

After parsing sheet:

```

    181     if (!request) throw new Error(`empty request body`),;
    182     validateTime(request.time, tlConfig); request = (sheet: Array(1), time: {...})
    183
    184     tlConfig.time = request.time; request = (sheet: Array(1), time: {...})
    185     tlConfig.time.to = parseDateMath(request.time.to, true).valueOf();
    186     tlConfig.time.from = parseDateMath(request.time.from).valueOf();
    187     tlConfig.time.interval = calculateInterval(
    188       tlConfig.time.from,
    189       tlConfig.time.to,
    190     );
    191     tlConfig.settings[ `timelion:target_buckets` ] || 200,
    192     tlConfig.time.interval,
    193     tlConfig.settings[ `timelion:min_interval` ] || '1ms',
    194   );
    195
    196   tlConfig.setTargetSeries();
    197
    198   stats.invokeTime = (new Date()).getTime();
    199   stats.queryCount = 0;
    200   queryCache = {};
    201
    202   // This is setting the "global" sheet, required for resolving references
    203   const sheet = parseSheet(request.sheet); request = (sheet: Array(1), time: {...})
    204   return _map(sheet, function (chainList, i) {
    205     return _map(chainList, function (seriesList) {
    206       stats.sheetTime = (new Date()).getTime();
    207       return seriesList;
    208     }).catch(function (e) {
    209       | throwWithCell(i, e);
    210     });
    211   });
    212
    213 }
    214
    215 return {
    216   processRequest: processRequest,
    217   getStats: function () { return stats; }
    218 };
    219
  220
}

```

(From `chain_runner.js`) Coverage: n/a

After pre-process sheet, we have a `queries` object:

```

    127   return Promise.all(seriesList).then(function (args) {
    128     const list = _chain(args).pluck('list').flatten().value();
    129     const seriesList = _merge.apply(this, _flatten([[], args]));
    130     seriesList.list = list;
    131     return seriesList;
    132   });
    133
    134
    135   function preprocessSheet(sheet) { sheet = [Array(1)] }
    136
    137   let queries = (); queries = _es(*): {...}
    138   _each(sheet, function (chainList, i) { sheet = [Array(1)]
    139     try {
    140       const queriesInCell = _mapValues(preprocessChain(chainList), function (val
    141         val.cell = i;
    142         return val;
    143       ));
    144       queries = _extend(queries, queriesInCell); queries = _es(*): {...}
    145     } catch (e) {
    146       | throwWithCell(i, e);
    147     });
    148   });
    149   queries = _values(queries);
    150
    151   const promises = _chain(queries).values().map(function (query) {
    152     return invoke(query.function, query.arguments);
    153   }).value();
    154
    155   return Promise.settle(promises).then(function (resolvedDatasources) {
    156     stats.queryTime = (new Date()).getTime();
    157     _each(resolvedDatasources, function (query, i) {
    158       const funcDef = tlConfig.server.plugins.timelion.getFunction(query.func
    159       const resolvedDataSource = resolvedDatasources[i];
    160
    161       if (resolvedDataSource.isRejected()) {
    162         if (funcDef.isBuiltin) {
    163           | throw resolvedDataSource.reason();
    164         } else {
    165           | throwWithCell(query.cell, resolvedDataSource.reason());
    166         }
    167       }
    168       | queryCache[funcDef.cacheKey(query)] = resolvedDataSource.value();
    169     });
    170   });
    171
  172
}

```

(From `chain_runner.js`) Coverage: n/a

Then, we loop over the `queries` array (which is converted to an array by using `values` method) and pass into `invoke` the `query.function` and `query.arguments` for invoking the function.

In the `invoke` function, first we get the definition of the function based on its name (`query.function`):

```

19
20
21 import _ from 'lodash';
22 import Promise from 'bluebird';
23 import parseSheet from './lib/parse_sheet.js';
24 import parseDateMath from './lib/date_math.js';
25 import repositionArguments from './lib/reposition_arguments.js';
26 import validateTime from './lib/validate_time.js';
27 import calculateInterval from '../common/lib';
28
29
30 export default function chainRunner(tlConfig) {
31   const preprocessChain = require('./lib/preprocess_chain')(tlConfig);
32
33   let queryCache = {};
34   const stats = {};
35   let sheet;
36
37   function throwWithCell(cell, exception) {
38     throw new Error(`in cell #${cell} + ${exception.message}`);
39   }
40
41   // Invokes a modifier function, resolving arguments into series as needed
42   function invoke(fnName, args) {
43     const functionDef = tlConfig.server.plugins.timelion.getFunction(fnName);
44
45     function resolveArgument(item) {
46       if (Array.isArray(item)) {
47         return Promise.all(item.map(resolveArgument));
48       }
49
50       if (!_.isObject(item)) {
51         switch (item.type) {
52           case 'function':
53             const itemFunctionDef = tlConfig.server.plugins.timelion.getFunction(item.name);
54             if (itemFunctionDef.cacheKey &amp; queryCache[itemFunctionDef.cacheKey]) {
55               stats.queryCount++;
56               return Promise.resolve(_.cloneDeep(queryCache[itemFunctionDef.cacheKey]));
57             }
58             return invoke(item.function, item.arguments);
59           case 'reference':
60             let reference;
61             if (item.series) {
62               reference = sheet[item.plot - 1][item.series - 1];
63             } else {
64               ...
65             }
66             return invoke('first', [reference]);
67           case 'chain':
68             return invokeChain(item);
69           case 'chainList':
70             return resolveChainList(item.list);
71           case 'literal':
72             return item.value;
73           case 'requestList':
74             case 'seriesList':
75             return item;
76           ...
77         }
78       }
79       throw new Error(`Argument type not supported: ${JSON.stringify(item)}`);
80     } else {
81       return item;
82     }
83   }
84
85   args = repositionArguments(functionDef, args);
86   args = [Array(1), functionDef];
87
88   args = _.map(args, resolveArgument);
89
90   ...
91 }

```

Line 88, Column 5 (From chain_runner.js) Coverage: n/a

Debugger paused

This function definition includes three types of function: `fn`, `originalFn` and `timelionFn`. Then, it loops over the `args` (which is `query.arguments`) and resolve arguments:

```

45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90

