクロールに隠れた部分のサイズ	→PositionInfo 配列 [x, y] プロパティ:left, top 配列でアクセスできプロパティーをもつ	elt.cumulativeScrollOffset().left	
element.positionedOffset()	位置取得 コンテナ内座標系	→PositionInfo 配列 [x, y] プロパティ:left, top 配列でアクセスできプロパティーをもつ	elt.positionedOffset().left	
element.viewportOffset()	位置取得 ブラウザの見えている範囲での位置	→PositionInfo 配列 [x, y] プロパティ:left, top 配列でアクセスできプロパティーをもつ	elt.viewportOffset().left	
element.getOffsetParent()	配置の基準となるコンテナ要素の取得	→HTMLElement		
element.absolutize()	絶対位置指定要素への変換	→要素		視覚上の位置は変わらない
element.relativize()	相対位置指定要素への変換	→要素		視覚上の位置は変わらない
element.clonePosition(source [, options])	位置情報をコピーして摘要	source:位置情報取得要素 options:取得情報指定 setLeft,setTop,setWidth,setHeight offsetTop,OffsetLeft →PositionInfo 配列 [x, y] プロパティ:left, top 配列でアクセスできプロパティーをもつ	elt.clonePosition(src, { setHeight: false, offsetTop:10 });	set?はどの位置情報を使用するかの指定（初期値：true） offset?は元要素位置からのオフセット（初期値：0）
element.getDimensions()	要素のサイズの取得	→{width:num height:num}		
element.getWidht()	要素の幅の取得	→サイズ		
element.getHeidht()	要素の高さの取得	→サイズ		
element.makeClipping()	要素にクリッピング領域を持たせる クリッピング領域外は表示されない	→付加した要素	$('framer').makeClipping().setStyle({width: '100px', height: '100px'});	
element.undoClipping()	要素からクリッピング領域を外す	→外した要素	$('framer').undoClipping()	
element.makePositioned()	相対座標系に変更”relative”へ	→要素		
element.undoPositioned()	座標系を無指定に変更	→要素		

element.scrollTo()	指定要素までスクロールする	→要素		
element.hasAttribute(attrName)	指定の属性を持つか検査	attrName:HTML属性名 →要素	Prototype $('link').hasAttribute('href');	
element.readAttribute(attrName)	指定の属性値の読み込み	attrName:HTML属性名 →文字列 属性値	$('tag').readAttribute('href'); → '/tags/prototype'	
element.writeAttribute(attrName[,value])	属性値の設定	attrName:HTML属性名 value:属性値（初期値true） ※ハッシュ形式で複数指定可能 →要素	$('tag').writeAttribute({ href:'/tags/prototype'});	値にfalse,nullで属性削除 trueでdisabled="disabled"などになる。
element.identify()	一意なID値の設定 既にある場合はID値を返す 無い場合はID値を設定し、設定値を返す	→ID値（文字列）		
element.inspect()				
element.match(selector)	子要素にselectorの要素があるか調べる	selector:CSS3セレクタ →true:有り false:無し	$('divtag').match('p#classa');	
element.select(selector)	子要素のselectorの要素を取得する $$()との違いは、開始エレメントの指定ができること	selector:CSS3セレクタ →ノード配列	$('divtag').select('p#classa img');	
Selector				
new Selector(css_sel)	セレクタオブジェクトの作成	css_sel:cssセレクタ	selobj = new Selector("div.selec");	findElements()と合わせて$$()と同様機能となる
sel.findElements([root])	セレクタオブジェクトによる要素検索	root:検索範囲 　　初期値：document →要素	selobj.findElements()	
Selector.matchElements(elements, expression)				
Selector.findElement(elements, [,expression='*'][,index=0])				
Selector.findChildElemens(element, expression...)				

Form				
Form.Element→Field	別名			
Field.focus(fieldelt)	フィールドへフォーカスをあてる	fieldelt:id →要素	Field.focus('youso').select()	
Filed.select(fieldelt)	フィールドのテキストを選択状態にする	fieldelt:id →要素	Field.select('youso')	
fieldelt.activate()	フィールドへフォーカスしコンテンツを選択状態にする 選択はtext状態のものなど focus+select	→要素	$('youso').activate()	
fieldelt.clear()	フィールドコンテンツのクリア（空欄へ）	→要素	$('youso').clear()	
fieldelt.present()	フィールドコンテンツの空欄調査	→Boolean true:入力有り false:空文字列	$('youso').present() == true	
formelt.findFirstElement()	フォーム中最初の要素を取得 tabindexの一番小さい物 値無し時は最初に現れる <input><select><textarea>	→要素	felt = $('formyouso').findFirstElement()	
formelt.focusFiestElement()	フォーム中の最初の要素を取得し、フォーカスをあてテキストを選択状態にする findFirstElement()+activate()	→要素	felt = $ ('formyouso').focusFirstElement()	
fieldelt.enable()	フィールドを有効にする	→要素		
fieldelt.disable()	フィールドを無効にする	→要素		
formelt.enabale()	フォームを有効にする	→要素		
formelt.disable()	フォームを無効にする	→要素		
formelt.getElements()	フォーム内全フィールド要素を取得 <input><select><textarea>を対象 'hidden'でも取得	→要素の配列		
formelt.getInputs([type][,name])	指定の<input>の要素を取得 type属性、name属性で絞り込み nameのみ時はtypeをnullに	→要素の配列	$('myform').getInputs('text', 'textname')	
fieldelt.getValue()	フィールドから値の取得 ＝$F() 基本value値 チェック無しcheckboxはnull リストボックスは選択値（マルチは配列）	→値、値の配列		

