Интеграционные возможности: развитие проекта в модели Wire.js

TODO: вынести то, что касается wire, переводов, примеров, ссылок на документацию, в отдельный раздел.

презентация wire.js	вебинар по cujo.js
Слайды: http://www.slideshare.net/briancavalier/ioc-javascript Github-репозиторий: https://github.com/cujojs/wire	

AMD vs IOC в исполнении wire.js

Существенное отличие - Wire IOC-контейнер строится поверх существующего AMD-loader'a (это может быть requirejs, curl, etc...), дирижирует взаимодействием модулей и тем самым придает осмысленность и глубину процессу их загрузки. Иначе говоря, AMD-загрузчик рисует шахматную доску, и расставляет на ней фигуры. Wire.js представляет партию.

То-есть спецификации wire представляют собой композиционный слой приложения.

[Использование wire.js в проекте представляется стратегически важным.]

http://www.slideshare.net/briancavalier/ioc-javascript, слайд 106

Wire.js и композитные приложения

http://open.bekk.no/composite-architecture-with-javascript

http://open.bekk.no/next-generation-javascript

Сравнение с нынешним состоянием ядра и контролов (неудачные дизайнерские решения, случайные и намеренные)

1. Ядро строится на основе Backbone.Marionette с использованием в ключевых моментов системы promise'ов. Для работы с promise и defered в свое время в прект была включена библиотека when.js (входит в cujo.js ToolKit). В результате ядро представляет из себя некий гибрид из средств отображения представлений (layout, itemView, collectionView etc) и фрагментов resolving'a, идеологически относящихся к IOC. Эту ситуацию хотелось бы исправить, следуя принципу Separation of Concerns.
2. В случае с многоуровневыми контролами (dropdown, например) некоторые объекты передаются по цепочке от главного уровня к подчиненным (collection), поскольку верхний уровень представляет API для работы с остальными. Там, где у разработчика не хватает выдержки строго следовать правилу "never speak with strangers", встречаются обращения с двумя и более точками в ссылке на объект. Этот неудачный дизайн можно будет исправить в wire-спецификации, инъектируя свойства, используя композицию (см.статьи по ссылкам в предыдущем разделе)
3. Options для инициализации путаются с model fields представления, во избежание путаницы нужно определиться с необходимостью указывать настройки внешнего вида контрола в полях модели (изначальный смысл Backbone.Model - в возможности синхронизации данных с сервером). Wire в этом случае предлагает инициализацию с добавлением args к конструктору, или инъекции, заданные посредством properties - они будут доступны, когда компонент перейдет в состояние ready. На этом фоне нынешний метод applyModelProperties, действительно, выглядит нелепым костылем.
4. Как следствие из предыдущего: значимые для бизнес-логики данные находятся в специальной dataModel для отправки в контроллер. При наличии нескольких полей ввода, соответственно, мы имеем несколько dataModel (и все они безо всякой необходимости происходят от Backbone.Model), которые динамически или по требованию агрегируются в ключевой модели

сложного компонента для отправки в медиатор, на сервер, или в localStorage. Cujo.js и в частности wire предлагают здесь data-binding (плагин cola), или решение из области frp-программирования most.js (eventStreams, которые сейчас в проекте реализует библиотека Bacon.js).