```

Line 50, Column 7 (From chain_runner.js) Coverage: n/a

Debugger paused

In this case, the argument in `args` is an object with '`literal`' type so it just return the literal value:

```

    Jun 27 9:17:07 PM
    DevTools
    Debugger paused

    Scripts Workspace Snippets ...
    child_process.js internal/child_process.js socket.js chain_runner.js run.js >>

    + Add folder
    table_vis tagcloud testbed tests_bundle tile_map timelion common/lib public server fit_functions handlers __tests__ lib chain_runner.js lib __tests__ classes alter.js asSorted.js build_target.js date_math.js functions_md.js get_namespaced_setting load_functions.js offset_time.js offset_time.test.js process_function_definiti reduce.js split_interval.js to_milliseconds.js unzipPairs.js routes functions.js run.js validate_es.js

    43 const functionDef = tlConfig.server.plugins.timelion.getFunction(fName);
    44
    45 function resolveArgument(item) { item = {type: 'literal', value: '', location: ...}
    46   if (Array.isArray(item)) {
    47     return Promise.all(_.map(item, resolveArgument));
    48   }
    49
    50   if (_.isObject(item)) { item = {type: 'literal', value: '', location: ...}
    51   switch (item.type) {
    52     case 'function':
    53       const itemFunctionDef = tlConfig.server.plugins.timelion.getFunction(item.name);
    54       if (itemFunctionDef.cacheKey && queryCache[itemFunctionDef.cacheKey][item.location]) {
    55         stats.queryCount++;
    56         return Promise.resolve(_.cloneDeep(queryCache[itemFunctionDef.cacheKey]));
    57       }
    58       return invoke(item.function, item.arguments); item = {type: 'literal'}
    59     case 'reference':
    60       let reference;
    61       if (item.series) { item = {type: 'literal', value: '*', location: ...}
    62       reference = sheet[item.plot - 1][item.series - 1];
    63     } else {
    64       reference = {
    65         type: 'chainList',
    66         list: sheet[item.plot - 1] item = {type: 'literal', value: '*', location: ...}
    67       };
    68     }
    69     return invoke('first', [reference]);
    70   }
    71   case 'chain':
    72     return invokeChain(item); item = {type: 'literal', value: '*', location: ...}
    73   case 'chainList':
    74     return resolveChainList(item.list); item = {type: 'literal', value: '*'}
    75   case 'literal':
    76     return item.value;
    77   case 'requestList':
    78   case 'seriesList':
    79     return item;
    80   }
    81 }
    82 throw new Error ('Argument type not supported: ' + JSON.stringify(item));
    83 } else {
    84   return item;
    85 }
    86 }

    args = repositionArguments(functionDef, args);
  
```

(From chain_runner.js) Coverage: n/a

Jun 27 9:17:07 PM

DevTools

Debugger paused

Watch
Breakpoints
Pause on uncaught exceptions
Pause on caught exceptions
chain_runner.js
run.js
Scope
Local
args: Array(1)
Closure (invoke)
Closure (chainRunner)
Global
Call Stack
Show ignore-listed frames
resolveArgument
invoke
(anonymous)
preProcessSheet
processRequest
handler

Actually, the value is wrapped in a promise.

```

    Jun 27 9:18:31 PM
    DevTools
    Debugger paused

    Scripts Workspace Snippets ...
    child_process.js internal/child_process.js socket.js chain_runner.js run.js >>

    + Add folder
    table_vis tagcloud testbed tests_bundle tile_map timelion common/lib public server fit_functions handlers __tests__ lib chain_runner.js lib __tests__ classes alter.js asSorted.js build_target.js date_math.js functions_md.js get_namespaced_setting load_functions.js offset_time.js offset_time.test.js process_function_definiti reduce.js split_interval.js to_milliseconds.js unzipPairs.js routes functions.js run.js validate_es.js

    68   }
    69   return invoke('first', [reference]);
    70 }
    71 case 'chain':
    72   return invokeChain(item);
    73 case 'chainList':
    74   return resolveChainList(item.list);
    75 case 'literal':
    76   return item.value;
    77 case 'requestList':
    78 case 'seriesList':
    79   return item;
    80 }

    args = repositionArguments(functionDef, args); args = [Promise], functionDef = ...
    81
    82   args = ..._map(args, resolveArgument); args = [Promise], resolveArgument = f
    83
    84   return Promise.all(args).then(function (args) {
    85     args.byName = indexArguments(functionDef, args);
    86     return functionDef.fn(args, tlConfig);
    87   });
    88
    89   function invokeChain(chainObj, result) {
    90     if (chainObj.chain.length === 0) return result[0];
    91     const chain = _.clone(chainObj.chain);
    92     const link = chain.shift();
    93
    94     let promise;
    95     if (link.type === 'chain') {
    96       promise = invokeChain(link);
    97     } else if (!result) {
    98       promise = invoke('first', [link]);
    99     } else {
    100       const args = link.arguments ? result.concat(link.arguments) : result;
    101       promise = invoke(link.function, args);
    102     }
    103
    104     const args = link.arguments ? result.concat(link.arguments) : result;
    105     promise = invoke(link.function, args);
    106   }
    107
    108   const args = link.arguments ? result.concat(link.arguments) : result;
    109   promise = invoke(link.function, args);
    110
    111   const args = link.arguments ? result.concat(link.arguments) : result;
    112   promise = invoke(link.function, args);
    113 }

    args = repositionArguments(functionDef, args); args = [Promise], functionDef = ...
  
```

(From chain_runner.js) Coverage: n/a

Jun 27 9:18:31 PM

DevTools

Debugger paused

Watch
Breakpoints
Pause on uncaught exceptions
Pause on caught exceptions
chain_runner.js
run.js
Scope
Local
args: Array(1)
Closure (invoke)
Closure (chainRunner)
Global
Call Stack
Show ignore-listed frames
resolveArgument
invoke
(anonymous)
preProcessSheet

The indexed argument is an argument with `byName` object that has `q` array inside it:

After that, the `invoke` function will call `functionDef.fn`, which is a function of the `Timelion` class, with all of the indexed arguments:

```

    70     , return invoke('first', [reference]);
    71   }
    72   case 'chain':
    73     return invokeChain(item);
    74   case 'chainList':
    75     return resolveChainList(item.list);
    76   case 'literal':
    77     return item.value;
    78   case 'requestList':
    79   case 'seriesList':
    80     return item;
    81   }
    82   throw new Error ('Argument type not supported: ' + JSON.stringify(item));
    83 } else {
    84   return item;
    85 }
    86 }

    args = repositionArguments(functionDef, args);
    88
    89   args = _map(args, resolveArgument);
    90
    91   return Promise.all(args).then(function (args) {
    92     | argsByName = _indexArguments(functionDef, args);
    93     | return functionDef.fn(args, tlConfig);
    94   });
    95 }

    function invokeChain(chainObj, result) {
      if (chainObj.chain.length === 0) return result[0];
    100
    101   const chain = _clone(chainObj.chain);
    102   const link = chain.shift();
    103
    104   let promise;
    105   if (link.type === 'chain') {
    106     promise = invokeChain(link);
    107   } else if (!result) {
    108     promise = invoke('first', [link]);
    109   } else {
    110     const args = link.arguments ? result.concat(link.arguments) : result;
    111     promise = invoke(link.function, args);
    112   }
    113
    114   return promise.then(function (result) {
      | return invokeChain(result, chain);
    115 });
    116
    117 }

    promise.then(function (result) {
      | return invokeChain(result, chain);
    118 });
    119
    120
    121
    122
    123
    124
    125
    126
    127
    128
    129
    130
    131
    132
    133
    134
    135
    136
    137
    138
    139
    140
    141
    142
    143
    144
    145
    146
    147
    148
    149
    150
    151
    152
    153
    154
    155
    156
    157
    158
    159
    160
    161
    162
    163
    164
    165
  
```

(From `chain_runner.js` Coverage: n/a)

After that, the `invoke` function will call `functionDef.fn`, which is a function of the `Timelion` class, with all of the indexed arguments:

