示例Explainer文档- [微软]

示例Explainer2 -https://github.com/opencap/protocol

节选

1. Why no blockchain?不希望把无用数据（容许篡改）上在chain
1. We didn't want to bloat an existing blockchain with superflous data that isn't directly related to transactions and balances.
2. OpenCAP leaves the level of centralization up to the users. Some will use third-party OpenCAP providers. Others will run their own OpenCAP servers for complete control.
3. In many cases it isn't desirable for an alias system to be immutable. OpenCAP leaves this up to the implementation.
4. OpenCAP can achieve much faster create/update/delete operations than data stored on a blockchain, which means an alias can be changed quickly over time if desired.

First found ~7thApr19 (github or ipfs代码揭秘）, didn’t save original example, just the link to below definition

EXPLAINERS

Status: PUBLIC
Last Updated: 2017-07-26

Introduction:

Your explainer is a living document that you should consider a design-doc in the early phase of design. As you iterate on the early design, the explainer content may change drastically, which is a key reason to start the design process with an explainer and not a formal spec document.

While specs are intended for implementers, the explainer should address a web developer audience. The explainer should outline what need the design meets rather than dwelling on algorithms and minutia.

Benefits of Explainers:

- Historically you may have written a “requirements” document; a good explainer can subsume that by focusing on user needs.
- The explainer can *change more quickly than a spec can*. This is important because your design *will* change. Changing an explainer to accommodate a design change is much less work than changing a spec and increases your agility.
- Web developers can get excited about Explainers! Unlike specs, they include terse descriptions and jargon-free text that allows web developers (your most important constituency) to get excited by your proposal. After the spec is written the explainer can become a reference point for developers and an aid to doc writers.
- Writing down your key use cases helps to focus the design process and minimise scope-creep. It also helps readers understand what problem you’re solving.
- It is a useful companion even after you have a formal spec. When you eventually add a formal spec, your explainer shifts in value: instead of helping to organize your design thoughts, it helps *others* understand key points of the design, including spec reviewers and interested web developers. Long-term maintenance of explainers is encouraged.
- Explainers help you to avoid premature specificity. Spec text forces you to write down fine-grained details well before you're ready to commit to them.

Web Platform work often suffers from *premature ossification*. Receiving too few iterations on a design is one of the primary reasons features fail. Specs are artifacts of *finished, complete designs.* Explainers are artifacts of the *early and mid-stage design* process and are designed to be light-weight enough to enable (rather than prevent) wholesale iteration. If your design is new-ish (hasn't shipped and isn't fully implemented), prefer an Explainer to a formal spec document.

Content:

Like other design docs, your explainer should:

- Introduce the problem and explain the motivation for solving it
- State assumptions
- Provide evidence or argument that this problem is important to solve

- List goals
- List explicit non-goals
- Discuss key use-cases - it has to cover the motivating examples you're considering in your design
- Propose a solution
- Provide example code that ties together problem, proposed solution and how it enables the listed use-cases
- Discuss alternative designs that were considered (or may be explored in future iterations) with pros/cons. You might want to include the rationale for any controversial decisions you made, for example in tradeoffs between functionality and security..

Example code is your primary mechanism for demonstrating features of the API or design. Avoid IDL in explainers.

Examples:

https://github.com/w3c/ServiceWorker/blob/master/explainer.md
https://github.com/zkoch/paymentrequest/blob/gh-pages/docs/explainer.md
https://github.com/WICG/web-share/blob/master/docs/explainer.md
https://github.com/WICG/ViewportAPI/blob/gh-pages/README.md
https://github.com/WICG/EventListenerOptions/blob/gh-pages/explainer.md
https://github.com/WICG/IntersectionObserver/blob/gh-pages/explainer.md (although this one includes IDL, which explainers should not)

Explainer Template:

# Web Frobulators Explained

<last update date>

## What's all this then?

A brief, 4-5 paragraph explanation of the feature's value. Outline what the feature does and how it accomplishes those goals (in prose). If your feature creates UI, this is a great place to show mocks and user flows.

### Goals

How will the web be better when this feature launches? And who will it help?

### Non-goals

You're not going to solve every problem so enumerate the attractive, nearby problems that are out of scope for this effort. This may include details on the tradeoffs made due to architectural limitations made due to implementation details, and features left out either due to interoperability concerns or other hurdles, and how you plan to improve on this. This can often be the single most important part of your document, so give it careful thought.

## Getting started / example code

Provide a terse example for the most common use case of the feature. If you need to show how to get the feature set up (initialized, or using permissions, etc.), include that too

<here>

## Key scenarios

Next, discuss the key scenarios which move beyond the most canonical example, showing how they are addressed using example code:

### Scenario 1

Outline the scenario, then provide:

<sample code that demonstrates the feature>

### Scenario 2

Outline the scenario, then provide:

<sample code that demonstrates the feature>

…

## Detailed design discussion

### Tricky design choice #1

Talk through the tradeoffs in coming to the specific design point you want to make, hopefully:

<illustrated with example code>

…

### Tricky design choice N

…

## Considered alternatives

One of the most important things you can do in your design process is to catalog the set of roads not taken. As you iterate on your design, you may find that major choices in your approach or API style will be revisited and enumerating the full space of alternatives can help you apply one (or more) of them later, may serve as a “graveyard” for u-turns in your design, and can give reviewers and potential users confidence that you’ve got your ducks in a row.

## References & acknowledgements

Your design will change and be informed by many people; acknowledge them in an ongoing way! It helps build community and, as we only get by through the contributions of many, is only fair.