5. Локализация в представлении контрола осуществляется встроенной на этапе bootstrap'а приложения в представление функцией prepareLocalized. - А можно было бы сделать это в специальном wire-плагине, при загрузке темплейта! (Остается под вопросом, в какой момент нужно запускать прекомпиляцию handlebars, если мы используем компилируемый coffescript, не исключено, что можно делать это одновременно.).
6. Хотелось бы уменьшить количество связок triggerEvent - catchEvent (trigger-on), а возможно, в дальнейшем и совсем отказаться от event-driven архитектуры в распространенной практике (когда vent-объект назначается арегатором событий в качестве внешнего shared-объекта: нет укорененности в IOC!). Как следствие и попытка создать систему - появление в подчиненных объектах встроенной переменной context (в текущем понимании фреймворка). Вместо этого wire предлагает осмысленный connect к целевой функции и модель синтетических событий. Связывание компонентов пр помощи wire напоминает сложение дробей и приведение к общему знаменателю - connections срабатывают на определенном, общем для всех компонентов шаге (connect stage). На сцену выходит понятие wire-контекста - не как старшего в иерархии элемента (контроллера модуля, например), а как scope IOC-контейнера.
7. Для выяснения массива необходимых прототипов на этапе предварительной загрузки используется regexp для выдергивания из JSON.parsed строки (исходя из идеи, что названия возможных инъенктируемых типов - величина постоянная, что не соответствует действительности). - Wire предлагает каскады контекстов и полноценный IOC-контейнер.
8. БЭМ не нужен, если есть less-компиляция. Но ее можно было бы устроить иначе - css файл должен лежать в той же папке, что и соответствующий компонент, загружаться css!-плагином (resolver'ом, в терминологии wire), и приводиться к namespace при компиляции по возможности автоматически. Таким образом, любой компонент может переноситься без проблем и разрабатываться (/визуально тестироваться) отдельно от неповоротливого и пространного перечня блоков, лежащего где-то вовне.
9. Все элементы управления должны поддерживать события клавиатуры. Выпадающие списки фильтров и выпадающие списки комбобоксов должны расширяться одним и тем же объектом (переделанным keyActiveBase или KeyboardService, как в текущей версии dropDownList). Wire предлагает mixin facet (safe mixin) и возможность переопределения контекстов.
10. Если форма может быть описана отдельно от представления, она (скорей всего) должна быть описана отдельно от представления (backbone.view). Конечно, если это не наносит ущерба для логики. Важней привязать поля формы к модели, и если есть инструмент, позволяющий это сделать без использования множества контролов - почему бы его не использовать? Cujo.js предлагает такой инструмент: cola.js позволяет делать привязку к модели, к коллекции, к localStorage, к HTML-node, к REST-сервису (и если в этом списке нет нужного адресата, может быть написан адаптер для взаимодействия). После binding'а темплейт (.html), содержащий форму, передается в качестве инъекции свойства template в Backbone-(Marionette-)представление, которое умеет работать с внешним видом формы, не изменяя сути уже установленного подключения к источнику (получателю) данных. Таким образом, процесс работы с формой делится на этапы:

этап	отвечает за процессинг
template loading	wire, text!
data binding	cola.js
template injecting	wire
template localization	посредством wire-плагина на стадии ready:before
view (template) rendering	backbone- (marionette-) view
view modification (error reporting)	view (или внешний по отношению к view controller - как правило, он находится в той же директории и описан в той же wire-спецификации, что и view) Модификация происходит с элементами формы, но без нарушения data binding, например, в ответ на ввод неправильных данных
data validation	внешняя функция, оформленная как define-модуль, которая содержит стратегию валидации
destroy phase	controller

11. Wire также включает debug-плагин, который может отдельно использоваться для каждого scope при отладке.

..... (продолжение следует...)

Набор техник

Сделаем акцент на некоторые полезные в дальнейшем техники в исполнении wire в презентации автора:

Принцип разделения ответственности:

http://www.slideshare.net/briancavalier/ioc-javascript, слайды 46 - 54

Композиция методов - для отчетливости workflow

http://www.slideshare.net/briancavalier/ioc-javascript, слайд 84

Перехват и вывод ошибки во внешнем компоненте

http://www.slideshare.net/briancavalier/ioc-javascript, слайд 89

Функции wire в приложении

В зависимости от конфигурации wire-спецификация представляет

- загрузку, ioc-контейнер (базовая функция), app/component bootstrap
- медиатор, связывающий объекты между собой и задающий направления взаимодействия
- при наличии плагинов - обеспечение синхронизации данных и валидацию

Производительность wire.js

Поскольку wire.js является promise-based контейнером, и построена на одной из самых быстрых (из известных) promise-библиотек when.j s, есть основания надеяться, что wire.js не будет слабым звеном с точки зрения производительности:

https://github.com/cujojs/promise-perf-tests#test-results

Последний релиз wire - https://github.com/cujojs/wire/releases/tag/0.10.7 - совместим с when.js 3.0.0, для которого отмечается значительный прирост производительности:

https://github.com/cujojs/when/blob/master/CHANGES.md#300

Архитектура wire и плагинов

Нужно ознакомиться с принципами архитектуры плагинов для wire.js, сводная таблица:

- Plugin API

метод объекта wire	аргумент(ы)		
`resolveRef`	`componentName`	реализация компонента по имени (ссылки на компонент)	
`loadModule`	`moduleId`	загрузка модуля по Id (в понимании AMD), используя текущую платформу для работы с модулями (AMD, CommonJS, etc.).	
`createChild`	`childWireSpec`	метод для создания иерархии контекстов	
`getProxy`	`componentNameOrInstance`	проксирование компонента или произвольного объекта.	
`addInstance`	`instance, name`	регистрация экземпляра компонента под данным именем. Отличается от addComponent тем, что не запускает процессинг в lifecircle	
`addComponent`	`component, name`	регистрация экземпляра компонента под данным именем и его процессинг в component lifecycle	

API располагается здесь https://github.com/cujojs/wire/blob/master/lib/scope.js

- Жизненнный цикл компонента

группа	фазы (steps)	ссылка на код	
init	'create', 'configure', 'initialize'	https://github.com/cujojs/wire/blob/master/lib/lifecycle.js#L24	
startup	'connect', 'ready'	https://github.com/cujojs/wire/blob/master/lib/lifecycle.js#L25	
shutdown	'destroy'	https://github.com/cujojs/wire/blob/master/lib/lifecycle.js#L26	

основные фазы (шаги) дополняются "полутонами" с помощью суффиксов "before", "after": https://github.com/cujojs/wire/blob/master/lib/lifecycle.js#L108-L110

- Типы компонентов
  Компоненты могут быть как
  - одного из простых (тривиальных) javascript-типов: Number, String, Boolean, Date, RegExp, Array, Object literal
  - так и более интересных и сложных типов - для их создания используются Factories

  Назначение плагинов - расширение базового функционала wire, и следовательно, их архитектура является гомогенной по отношению к wire.
  Отсюда же следует, что все возможные factories определяются исключительно задействованными плагинами, (https://github.com/cujojs/wire/blob/master/lib/ComponentFactory.js#L145) - кастомные плагины расширяют basePlugin, который предоставляет встроенные factories (см. ниже).

- Ссылки (References)
  Reference говорит само за себя - это ссылка на существующий ресурс или компонент.

  Пример:

  { $ref: 'myComponent' }

  Ссылка может передаваться плагину для обработки (загрузки компонента или ресурса).

  Пример:

  { $ref: 'resolver!reference-identifier' }

- Reference Resolvers
  Собственно, в понятии Reference Resolvers мы подобрались к плагинам.
  В wire.js об успехе (срыве) разрешения загружаемого компонента или ресурса (точнее, речь идет о ссылке на объект: "resolving a reference" -) сигнализирует объект Resolver, с интегрированным методом resolve (https://github.com/cujojs/wire/blob/master/lib/resolver.js#L72-L75).

  Плагины работают с resolver'ом как инъектированным свойством (это видно по интерфейсу, который должен реализовывать wire-плагин https://github.com/cujojs/wire/blob/master/docs/plugins.md#plugin-instance)

- Интерфейс плагина (или формат плагина - https://github.com/cujojs/wire/blob/master/docs/plugins.md#plugin-instance)

  Плагин (имеется в виду basePlugin.js: https://github.com/cujojs/wire/blob/master/lib/plugin/basePlugin.js), снабжен следующими встроенными особенностями:

встроенные factories	ссылка на код
'module', 'create', 'literal', 'prototype', 'clone', 'compose', 'invoker'	https://github.com/cujojs/wire/blob/master/lib/plugin/basePlugin.js#L277-L283

Factories создают компоненты нетривиальных типов.