The `fn` function will call the `originalFn`, which comes from `config.fn` where `config` is `tlConfig`. And that function is `fn` of `Datasource` extends `Timelion` class:

```

    22 const fitFunctions = loadFunctions('fit_functions');
    23
    24 export default class TimelionFunction {
    25   constructor(name, config) {
    26     this.name = name;
    27     this.args = config.args || [];
    28     this.argsByName = _indexBy(this.args, 'name');
    29     this.help = config.help || '';
    30     this.alises = config.alises || [];
    31     this.extended = config.extended || false;
    32
    33   // WTF is this? How could you not have a fn? Wtf would the thing be used for?
    34   const originalFunction = config.fn || function (input) { return input; };
    35
    36   // Currently only re-fits the series.
    37   this.originalFn = originalFunction;
    38
    39   this.fn = function (args, tlConfig) {
    40     const config = _clone(tlConfig);
    41     return Promise.resolve(originalFunction(args, config)).then(function (seriesList) {
    42       seriesList.list = _map(seriesList.list, function (series) {
    43         const target = tlConfig.getTargetSeries();
    44
    45         // Don't fit if the series are already the same
    46         if (_isEqual(_map(series.data, 0), _map(target, 0))) return series;
    47
    48         let fit;
    49         if (args.byName.fit) {
    50           fit = args.byName.fit;
    51         } else if (series.fit) {
    52           fit = series.fit;
    53         } else {
    54           fit = 'nearest';
    55         }
    56
    57         series.data = fitFunctions[fit](series.data, tlConfig.getTargetSeries());
    58       });
    59
    60       return seriesList;
    61     });
    62   };
    63 }
    64
    65
  
```

(From `timelion_function.js` Coverage: n/a)

The `fn` function will call the `originalFn`, which comes from `config.fn` where `config` is `tlConfig`. And that function is `fn` of `Datasource` extends `Timelion` class:

```

    export default class Datasource extends TimelionFunction {
      constructor(name, config) {
        // Additional arguments that every dataSource take
        config.args.push({
          name: 'offset',
          type: ['string', 'null'],
          help: 'Offset the series retrieval by a date expression, e.g., -1M to make events from one month ago appear as if they are happening now. Offset the series relative to the charts overall time range, by using the e.g. "timerange:-2" will specify an offset that is twice the overall chart'
        });

        config.args.push({
          name: 'fit',
          types: ['string', 'null'],
          help: 'Algorithm to use for fitting series to the target time span and interval'
        });

        // Wrap the original function so we can modify inputs/outputs with offset & fit
        const originalFunction = config.fn;
        config.fn = function (args, tlConfig) {
          const config = _clone(tlConfig);
          let offset = args.byName.offset;
          if (offset) {
            offset = preprocessOffset(offset, tlConfig.time.from, tlConfig.time.to);
            config.time = _cloneDeep(tlConfig.time);
            config.time.from = offsetTime(config.time.from, offset);
            config.time.to = offsetTime(config.time.to, offset);
          }

          return Promise.resolve(originalFunction(args, config)).then(function (seriesList) {
            if (seriesList.list === 0) throw new Error(name + ' returned no results');
            seriesList.data = offsetSeries(seriesList.data, offset);
            seriesList.fit = args.byName.fit || seriesList.fit || 'nearest';
            return seriesList;
          });
        };
      }
    }
  
```

(From datasource.js) Coverage: n/a

As we can see, it calls to another `originalFn` named `esFn` which returns a `seriesList` object:

```

    ];
  },
  help: 'Pull data from an elasticsearch instance',
  aliases: ['elasticsearch'],
  fn: async function esFn(args, tlConfig) {
    args = [Array(1), byName: {}], tlConfig = {server: m.e.e.s.i.s.Plugin}
    const config = _defaults(_clone(args.byName), { config: {q: Array(1), metric: Array(1), split: undefined, type: 'index-pattern', fields: ['title', 'fields'], search: `${config.index}`}, index: tlConfig.settings['timelion:es.default_index'], tlConfig = {server: m.e.e.s.i.s.Plugin, request: i.timefield: tlConfig.settings['timelion:es.timefield'], interval: tlConfig.time.interval, kibana: true, fit: 'nearest'}});
    const findResp = await tlConfig.request.getSavedObjectsClient().find({ findResp = {page: 1, per_page: 20, type: 'index-pattern', fields: ['title', 'fields'], search: `${config.index}`}, config = {q: Array(1), metric: Array(1), split: undefined, index: 'all', type: 'index-pattern'} });
    const indexPatternSavedObject = findResp.saved_objects.find(savedObject => { indexPatternSavedObject = undefined | return savedObject.title === config.index; config = {q: Array(1), metric: Array(1), split: undefined} });
    let scriptedFields = []; scriptedFields = []
    if (indexPatternSavedObject) { indexPatternSavedObject = undefined | const fields = JSON.parse(indexPatternSavedObject.attributes.fields); scriptedFields = fields.filter(field => { scriptedFields = [] | return field.scripted; });
    }

    const body = buildRequest(config, tlConfig, scriptedFields); body = {index: 'all', body: {}, timeout: '30s'}
    const { callWithRequest } = tlConfig.server.plugins.elasticsearch.getCluster('data'); tlConfig = {server: m.e.e.s.i.s.Plugin}
    const resp = await callWithRequest(tlConfig.request, 'search', body);
    if (!resp._shards.total) throw new Error(`Elasticsearch index not found: ${config.index}`);
    return {
      type: 'serieslist',
      list: toSeriesList(resp.aggregations, config)
    };
  }
}
  
```

(From index.js) Coverage: n/a

That function is used for "Pull data from an elasticsearch instance" by calling `callWithRequest` function.

The `seriesList` looks like this in `Datasource.fn`:

```

36     export default class Datasource extends TimelionFunction {
37       constructor(name, config) {
38
39         // Additional arguments that every dataSource take
40         config.args.push({
41           name: 'offset',
42           type: ['string', 'null'],
43           help: 'Offset the series retrieval by a date expression. ' +
44             ' e.g. -1M to make events from one month ago appear as if they are happening now. ' +
45             ' Offset the series relative to the charts overall time range, by using the value "timerange", ' +
46             ' e.g. "timerange:-2" will specify an offset that is twice the overall chart time range to the past.'
47         });
48
49         config.args.push({
50           name: 'fit',
51           type: ['string', 'null'],
52           help: 'Algorithm to use for fitting series to the target time span and interval. Available: ' + _keys(fit)
53         );
54
55         // Wrap the original function so we can modify inputs/outputs with offset & fit
56         const originalFunction = config.fn;
57         config.fn = function (args, tlConfig) {
58           const config = _clone(tlConfig);
59           let offset = args.byName.offset;
60           if (offset) {
61             offset = preprocessOffset(offset, tlConfig.time.from, tlConfig.time.to);
62             config.time = _cloneDeep(tlConfig.time);
63             config.time.from = offsetTime(config.time.from, offset);
64             config.time.to = offsetTime(config.time.to, offset);
65           }
66
67           return Promise.resolve(originalFunction(args, config)).then(function (seriesList) {
68             seriesList.list = _map(seriesList.list, function (series) {
69               if (series.data.length === 0) throw new Error(name + ' returned no results');
70               series.data = offsetSeries(series.data, offset);
71               series.fit = args.byName.fit || series.fit || 'nearest';
72               return series;
73             });
74           });
75
76           super(name, config);
77         };
78
79         // You need to call timelionFn if calling up a datasource from another datasource,
80       }
81     }
  