fieldelt.setValue(value | [value,])	フィールド値の設定	value:設定値、設定値の配列 →要素		
fieldelt.serialize()	フィールド値をシリアライズ化（URLエンコード）する name属性=value属性値の形になる	→エンコード後文字列		
formelt.serialize([option])	フォームのシリアライズ化 値:getValue()で取得 null値→空文字列 undefine→name属性値のみ 配列→同名の複数フィールドとして	option:ハッシュ{} hash:true 返値のハッシュ化 submit:name submit要素の特定 →string | hash オプションによりハッシュオブジェクト	$('formid').serialize({hash:true, submit:'send'}) 'username=sulien&age=22&hobbies=coding&hobbies=hiking' ハッシュ {username: 'sulien', age: '22', hobbies: ['coding', 'hiking']}	引数がtrueのみだとハッシュ化となる
Form.serializeElements(elements, [option])	form.serialize()と同			
new Form.Observer(formelt, interval, callback)	フォーム内フィールドの監視時間間隔指定の変更ハンドラ設定	formelt:フォーム要素 時間間隔:秒指定（小数指定可能） callback:ハンドラ	new Form.Observe('formid', 0.3,function{...})	
new Field.Observer(fieldelt, interval, callback)	フィールドの監視時間間隔指定の変更ハンドラ設定	fieldelt:フィールド要素 時間間隔:秒指定（小数指定可能） callback:ハンドラ	new Form.Observe('fieldid', 0.3,function{...})	
Hash				
new Hash([obj])	Hashの作成=$H([obj]) オブジェクトのPrototype拡張化			
hash.get(propNamae)	プロパティ値の取得	propName:プロパティ名 →プロパティ値	$H({a:111, b:222}).get(a) →111	
hash.set(propName, propValue)	プロパティ値の設定 重複の場合は値置換	propName:プロパティ名 propValue:プロパティ値 →プロパティ値(設定値)	$H({a:111, b:222}).set('a', 333) →{a:333, b:222}	
hash.unset(propName)	指定のプロパティを削除する	→削除したプロパティ値	h.unset('b')	
hash.keys()	ハッシュ内キーの取得	→キーの配列	$H({a:1, b:2}).keys() →['a', 'b']	
hash.values()	ハッシュ内値の取得	→値の配列	$H({a:1, b:2}).values() →[1, 2]	

hash.index(value)	指定データを持つキーの取得 最初に一致したキーひとつ 順番はfor..inに依存	→キー	$H({a:1, b:2}).index(2) →'b'	
hash.merge(obj)	ハッシュの結合 引数の方が優先度が上	obj:結合するハッシュオブジェクト →新しいハッシュ	$H({a:1, b:2}).merge({b:3, c:4}) →{a:1,b:3,c:4}	
hash.update(obj)	新しいハッシュを作成さずにオリジナルを更新する	obj:更新データのハッシュ →hash	h={a:11, b:22} h.update({b:33}) →h={a:11, b,33}	
hash.toQueryString()	ハッシュからクエリー文字列を作成する undefine値はプロパティ名のみに null値を持つ場合は削除 返り値はエンコードされる (encodeURIComponent())	→String	$H({a:1, b:2}).toQueryString() →'a=1&b=2'	Object.toQueryString(obj) は引数はハッシュではなくて通常オブジェクト デコード string.parseQuery()
hash.each(iterator)	enumerableのeachとは機能は同じだが別 イテレータの引数はキー、値を持つオブジェクト（配列化する）	iterator:イテレータ イテレータ引数：obj obj.key obj.value obj[0]：キー、obj[1]：値	h.each(function(vl){ vl.key vl.value })	
hash.inspect()	ハッシュ内を文字列化	→String	→'<#Hash:{a:1,b:2}>'	
hash.toObject()	ハッシュを通常のオブジェクトへ	→Object		
hash.clone()	ハッシュのクローン作成	→newHash		
Template				
new Template(tempText)	テンプレート元テキスト作成	→Template(オブジェクト)	tpl = new Template('NAME: #{name}. AGE:#{age}') obj {name:'Akira', age:function(){..}}	#{name},#{age}が置き換わる nameがオブジェクトのプロパティ名 これのプロパティ値で置き換え tpl.evalute(obj)で適用 メソッドの場合はその返値
tpl.evalute(obj)	テンプレートへobjの内容を適用する	obj:オブジェクト →摘要後テキスト	txt = tpl.evalute(obj)	

toTemplateReplacements()	テンプレートへ渡すオブジェクトをこの関数内の自作ロジックで作成する(関数値がプロパティ値になる)
	tpl.evalute(obj)内で最初に呼ばれる
	tpl.evalute(obj)の引数になるobj内で定義する
		→Object	obj{a:function(){
			return 1;
			},
			toTemplateReplacements(){
			//関数値がテンプレートへ
			return a();
			}
			}	

new PeriodicalExecute(callback,intervalSec)	時間指定での関数定期起動 =setInterval() 二重起動無し 停止：pe:stop()	callback:コールバック関数 　引数：pe: PrriodicalExecuter intervalSec:間隔秒数（小数可）	new PriodicalExecuter(function(pe){ ・・・・ pe.stop();　// 起動終了 })	
公式 http://www.prototypejs.org/ ver1.6.0.2 cheatsheet http://attic.scripteka.com/prototyp				written by Samael