factory	документация, код	назначение	примечания
module	https://github.com/cujojs/wire/blob/master/docs/components.md#module, https://github.com/cujojs/wire/blob/master/lib/plugin/basePlugin.js#L215-L217	Загрузка модуля без вызова	Данный способ можно использовать shared-сервисов, введении синглто

create	https://github.com/cujojs/wire/blob/master/docs/components.md#create, https://github.com/cujojs/wire/blob/master/lib/plugin/basePlugin.js#L227-L256	Загрузка модуля с вызовом	Использование конструктора - получ функции. Create не всегда может до функцию как конструктор (с new), ил основании наличия у функции непус способ вызова, см. https://github.com nstructor-option-notes т.е. решает is
compose	https://github.com/cujojs/wire/blob/master/docs/components.md#compose, https://github.com/cujojs/wire/blob/master/lib/plugin/basePlugin.js#L258-L273	Композиция функций	
literal	https://github.com/cujojs/wire/blob/master/docs/components.md#literal, https://github.com/cujojs/wire/blob/master/lib/plugin/basePlugin.js#L100-L124	Задание объектного литерала напрямую в wire-спецификации	
wire	https://github.com/cujojs/wire/blob/master/docs/components.md#wire, (код - ?)	Создание child-контекста	
prototype	https://github.com/cujojs/wire/blob/master/docs/components.md#prototype, https://github.com/cujojs/wire/blob/master/lib/plugin/basePlugin.js#L126-L142	см. create factory	Deprecated, рекомендуется использ

Facets позволяют дополнительно конфигурировать компонент или внедрять в него дополнительное поведение - после того, как компонент создан при помощи factory.

Любой из определенных в плагине facet'ов может предоставлять, если в том есть необходимость, соответствующий метод (hook) на любую из фаз lifecircle компонента. Каждый метод будет параметризирован, и получит одинаковый список параметров: (resolver, proxy, wire) - https://github.com/cujojs/wire/wiki/Plugin-format

встроенные facets (список)	ссылка на код
'properties', 'mixin', 'init', 'ready', 'destroy'	https://github.com/cujojs/wire/blob/master/lib/plugin/basePlugin.js#L288-L308

Дополнительные к встроенным facets предоставляются плагинами.

встроенные facets	применение
'properties'	внедрение свойств - инъекции в поля объекта
'mixin'	расширение объекта при помощи "safe mixin" - введение дополнительных свойств и дополнительного поведения, в mixin могут быть $ref-ссылки на декораторы
'init'	метода инициализации объекта (некоторые действия после инъекции свойств объекта (https://github.com/cujojs/wire/blob/master/docs/configure.md#properties), но до того, как будут установлены connections с другими компонентами, и до того, как другие компоненты смогут создать коннект с данным объектом)
'ready'	во многом похоже на стадию init, единственное но: здесь представлены методы, которые выполнятся после установления connections
'destroy'	методы на стадии уничтожения объекта

- Техника создания посредников (proxies) с помощью proxy API. Данная особенность важна при наличии специализированных подключаемых модулей (компонентов), написанных с учетом set/get API стороннего фреймворка. [уточнить]
- Техника соединений (Connections)
  Эта особенность wire.js позволяет взаимодействовать частям системы без нарушения их кода. Авторы предлагают рассматривать Connections как линии блоксхем, соединяющие компоненты: вызовы методов, потоки событий, предоставление данных укладываются в парадигму connections. Коннекторы бывают разных типов: DI, javascript events, AOP programming. Удобство connections также и в том, что может быть задействован Механизм трансформаций данных https://github.com/cujojs/wire/blob/master/docs/connections.md#transform-connections - компоненты освобождаются от логики совмещения данных, этим занимаются трансформации. Профит - в разделении логики и облегчении unit-тестирования.

Примеры плагинов

Нужно начать с того, что cola.js, входящая в cujo.js ToolKit, по существу, является одним большим плагином для wire.