```

(From `datasource.js`) Coverage: n/a

And it looks like this in `Timelion.fn`:

```

22   const fitFunctions = loadFunctions('fit_functions');
23
24   export default class TimelionFunction {
25     constructor(name, config) {
26       this.name = name;
27       this.args = config.args || [];
28       this.argsByName = _indexBy(this.args, 'name');
29       this.help = config.help || '';
30       this.alternatives = config.alternatives || [];
31       this.extended = config.extended || false;
32
33       // WTF is this? How could you not have a fn? Wtf would the thing be used for?
34       const originalFunction = config.fn || function (input) { return input; };
35
36       // Currently only re-fits the series.
37       this.originalFn = originalFunction;
38
39       this.fn = function (args, tlConfig) {
40         const config = _clone(tlConfig);
41         return Promise.resolve(originalFunction(args, config)).then(function (seriesList) {
42           seriesList.list = _map(seriesList.list, function (series) {
43             const target = tlConfig.getTargetSeries();
44
45             // Don't fit if the series are already the same
46             if (_isEqual(_.map(series.data, 0), _.map(target, 0))) return series;
47
48             let fit;
49             if (args.byName.fit) {
50               fit = args.byName.fit;
51             } else if (series.fit) {
52               fit = series.fit;
53             } else {
54               fit = 'nearest';
55             }
56
57             series.data = fitFunctions[fit](series.data, tlConfig.getTargetSeries());
58             return series;
59           });
60           return seriesList;
61         });
62       };
63     }
64   }
  
```

(From `timelion_function.js`) Coverage: n/a

After pre-process sheet, the chain list is:

```

182 validateTime(request.time, tlConfig);
183
184     tlConfig.time = request.time;
185     tlConfig.time.to = parseDateMath(request.time.to, true).valueOf();
186     tlConfig.time.from = parseDateMath(request.time.from).valueOf();
187     tlConfig.time.interval = calculateInterval(
188         tlConfig.time.to,
189         tlConfig.time.from,
190         tlConfig.settings['timelion:target_buckets'] || 200,
191         tlConfig.time.interval,
192         tlConfig.settings['timelion:min_interval'] || '1ms',
193     );
194
195     tlConfig.setTargetSeries();
196
197     stats.invokeTime = (new Date()).getTime();
198     stats.queryCount = 0;
199     queryCache = {};
200
201     // This is setting the "global" sheet, required for resolving references
202     sheet = parseSheet(request.sheet);
203     return preprocessSheet(sheet).then(function () {
204         return _map(sheet, function (chainList, i) {
205             return _map(sheet, function (seriesList) {
206                 stats.sheetTime = (new Date()).getTime();
207                 return seriesList;
208             }).catch(function (e) {
209                 throwWithCell(i, e);
210             });
211         });
212     });
213
214     return {
215         processRequest: processRequest,
216         getStats: function () { return stats; }
217     };
218 }
219
220

```

(From chain_runner.js) Coverage: n/a

And the code will call the `resolveChainList` for resolving the chain list:

```

98     function invokeChain(chainObj, result) {
99         if (!chainObj.chain.length === 0) return result[0];
100
101         const chain = _clone(chainObj.chain);
102         const link = chain.shift();
103
104         let promise;
105         if (link.type === 'chain') {
106             promise = invokeChain(link);
107         } else if (!result) {
108             promise = invoke('first', [link]);
109         } else {
110             const args = link.arguments ? result.concat(link.arguments) : result;
111             promise = invoke(link.function, args);
112         }
113
114         return promise.then(function (result) {
115             return invokeChain({ type: 'chain', chain: chain }, [result]);
116         });
117     }
118
119
120     function resolveChainList(chainList) { chainList = [...] }
121     const seriesList = _map(chainList, function (chain) {
122         const values = invoke('first', [chain]);
123         return values.then(function (args) {
124             return args;
125         });
126     });
127     return Promise.all(seriesList).then(function (args) {
128         const list = _chain(args).pluck('list').flatten().value();
129         const seriesList = _merge.apply(this, _flatten([[], args]));
130         seriesList.list = list;
131         return seriesList;
132     });
133
134     function preprocessSheet(sheet) {
135
136         let queries = {};
137         _each(sheet, function (chainList, i) {
138             try {
139                 const queriesInCell = _mapValues(preprocessChain(chainList), function (val) {
140                     val.cell = i;
141                     return val;
142                 });
143                 queries = _extend(queries, queriesInCell);
144             }
145         });
146     }

```

(From chain_runner.js) Coverage: n/a

Each chain will be mapped to a Promise that is returned from a function that `invoke` with '`first`' as `fnName` and the chain itself as `args`.

The function definition is get by the name again:

```

function invoke(fnName, args) {
  const functionDef = tlConfig.server.plugins.timelion.getFunction(fnName);
  if (Array.isArray(item)) {
    return Promise.all(_.map(item, resolveArgument));
  }
  if (_.isObject(item)) {
    switch (item.type) {
      case 'function':
        const itemFunctionDef = tlConfig.server.plugins.timelion.getFunction(item.function);
        if (itemFunctionDef.cacheKey & queryCache[itemFunctionDef.cacheKey[item]]) {
          stats.queryCount++;
          return Promise.resolve(_.cloneDeep(queryCache[itemFunctionDef.cacheKey[item]]));
        }
        return invoke(item.function, item.arguments);
      case 'reference':
        let reference;
        if (item.series) {
          reference = sheet[item.plot - 1][item.series - 1];
        } else {
          reference = {
            type: 'chainList',
            list: sheet[item.plot - 1]
          };
        }
        return invoke('first', [reference]);
      case 'chain':
        return invokeChain(item);
      case 'chainList':
        return resolveChainList(item.list);
      case 'literal':
        return item.value;
      case 'requestList':
      case 'seriesList':
        return item;
    }
    throw new Error('Argument type not supported: ' + JSON.stringify(item));
  } else {
    return item;
  }
}

```

(From chain_runner.js) Coverage: n/a

Line 88, Column 5

Debugger paused

This time, `resolveArguments` receive `args` as an array and it will call the `resolveArguments` recursively for resolving each argument. But, we have only one argument with index `0`. And that argument is an object with `type == chain`.

So, in the switch case, it will call the `invokeChain` function and pass the `chain` object into it:

```

function invoke(fnName, args) {
  const functionDef = tlConfig.server.plugins.timelion.getFunction(fnName);
  if (Array.isArray(item)) {
    return Promise.all(_.map(item, resolveArgument));
  }
  if (_.isObject(item)) {
    switch (item.type) {
      case 'function':
        const itemFunctionDef = tlConfig.server.plugins.timelion.getFunction(item.function);
        if (itemFunctionDef.cacheKey & queryCache[itemFunctionDef.cacheKey[item]]) {
          stats.queryCount++;
          return Promise.resolve(_.cloneDeep(queryCache[itemFunctionDef.cacheKey[item]]));
        }
        return invoke(item.function, item.arguments);
      case 'reference':
        let reference;
        if (item.series) {
          item = {type: 'chain', chain: Array(2)};
          reference = sheet[item.plot - 1][item.series - 1];
        } else {
          reference = {
            type: 'chainList',
            list: sheet[item.plot - 1]
          };
        }
        return invoke('first', [reference]);
      case 'chain':
        return invokeChain(item);
      case 'chainList':
        return resolveChainList(item.list);
      case 'literal':
        return item.value;
      case 'requestList':
      case 'seriesList':
        return item;
    }
    throw new Error('Argument type not supported: ' + JSON.stringify(item));
  } else {
    return item;
  }
}

function invokeChain(item) {
  if (item.type === 'chain') {
    const itemFunctionDef = tlConfig.server.plugins.timelion.getFunction(item.function);
    if (itemFunctionDef.cacheKey & queryCache[itemFunctionDef.cacheKey[item]]) {
      stats.queryCount++;
      return Promise.resolve(_.cloneDeep(queryCache[itemFunctionDef.cacheKey[item]]));
    }
    return invoke(item.function, item.arguments);
  }
  return item;
}