- https://github.com/cujojs/cola/blob/dev/cola.js#L83-L112

- https://github.com/conversocial/bbRouter - backbone-роутер (плагин еще очень сырой, но в переработанном виде может быть использован для работы с routeMap)
- авторский плагин, - "A recent wire router plugin sample", в качестве gist https://gist.github.com/briancavalier/5970232
- https://github.com/gnesher/bbevents - backbone-события (тоже сырой, но пока нам просто важны разные подходы)
- Крайне полезный плагин для работы с backbone.events listenTo.js - пример использования в ветке дискуссии о msgs.js https://groups.google.com/d/msg/cujojs/_zT9H-GN1Jg/58DTEDR6ypcJ и в gist https://gist.github.com/eschwartz/8188988
  Как видно из gist, плагин позволяет триггерить событие простым локальным view.trigger("someevent", data) вместо необходимости использовать внешние объекты vent или context, как мы это делаем сейчас (благодаря https://gist.github.com/eschwartz/8188988#file-listento-js-L51).
  Также нет необходимости подписываться внутри объекта-слушателя на событие при помощи .on и писать _.bindAll [массив функций] для их привязки к scope данного объекта. В результате межобъектные pub-sub взаимодействия описываются в wire-спецификации - что ведет к очищению кода и выразительности.
  (Кстати, из кода плагина становится понятно, что facet options могут быть трактованы как "спецификация в спецификации" и превращены с помощью функции wire в соответствующий контекст:
  https://gist.github.com/eschwartz/8188988#file-listento-js-L17, https://gist.github.com/eschwartz/8188988#file-listento-js-L31-L33)
- В cujo.js toolkit входит pubsub channels system msgs.js - и есть соответствующий плагин в dev-branch https://github.com/cujojs/msgs/blob/dev/wire.js для работы в wire-спецификациях.
- Простейший плагин для работы с темплейтами (hbs, text) https://github.com/skiadas/example-cujo-guide/blob/master/app/plugins/hbs.js

Routing

Одна из ключевых особенностей нынешней реализации ядра - конфигурация всего приложения про помощи routeMap.

(см. Core и загрузка модулей)

Аналогичное решение, базирующееся на wire.js и backbone.js, может быть представлено на основе wire-плагина - https://github.com/conversocial/bbRouter.

Загрузка контролов, доступ к public API контрола

- В настоящее время загрузка контрола производится в ItemView. Сумма таких ItemView составляет CompositeView. В результате ItemView является sandbox'ом (оберткой) для контрола.

Контрол предоставляет метод publicAPI, возвращающий объект, поля которого соответствуют публичным методам (или, точней говоря, методы, которые мы хотим представить как публичные - это не значит, что остальные методы представления не могут быть вызваны - из-за отсутствия в js модификаторов доступа). Вопрос - есть ли смысл в таком sandbox'е?

Коллекция моделей контролов, полученная из description модуля, инстанциируется (превращается в реальные объекты соответствующих классов) как раз при создании collection (composite) view из коллекции.

Т.е. в настоящий момент мы имеем смешение логики отображения, в контексте которой находится ItemView, и логики загрузки-привязки к экземпляру объекта (resolver).

Было бы правильнее сначала выполнять загрузку с помощью wire, и затем передавать контекст-провайдер в контроллер, выполняющий наполнение коллекции.

Это достигается выстраиванием иерархии wire-контекстов (см. ниже)

- В настоящее время контрол - это Marionette.ItemView либо Marionette.Layout, и часть внешних, не всегда очевидных операций, связанных именно с этим обстоятельством (rendering service, AOP взаимодействие).

Лучше постараться уйти от данного ограничения.

В стратегическом плане было ошибкой создавать коллекцию контролов, зависящих от ядра. Хотя эти зависимости и сведены к минимуму, требуется время для адаптации контролов к изменяющейся архитектуре. Нужно сформулировать, использования каких именно библиотек при разработке и тестировании контролов нельзя избежать: например, backbone, backbone.marionette не избегаем и не собираемся. Но ядро, в сущности, превращающееся в надстройку над marionette, не должно быть задействовано при разработке (и тестировании!), контролы должны уметь работать и в другом, отличном от исходного, окружении.

Ненавязчивость wire.js и тестирование

Wire.js не требует включения себя в качестве AMD-модуля, что является плюсом при написании и тестировании модулей бизнес-логики.

Вместо AMD-зависимости и, как следствие, загрузки wire, можно инъектировать wire! как функцию.

https://github.com/cujojs/wire/blob/master/docs/wire.md#injecting-wire

В двух словах, wire представляет собой платформу загрузки и инициализации (bootstrap).

Один из ключевых принципов wire - за рамками bootstrap'а код приложения никогда не должен иметь жесткой зависимости от wire. Например, модули, реализующие бизнес-логику, ни в коем случае не должны указывать wire в списке define- (require-) зависимостей.

wire!, инъектированная как функция, в тестах может быть заменена на fake function.

Таким образом, при разработке приложения, крайне важным становится разделение приложения на тестируемые по отдельности смысловые части. Основной акцент должен быть сделан на тестирование ключевых методов объектов, т.е. методов, поставляющих данные для коннектов - объекты в дальнейшем будут связаны посредством wire-connections: https://github.com/cujojs/wire/blob/master/docs/concepts.md#connections.

(при необходимости данные могут быть трансформированы, см. выше, а также https://github.com/cujojs/wire/blob/master/docs/connections.md#transform-connections)

Кроме того, авторы cujo.js справедливо настаивают на минимализации зависимостей в unit-тестах, см. ветку в cujojs google group: https://groups.google.com/forum/#!msg/cujojs/1aoTzffFkhU/Z33OC7fdtg0J

Создать минималистичное тестовое окружение возможно и при тестировании UI. В приведенном примере авторы предлагают стратегию тестирования "крупнозернистых" ui-компонентов в iframe. При таком подходе нет необходимости загружать все приложение: https://github.com/know-cujojs/monty-hall/blob/master/client/test/ui/launch.html (запуск), https://github.com/know-cujojs/monty-hall/blob/master/client/app/instructions/test/harness.html (что тестируем)

Также отметим полезную возможность перезагружать контексты:

перезагрузка контекста

```
wire([mainSpec, overridesSpec]).then(function(context) { ... });
```

Тестирование в node.js-окружении

Разработчики wire.js используют Buster.js для тестирования браузеро-независимого кода непосредственно в node.js.

Собственно, о платформе Buster.js должно быть написано отдельное исследование, пока ограничимся ссылкой на пример тестирования wire-контекста:

https://github.com/cujojs/wire/blob/1a9861e404b6e5e79fa355366f5b011d71dc8595/test/node/lib/plugin/basePlugin-test.js#L94-L122 (весь файл любопытен для понимания wire).

Примечание. Для тестирования в авторском окружении (т.е. на buster -v 0.6) нужно исправить bugfix, как написано в issue https://github.com/busterjs/buster/issues/363

См. также issue-ветку https://github.com/cujojs/wire/issues/61

(см. также https://github.com/briancavalier/hello-wire-node)

Заслуживает внимания авторский инструмент для генерации значений в тестах: https://github.com/briancavalier/gent

Тестирование в браузере при помощи jasmine

Хорошая новость: вышел jasmine-2.0.0, в котором поддерживается асинхронный запуск тестов с функцией done() (по аналогии с mocha.js).

Пример тестирования загруженных wire компонентов

```
define [
    "wire"
], (wire) ->
    describe "After wire context created", ->
        beforeEach (done) ->
            wire({
                rootComponent: {wire: "core/modules/root/rootModuleSpec"}
            }).then (@ctx) =>
                done()
            .otherwise (err) ->
                console.log "ERROR", err
        it "rootComponent", (done) ->
            expect(@ctx.rootComponent).toBeDefined()
            done()
```

В новом jasmine THIS в beforeEach изначально - empty object, который передается далее в it-блок и может быть использован как транспортный объект.

При переводе предыдущей require-based платформы на jasmine-2.0.0 (см. инструкцию https://groups.google.com/forum/#!topic/jasmine-js/SgjHOUoFG58, а также https://www.packtpub.com/sites/default/files/downloads/7204OS_The_Future_Jasmine_2_0.pdf - "Migration") Jasmine-Matchers в текущей версии не работает из коробки - можно запускать с https://github.com/testdouble/jasmine-matcher-wrapper

Контекстуализация

Ссылка на { $ref: 'wire!' } внедряет wire-функцию, которая привязана к текущему wire-контексту (https://github.com/cujojs/wire/blob/master/docs/concepts.md#contexts).