```

(From chain_runner.js) Coverage: n/a

Line 73, Column 20

invokeChain

4 matches

Cancel

Debugger paused

Inside the `invoke chain`, it will `shift` the `chain` array. The `shift` function will return the removed element so `link` will be `es` object:

The screenshot shows the Chrome DevTools interface with the DevTools tab selected. The left sidebar shows the project structure under the Scripts tab, with the `chain_runner.js` file currently open. The main pane displays the `invokeChain` function. A yellow box highlights the line `promise = invoke(link.function, args);`. The status bar at the bottom right shows "Paused on breakpoint".

```
84 |     return item;
85 |
86 |
87 |   }
88 |
89 |   args = repositionArguments(functionDef, args);
90 |
91 |   args = _map(args, resolveArgument);
92 |
93 |   return Promise.all(args).then(function (args) {
94 |     args.byIdName = indexArguments(functionDef, args);
95 |     return functionDef.fn(args, tlcConfig);
96 |   });
97 |
98 | }
99 |
100| function invokeChain(chainObj, result) { chainObj = {type: 'chain', chain: Array(2)}, result = if (chainObj.Dchain.length === 0) Dreturn result[0];
101|
102|   const chain = _clone(chainObj.chain); chain = { ... }, chainObj = {type: 'chain', chain: Arr
103|   const link = chain.shift(); link = {type: 'function', function: 'es', arguments: Array(1), :
104|   let promise; promise = undefined
105|   if (link.type === 'chain') {
106|     promise = [invokeChain(link);
107|   } else if (!result) {
108|     promise = invoke('first', [link]);
109|   } else {
110|     const args = link.arguments ? result.concat(link.arguments) : result;
111|     promise = invoke(link.function, args);
112|   }
113|
114|   return promise.then(function (result) {
115|     return [invokeChain({ type: 'chain', chain: chain }, [result]);
116|   });
117|
118| }
119|
120| function resolveChainList(chainList) {
121|   const seriesList = _map(chainList, function (chain) {
122|     const values = invoke('first', [chain]);
123|     return values.then(function (args) {
124|       | return args;
125|     });
126|   });
127|   return Promise.all(seriesList).then(function (args) {
128|     | return item;
129|   });
130|
131| }
132|
133| args = repositionArguments(functionDef, args);
134|
135| args = _map(args, resolveArgument);
136|
137| return Promise.all(args).then(function (args) {
138|   args.byIdName = indexArguments(functionDef, args);
139|   return functionDef.fn(args, tlcConfig);
140| });
141|
142| }
143|
144| args = _map(args, resolveArgument);
145|
146| return Promise.all(args).then(function (args) {
147|   args.byIdName = indexArguments(functionDef, args);
148|   return functionDef.fn(args, tlcConfig);
149| });
150|
151| }
152|
153| args = _map(args, resolveArgument);
154|
155| return Promise.all(args).then(function (args) {
156|   args.byIdName = indexArguments(functionDef, args);
157|   return functionDef.fn(args, tlcConfig);
158| });
159|
160| }
161|
162| args = _map(args, resolveArgument);
163|
164| return Promise.all(args).then(function (args) {
165|   args.byIdName = indexArguments(functionDef, args);
166|   return functionDef.fn(args, tlcConfig);
167| });
168|
169| }
170|
171| args = _map(args, resolveArgument);
172|
173| return Promise.all(args).then(function (args) {
174|   args.byIdName = indexArguments(functionDef, args);
175|   return functionDef.fn(args, tlcConfig);
176| });
177|
178| }
179|
180| args = _map(args, resolveArgument);
181|
182| return Promise.all(args).then(function (args) {
183|   args.byIdName = indexArguments(functionDef, args);
184|   return functionDef.fn(args, tlcConfig);
185| });
186|
187| }
188|
189| args = _map(args, resolveArgument);
190|
191| return Promise.all(args).then(function (args) {
192|   args.byIdName = indexArguments(functionDef, args);
193|   return functionDef.fn(args, tlcConfig);
194| });
195|
196| }
197|
198| args = _map(args, resolveArgument);
199|
200| return Promise.all(args).then(function (args) {
201|   args.byIdName = indexArguments(functionDef, args);
202|   return functionDef.fn(args, tlcConfig);
203| });
204|
205| }
206|
207| args = _map(args, resolveArgument);
208|
209| return Promise.all(args).then(function (args) {
210|   args.byIdName = indexArguments(functionDef, args);
211|   return functionDef.fn(args, tlcConfig);
212| });
213|
214| }
215|
216| args = _map(args, resolveArgument);
217|
218| return Promise.all(args).then(function (args) {
219|   args.byIdName = indexArguments(functionDef, args);
220|   return functionDef.fn(args, tlcConfig);
221| });
222|
223| }
224|
225| args = _map(args, resolveArgument);
226|
227| return Promise.all(args).then(function (args) {
228|   args.byIdName = indexArguments(functionDef, args);
229|   return functionDef.fn(args, tlcConfig);
230| });
231|
232| }
233|
234| args = _map(args, resolveArgument);
235|
236| return Promise.all(args).then(function (args) {
237|   args.byIdName = indexArguments(functionDef, args);
238|   return functionDef.fn(args, tlcConfig);
239| });
240|
241| }
242|
243| args = _map(args, resolveArgument);
244|
245| return Promise.all(args).then(function (args) {
246|   args.byIdName = indexArguments(functionDef, args);
247|   return functionDef.fn(args, tlcConfig);
248| });
249|
250| }
251|
252| args = _map(args, resolveArgument);
253|
254| return Promise.all(args).then(function (args) {
255|   args.byIdName = indexArguments(functionDef, args);
256|   return functionDef.fn(args, tlcConfig);
257| });
258|
259| }
260|
261| args = _map(args, resolveArgument);
262|
263| return Promise.all(args).then(function (args) {
264|   args.byIdName = indexArguments(functionDef, args);
265|   return functionDef.fn(args, tlcConfig);
266| });
267|
268| }
269|
270| args = _map(args, resolveArgument);
271|
272| return Promise.all(args).then(function (args) {
273|   args.byIdName = indexArguments(functionDef, args);
274|   return functionDef.fn(args, tlcConfig);
275| });
276|
277| }
278|
279| args = _map(args, resolveArgument);
280|
281| return Promise.all(args).then(function (args) {
282|   args.byIdName = indexArguments(functionDef, args);
283|   return functionDef.fn(args, tlcConfig);
284| });
285|
286| }
287|
288| args = _map(args, resolveArgument);
289|
290| return Promise.all(args).then(function (args) {
291|   args.byIdName = indexArguments(functionDef, args);
292|   return functionDef.fn(args, tlcConfig);
293| });
294|
295| }
296|
297| args = _map(args, resolveArgument);
298|
299| return Promise.all(args).then(function (args) {
300|   args.byIdName = indexArguments(functionDef, args);
301|   return functionDef.fn(args, tlcConfig);
302| });
303|
304| }
305|
306| args = _map(args, resolveArgument);
307|
308| return Promise.all(args).then(function (args) {
309|   args.byIdName = indexArguments(functionDef, args);
310|   return functionDef.fn(args, tlcConfig);
311| });
312|
313| }
314|
315| args = _map(args, resolveArgument);
316|
317| return Promise.all(args).then(function (args) {
318|   args.byIdName = indexArguments(functionDef, args);
319|   return functionDef.fn(args, tlcConfig);
320| });
321|
322| }
323|
324| args = _map(args, resolveArgument);
325|
326| return Promise.all(args).then(function (args) {
327|   args.byIdName = indexArguments(functionDef, args);
328|   return functionDef.fn(args, tlcConfig);
329| });
330|
331| }
332|
333| args = _map(args, resolveArgument);
334|
335| return Promise.all(args).then(function (args) {
336|   args.byIdName = indexArguments(functionDef, args);
337|   return functionDef.fn(args, tlcConfig);
338| });
339|
340| }
341|
342| args = _map(args, resolveArgument);
343|
344| return Promise.all(args).then(function (args) {
345|   args.byIdName = indexArguments(functionDef, args);
346|   return functionDef.fn(args, tlcConfig);
347| });
348|
349| }
350|
351| args = _map(args, resolveArgument);
352|
353| return Promise.all(args).then(function (args) {
354|   args.byIdName = indexArguments(functionDef, args);
355|   return functionDef.fn(args, tlcConfig);
356| });
357|
358| }
359|
360| args = _map(args, resolveArgument);
361|
362| return Promise.all(args).then(function (args) {
363|   args.byIdName = indexArguments(functionDef, args);
364|   return functionDef.fn(args, tlcConfig);
365| });
366|
367| }
368|
369| args = _map(args, resolveArgument);
370|
371| return Promise.all(args).then(function (args) {
372|   args.byIdName = indexArguments(functionDef, args);
373|   return functionDef.fn(args, tlcConfig);
374| });
375|
376| }
377|
378| args = _map(args, resolveArgument);
379|
380| return Promise.all(args).then(function (args) {
381|   args.byIdName = indexArguments(functionDef, args);
382|   return functionDef.fn(args, tlcConfig);
383| });
384|
385| }
386|
387| args = _map(args, resolveArgument);
388|
389| return Promise.all(args).then(function (args) {
390|   args.byIdName = indexArguments(functionDef, args);
391|   return functionDef.fn(args, tlcConfig);
392| });
393|
394| }
395|
396| args = _map(args, resolveArgument);
397|
398| return Promise.all(args).then(function (args) {
399|   args.byIdName = indexArguments(functionDef, args);
400|   return functionDef.fn(args, tlcConfig);
401| });
402|
403| }
404|
405| args = _map(args, resolveArgument);
406|
407| return Promise.all(args).then(function (args) {
408|   args.byIdName = indexArguments(functionDef, args);
409|   return functionDef.fn(args, tlcConfig);
410| });
411|
412| }
413|
414| args = _map(args, resolveArgument);
415|
416| return Promise.all(args).then(function (args) {
417|   args.byIdName = indexArguments(functionDef, args);
418|   return functionDef.fn(args, tlcConfig);
419| });
420|
421| }
422|