Благодаря легкости выстраивания иерархии контекстов (https://github.com/cujojs/wire/blob/master/docs/concepts.md#context-hierarchy), приложение приобретает высокий уровень модульности:

https://github.com/cujojs/wire/blob/master/docs/wire.md#contextual-ness

Можно переопределять базовые контексты (происходит context merge):

https://gist.github.com/briancavalier/5388378

Экспорт сложных компонентов

Для сложных компонентов может использоваться техника экспорта.

Сложный компонент декларируется собственной спецификацией, допустим, он состоит из двух простых компонентов rootModule и viewFake

core/modules/root/rootModuleSpec

```
define
    $exports:
        rootModule: {$ref: 'rootModule'}
        viewFake: {$ref: 'view'}

    rootModule:
        create:
            module: "core/modules/root/rootModule"
            properties:
                view: {$ref: 'view'}
            ready: 'view.play'
    view:
        create:
            module: "core/modules/root/viewFake"
```

Благодаря определению в ключе $exports, мы можем получить доступ к rootModule и viewFake в parent context посредством использования wire factory:

parent context

```
define
 rootComponent: {wire: "core/modules/root/rootModuleSpec"}
```

и распоряжаться ими по своему усмотрению:

parent context

```
define
    rootComponent: {wire: "core/modules/root/rootModuleSpec"}
    one:
        create:
            module: "core/modules/root/one"
        properties:
            view: {$ref: 'rootComponent.viewFake'}
        ready:
            insertView: {}

    $plugins:[
        "wire/debug"
    ]
```

https://github.com/cujojs/wire/blob/master/docs/components.md#exports-example

В приведенном примере для привязки child-контекста использовался короткий синтакс factory. Развернутый выглядит примерно так:

привязка child-контекста в parent-контексте

```
childContext: {
 wire: {
  spec: 'my/child/spec',
  // defer и waitParent взаимоисключающие директивы!
  // но defer имеет приоритет
  // привязываем ли тут же child-контекст
  defer: false, /* default */
  // если true, не привязываем child-контекст, пока не будет сформирован parent-контекст
  waitParent: false, /* default */
  // в child-контекст провайдятся компоненты и значения из текущего контекста
  provide: {
   aSpecialValue: 42,
   transform: { $ref: 'myTransform' }
  }
 }
}
```

https://github.com/cujojs/wire/blob/master/docs/components.md#wire

Перехват и обработка ошибок и исключений

С wire, так же как и с promise library when, перехват ошибок осуществляется в reject-функциях.

К изучению - дискуссия

https://github.com/cujojs/wire/issues/137

Вместо эпилога. Медиаторы, бизнес-логика - wire it!

Привлекательность wire.js в декларативности, в возможности выстраивания гибких связей между компонентами.

Спецификации, написанные в стиле Wire.js представляют собой красноречивый и лаконичный чертеж. Необходимости в подробном изучении кода объектов, участвующих в процессе, чтобы схватить суть действия, при грамотно сконструированной wire-спецификации должна уходить на второй план.

Сама реализация находится уровнем ниже и не отвлекает "множеством букв".

Разумеется, это работает только в случае тщательно подобранных имен объектов и переменных, в случае наличия самодокументированного кода (http://martinfowler.com/bliki/CodeAsDocumentation.html) и продуманного взаимодействия.

. . . . . .

(продолжение следует)

Примеры использования wire в сложных проектах

Хорошим примером использования wire и кастомных плагинов может служить проект https://github.com/eschwartz/aerisjs, использующий стек wire, requireJS, backbone, marionette, handlebars, jquery и др.

и в простых

https://github.com/know-cujojs/contacts/tree/master/app (пример с сайта http://cujojs.com/)

https://github.com/tastejs/todomvc/tree/gh-pages/labs/architecture-examples/cujo/app (todoMVC)

https://github.com/briancavalier/notes-demo-dojoconf-2011

P.S.

Вот здесь один человек из cujo-сообщества публикует наброски для учебника по wire: http://gehan.github.io/seed/. Обсуждение здесь: https://groups.google.com/forum/#!topic/cujojs/69qu-2e-_dY