423| args = _map(args, resolveArgument);
424|
425| return Promise.all(args).then(function (args) {
426|   args.byIdName = indexArguments(functionDef, args);
427|   return functionDef.fn(args, tlcConfig);
428| });
429|
430| }
431|
432| args = _map(args, resolveArgument);
433|
434| return Promise.all(args).then(function (args) {
435|   args.byIdName = indexArguments(functionDef, args);
436|   return functionDef.fn(args, tlcConfig);
437| });
438|
439| }
440|
441| args = _map(args, resolveArgument);
442|
443| return Promise.all(args).then(function (args) {
444|   args.byIdName = indexArguments(functionDef, args);
445|   return functionDef.fn(args, tlcConfig);
446| });
447|
448| }
449|
450| args = _map(args, resolveArgument);
451|
452| return Promise.all(args).then(function (args) {
453|   args.byIdName = indexArguments(functionDef, args);
454|   return functionDef.fn(args, tlcConfig);
455| });
456|
457| }
458|
459| args = _map(args, resolveArgument);
460|
461| return Promise.all(args).then(function (args) {
462|   args.byIdName = indexArguments(functionDef, args);
463|   return functionDef.fn(args, tlcConfig);
464| });
465|
466| }
467|
468| args = _map(args, resolveArgument);
469|
470| return Promise.all(args).then(function (args) {
471|   args.byIdName = indexArguments(functionDef, args);
472|   return functionDef.fn(args, tlcConfig);
473| });
474|
475| }
476|
477| args = _map(args, resolveArgument);
478|
479| return Promise.all(args).then(function (args) {
480|   args.byIdName = indexArguments(functionDef, args);
481|   return functionDef.fn(args, tlcConfig);
482| });
483|
484| }
485|
486| args = _map(args, resolveArgument);
487|
488| return Promise.all(args).then(function (args) {
489|   args.byIdName = indexArguments(functionDef, args);
490|   return functionDef.fn(args, tlcConfig);
491| });
492|
493| }
494|
495| args = _map(args, resolveArgument);
496|
497| return Promise.all(args).then(function (args) {
498|   args.byIdName = indexArguments(functionDef, args);
499|   return functionDef.fn(args, tlcConfig);
500| });
501|
502| }
503|
504| args = _map(args, resolveArgument);
505|
506| return Promise.all(args).then(function (args) {
507|   args.byIdName = indexArguments(functionDef, args);
508|   return functionDef.fn(args, tlcConfig);
509| });
510|
511| }
512|
513| args = _map(args, resolveArgument);
514|
515| return Promise.all(args).then(function (args) {
516|   args.byIdName = indexArguments(functionDef, args);
517|   return functionDef.fn(args, tlcConfig);
518| });
519|
520| }
521|
522| args = _map(args, resolveArgument);
523|
524| return Promise.all(args).then(function (args) {
525|   args.byIdName = indexArguments(functionDef, args);
526|   return functionDef.fn(args, tlcConfig);
527| });
528|
529| }
530|
531| args = _map(args, resolveArgument);
532|
533| return Promise.all(args).then(function (args) {
534|   args.byIdName = indexArguments(functionDef, args);
535|   return functionDef.fn(args, tlcConfig);
536| });
537|
538| }
539|
540| args = _map(args, resolveArgument);
541|
542| return Promise.all(args).then(function (args) {
543|   args.byIdName = indexArguments(functionDef, args);
544|   return functionDef.fn(args, tlcConfig);
545| });
546|
547| }
548|
549| args = _map(args, resolveArgument);
550|
551| return Promise.all(args).then(function (args) {
552|   args.byIdName = indexArguments(functionDef, args);
553|   return functionDef.fn(args, tlcConfig);
554| });
555|
556| }
557|
558| args = _map(args, resolveArgument);
559|
560| return Promise.all(args).then(function (args) {
561|   args.byIdName = indexArguments(functionDef, args);
562|   return functionDef.fn(args, tlcConfig);
563| });
564|
565| }
566|
567| args = _map(args, resolveArgument);
568|
569| return Promise.all(args).then(function (args) {
570|   args.byIdName = indexArguments(functionDef, args);
571|   return functionDef.fn(args, tlcConfig);
572| });
573|
574| }
575|
576| args = _map(args, resolveArgument);
577|
578| return Promise.all(args).then(function (args) {
579|   args.byIdName = indexArguments(functionDef, args);
580|   return functionDef.fn(args, tlcConfig);
581| });
582|
583| }
584|
585| args = _map(args, resolveArgument);
586|
587| return Promise.all(args).then(function (args) {
588|   args.byIdName = indexArguments(functionDef, args);
589|   return functionDef.fn(args, tlcConfig);
590| });
591|
592| }
593|
594| args = _map(args, resolveArgument);
595|
596| return Promise.all(args).then(function (args) {
597|   args.byIdName = indexArguments(functionDef, args);
598|   return functionDef.fn(args, tlcConfig);
599| });
600|
601| }
602|
603| args = _map(args, resolveArgument);
604|
605| return Promise.all(args).then(function (args) {
606|   args.byIdName = indexArguments(functionDef, args);
607|   return functionDef.fn(args, tlcConfig);
608| });
609|
610| }
611|
612| args = _map(args, resolveArgument);
613|
614| return Promise.all(args).then(function (args) {
615|   args.byIdName = indexArguments(functionDef, args);
616|   return functionDef.fn(args, tlcConfig);
617| });
618|
619| }
620|
621| args = _map(args, resolveArgument);
622|
623| return Promise.all(args).then(function (args) {
624|   args.byIdName = indexArguments(functionDef, args);
625|   return functionDef.fn(args, tlcConfig);
626| });
627|
628| }
629|
630| args = _map(args, resolveArgument);
631|
632| return Promise.all(args).then(function (args) {
633|   args.byIdName = indexArguments(functionDef, args);
634|   return functionDef.fn(args, tlcConfig);
635| });
636|
637| }
638|
639| args = _map(args, resolveArgument);
640|
641| return Promise.all(args).then(function (args) {
642|   args.byIdName = indexArguments(functionDef, args);
643|   return functionDef.fn(args, tlcConfig);
644| });
645|
646| }
647|
648| args = _map(args, resolveArgument);
649|
650| return Promise.all(args).then(function (args) {
651|   args.byIdName = indexArguments(functionDef, args);
652|   return functionDef.fn(args, tlcConfig);
653| });
654|
655| }
656|
657| args = _map(args, resolveArgument);
658|
659| return Promise.all(args).then(function (args) {
660|   args.byIdName = indexArguments(functionDef, args);
661|   return functionDef.fn(args, tlcConfig);
662| });
663|
664| }
665|
666| args = _map(args, resolveArgument);
667|
668| return Promise.all(args).then(function (args) {
669|   args.byIdName = indexArguments(functionDef, args);
670|   return functionDef.fn(args, tlcConfig);
671| });
672|
673| }
674|
675| args = _map(args, resolveArgument);
676|
677| return Promise.all(args).then(function (args) {
678|   args.byIdName = indexArguments(functionDef, args);
679|   return functionDef.fn(args, tlcConfig);
680| });
681|
682| }
683|
684| args = _map(args, resolveArgument);
685|
686| return Promise.all(args).then(function (args) {
687|   args.byIdName = indexArguments(functionDef, args);
688|   return functionDef.fn(args, tlcConfig);
689| });
690|
691| }
692|
693| args = _map(args, resolveArgument);
694|
695| return Promise.all(args).then(function (args) {
696|   args.byIdName = indexArguments(functionDef, args);
697|   return functionDef.fn(args, tlcConfig);
698| });
699|
700| }
701|
702| args = _map(args, resolveArgument);
703|
704| return Promise.all(args).then(function (args) {
705|   args.byIdName = indexArguments(functionDef, args);
706|   return functionDef.fn(args, tlcConfig);
707| });
708|
709| }
710|
711| args = _map(args, resolveArgument);
712|
713| return Promise.all(args).then(function (args) {
714|   args.byIdName = indexArguments(functionDef, args);
715|   return functionDef.fn(args, tlcConfig);
716| });
717|
718| }
719|
720| args = _map(args, resolveArgument);
721|
722| return Promise.all(args).then(function (args) {
723|   args.byIdName = indexArguments(functionDef, args);
724|   return functionDef.fn(args, tlcConfig);
725| });
726|
727| }
728|
729| args = _map(args, resolveArgument);
730|
731| return Promise.all(args).then(function (args) {
732|   args.byIdName = indexArguments(functionDef, args);
733|   return functionDef.fn(args, tlcConfig);
734| });
735|
736| }
737|
738| args = _map(args, resolveArgument);
739|
740| return Promise.all(args).then(function (args) {
741|   args.byIdName = indexArguments(functionDef, args);
742|   return functionDef.fn(args, tlcConfig);
743| });
744|
745| }
746|
747| args = _map(args, resolveArgument);
748|
749| return Promise.all(args).then(function (args) {
750|   args.byIdName = indexArguments(functionDef, args);
751|   return functionDef.fn(args, tlcConfig);
752| });
753|
754| }
755|
756| args = _map(args, resolveArgument);
757|
758| return Promise.all(args).then(function (args) {
759|   args.byIdName = indexArguments(functionDef, args);
760|   return functionDef.fn(args, tlcConfig);
761| });
762|
763| }
764|
765| args = _map(args, resolveArgument);
766|
767| return Promise.all(args).then(function (args) {
768|   args.byIdName = indexArguments(functionDef, args);
769|   return functionDef.fn(args, tlcConfig);
770| });
771|
772| }
773|
774| args = _map(args, resolveArgument);
775|
776| return Promise.all(args).then(function (args) {
777|   args.byIdName = indexArguments(functionDef, args);
778|   return functionDef.fn(args, tlcConfig);
779| });
780|
781| }
782|
783| args = _map(args, resolveArgument);
784|
785| return Promise.all(args).then(function (args) {
786|   args.byIdName = indexArguments(functionDef, args);
787|   return functionDef.fn(args, tlcConfig);
788| });
789|
790| }
791|
792| args = _map(args, resolveArgument);
793|
794| return Promise.all(args).then(function (args) {
795|   args.byIdName = indexArguments(functionDef, args);
796|   return functionDef.fn(args, tlcConfig);
797| });
798|
799| }
799| }
```

Because the type of `es` is not `'chain'` (it is `'function'`) and `!result` is truthy, the code calls the `invoke` function with `'first'` as `fnName` the `es` object as `args`:

This time, the procedure is the same as above so skip to the call of `invokeChain` with `link` is `props` object:

Jun 27 10:02:59 PM

DevTools

Paused on breakpoint

```
    return item;
  case 'requestList':
  case 'seriesList':
    return item;
  }
  throw new Error ('Argument type not supported: ' + JSON.stringify(item));
} else {
  return item;
}

args = repositionArguments(functionDef, args);

args = _map(args, resolveArgument);

return Promise.all(args).then(function (args) {
  argsByName = indexArguments(functionDef, args);
  return functionDef.fn(args, tlcConfig);
});

function InvokeChain(chainObj, result) {
  if (!chainObj || chainObj.length === 0) return result[0];

  const chain = _clone(chainObj.chain); chain[]; chainObj = {type: 'chain', chain: chain};
  const link = chain.shift(); link = {type: 'function', function: 'props', arguments: chain[0].arguments};

  let promise; promise = undefined;
  if (link.type === 'chain') {
    promise = invokeChain(link);
  } else if (!result) {
    promise = invoke('first', [link]);
  } else {
    const args = link.arguments ? result.concat(link.arguments) : result;
    promise = invoke(link.function, args);
  }

  return promise.then(function (result) {
    return _invokeChain({ type: 'chain', chain: chain }, [result]);
  });
}

function resolveChainList(chainList) {
  const controller = _createChainListFunction(chainList);
  controller.resolveList();
}
```

Again, the code calls the `invoke` function with 'first' as `fnName` the `props` object as `args`:

```

32     let queryCache = {};
33     const stats = {};
34     let sheet;
35
36     function throwWithCell(cell, exception) {
37       throw new Error(`in cell #${(cell + 1)} : ${exception.message}`);
38     }
39
40     // Invokes a modifier function, resolving arguments into series as needed
41     function invoke(fnName, args) { fnName = 'props', args = [{}], [...] }
42     const functionDef = DtConfig.server.plugins.timelion.DgetFunction(fnName);
43
44     function resolveArgument(item) {
45       if (Array.isArray(item)) {
46         return Promise.all(...map(item, resolveArgument));
47       }
48
49       if (..._IsObject(item)) {
50         switch (item.type) {
51           case 'function':
52             const itemFunctionDef = DtConfig.server.plugins.timelion.DgetFunction(item.function);
53             if (itemFunctionDef.cacheKey && queryCache[itemFunctionDef.cacheKey[item]]) {
54               stats.queryCount++;
55               return Promise.resolve(_cloneDeep(queryCache[itemFunctionDef.cacheKey[item]]));
56             }
57             return invoke(item.function, item.arguments);
58           case 'reference':
59             let reference;
60             if (item.series) {
61               reference = sheet[item.plot - 1][item.series - 1];
62             } else {
63               reference = {
64                 type: 'chainList',
65                 list: sheet[item.plot - 1]
66               };
67             }
68             return invoke('first', [reference]);
69           case 'chain':
70             return invokeChain(item);
71           case 'chainList':
72             return resolveChainList(item.list);
73         }
74       }
75     }
    
```

invokeChain

Line 43, Column 25 (From chain_runner.js) Coverage: n/a

The `originalFn` of `props` now is `firstFn` instead of `fn` like `es` and only accept one argument:

```

35     let sheet;
36
37     function throwWithCell(cell, exception) {
38       throw new Error(`in cell #${(cell + 1)} : ${exception.message}`);
39     }
40
41     // Invokes a modifier function, resolving arguments into series as needed
42     function invoke(fnName, args) { fnName = 'props', args = [{}], [...] }
43     const functionDef = DtConfig.server.plugins.timelion.DgetFunction(fnName); functionDef = () =>
44
45     function resolveArgument(item) { resolveArgument = f resolveArgument(item) }
46     if (Array.isArray(item)) { return Promise.all(...map(item, resolveArgument)); resolveArgument = f resolveArgument(item) }
47
48     if (..._IsObject(item)) {
49       switch (item.type) {
50         case 'function':
51           const itemFunctionDef = DtConfig.server.plugins.timelion.DgetFunction(item.function);
52           if (itemFunctionDef.cacheKey && queryCache[itemFunctionDef.cacheKey[item]]) {
53             stats.queryCount++;
54             return Promise.resolve(_cloneDeep(queryCache[itemFunctionDef.cacheKey[item]]));
55           }
56           return invoke(item.function, item.arguments);
57         case 'reference':
58           let reference;
59           if (item.series) {
60             reference = sheet[item.plot - 1][item.series - 1];
61           } else {
62             reference = {
63               type: 'chainList',
64               list: sheet[item.plot - 1]
65             };
66           }
67           return invoke('first', [reference]);
68         case 'chain':
69           return invokeChain(item);
70         case 'chainList':
71           return resolveChainList(item.list);
72         case 'literal':
73           return item.value;
74         case 'requestList':
75       }
    
```

invokeChain

Line 88, Column 5 (From chain_runner.js) Coverage: n/a

That original function is will eventually call the vulnerable `unflatten` function. Actually, it will call the `fn` of `Timelion` first. Then `fn` will call the `originalFn` of `props`:

```

22 const fitFunctions = loadFunctions('fit_functions');
23
24 export default class TimelionFunction {
25   constructor(name, config) {
26     this.name = name;
27     this.args = config.args || [];
28     this.argsByName = _.indexBy(this.args, 'name');
29     this.help = config.help || '';
30     this.alternatives = config.alternatives || [];
31     this.extended = config.extended || false;
32
33     // WTF is this? How could you not have a fn? Wtf would the thing be used for?
34     const originalFunction = config.fn || function (input) { return input; };
35
36     // Currently only re-fits the series.
37     this._originalFn = originalFunction;
38
39     this.fn = function (args, tlConfig) {
40       const config = _clone(tlConfig);
41       const promise = Promise.resolve(originalFunction(args, config));
42       promise.then(function (seriesList) {
43         const target = tlConfig.getTargetSeries();
44
45         // Don't fit if the series are already the same
46         if (!_.isEqual(_.map(seriesList.list, function (series) {
47           let fit;
48           if (argsByName.fit) {
49             fit = argsByName.fit;
50           } else if (series.fit) {
51             fit = series.fit;
52           } else {
53             fit = 'nearest';
54           }
55
56           series.data = fitFunctions[fit](series.data, tlConfig.getTargetSeries());
57         }));
58         return seriesList;
59       });
60     };
61   };
62 }
63
64 }
65
66
67
68
69
70
71
72
73
74
75
76
77
78

```

The `firstFn` of `props` will omit the `inputSeries` and `global` properties in `byName` object of `args`:

```

43 export default new Chainerizer(props, t)
44 args: [
45   {
46     name: 'inputSeries',
47     types: ['seriesList']
48   },
49   {
50     name: 'global',
51     types: ['boolean', 'null'],
52     help: 'Set props on the seriesList vs on each series'
53   }
54 ],
55 extended: {
56   types: ['seriesList', 'number', 'string', 'boolean', 'null'],
57   // Extended args can not currently be multivalued,
58   // multi: false is not required and is shown here for demonstration purposes
59   multi: false
60 },
61 // extended means you can pass arguments that aren't listed. They just won't be in the ordered
62 // They will be passed as args.extended()
63 // help: 'Use at your own risk, sets arbitrary properties on the series. For example .props(label.
64 fn: function firstFn(args) { args = (4) [{}], undefined, Array(1), Array(1), byName: {}]
65   const properties = _unflatten(_.Omit(args.byName, 'inputSeries', 'global'));
66
67   if (args.byName.global) {
68     _assign(args.byName.inputSeries, properties);
69   } else {
70     return alter(args, function (eachSeries) {
71       _assign(eachSeries, properties);
72     });
73   }
74 }
75
76 });
77
78

```

So the `data` passed into `unflatten` will be:

```
{
  label.__proto__.env.AAA: "a"
}
```

Proof:

The screenshot shows a browser's developer tools interface with the 'Sources' tab selected. A file named 'props.js' is open, showing its code. A yellow box highlights the line of code where the pause happened:

```
25 if (!Object(data) || !Array.isArray(data)) return data;
```

The right side of the interface displays the 'Call Stack' and the 'Local' scope variables. The 'Local' scope includes variables like 'this', 'data', and 'label'. The 'data' variable is expanded to show its prototype and properties.