С орбитальной высоты

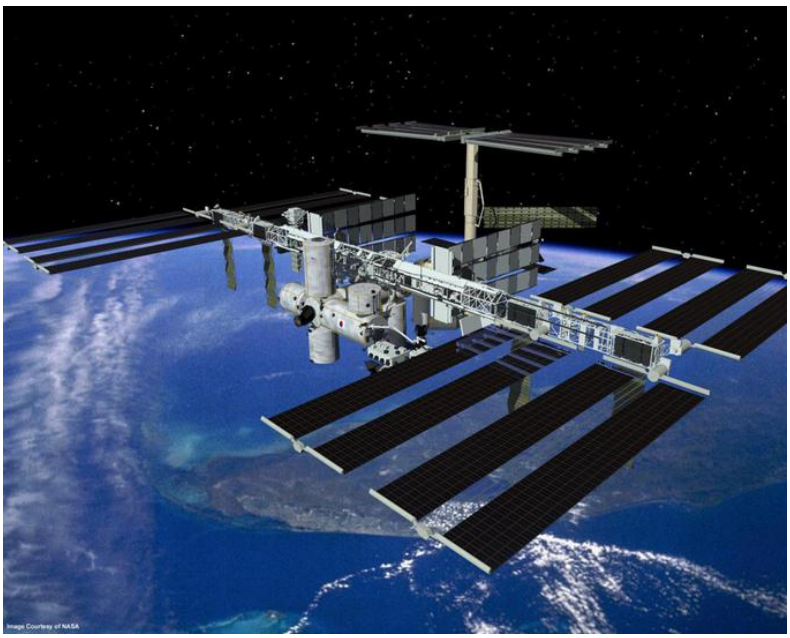
Этот раздел описывает набор современных стандартов для «веб-компонентов».

На текущий момент, эти стандарты находятся в процессе разработки. Некоторые фичи имеют хорошую поддержку и интеграцию в современный стандарт HTML/DOM, в то время как другие пока ещё в черновиках. Вы можете попробовать примеры в любом современном браузере (Google Chrome, скорее всего, имеет наиболее полную поддержку, так как ребята из Google стоят за большинством спецификаций по этой теме).

Что общего между...

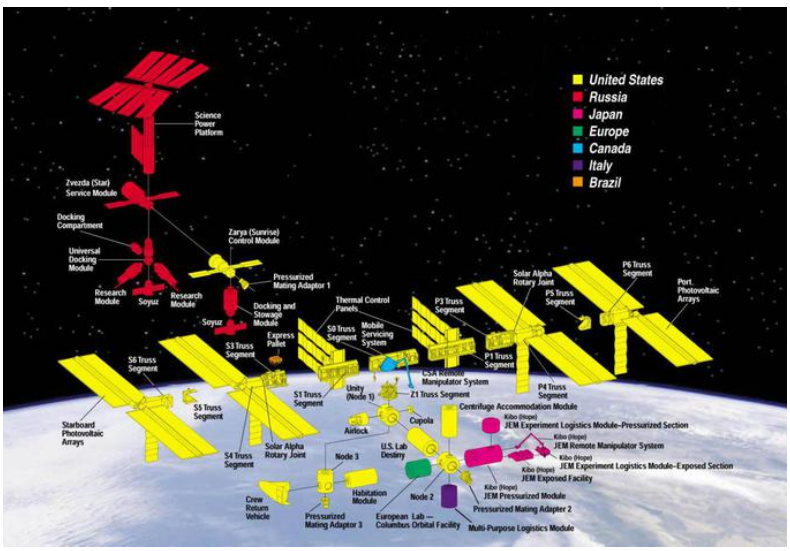
В идее самостоятельного компонента нет ничего нового. Такой подход используется во многих фреймворках.

Прежде чем мы погрузимся в детали реализации, взгляните на это великое достижение человечества:



Это международная космическая станция (МКС).

А это то, как она устроена (приблизительно):



Международная космическая станция:

- Состоит из множества компонентов.
- Каждый компонент в свою очередь, состоит из множества более мелких деталей.
- Компоненты имеют очень сложное устройство, и гораздо сложнее большинства сайтов.
- Компоненты разработаны на международной основе, командами из разных стран и говорящих на разных языках.

...И эта штука летает, поддерживая жизни людей в космосе!

Как создаются столь сложные устройства?

Какие принципы мы могли бы позаимствовать, чтобы сделать нашу разработку такой же надёжной и масштабируемой? Или, по крайней мере, приблизиться к такому уровню.

Компонентная архитектура

Хорошо известное правило разработки сложного программного обеспечения гласит: не создавай сложное программное обеспечение.

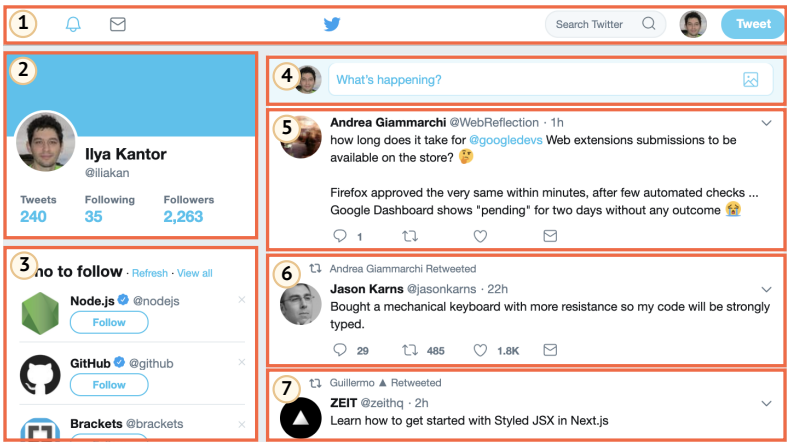
Если что-то становится сложным – раздели это на более простые части и соедини наиболее очевидным способом.

Хороший архитектор – это тот, кто может сделать сложное простым.

Мы можем разделить пользовательский интерфейс на визуальные компоненты: каждый из них занимает своё место на странице, выполняет определённую задачу, и отделен от остальных.

Рассмотрим какой-нибудь сайт, например Twitter.

Он естественным образом разделён на компоненты:



1. Верхняя навигация.
2. Данные пользователя.
3. Предложения подписаться.
4. Форма отправки сообщения.
5. (а так же 6 и 7) – сообщения.

Компоненты могут содержать подкомпоненты, например сообщения могут быть частями родительского компонента «список сообщений». Кликабельное фото пользователя может быть самостоятельным компонентом и т.д.

Как мы определяем, что является компонентом? Это приходит из соображений здравого смысла, а также с интуицией и опытом. Обычно это объект, отделимый визуально, который мы можем описать с точки зрения того, что он делает и как он взаимодействует со страницей. В примере выше, страница содержит блоки, каждый из которых играет свою роль, и логично выделить их в компоненты.

Компонент имеет:

- свой собственный JavaScript-класс.

- DOM-структура управляется исключительно своим классом, и внешний код не имеет к ней доступа (принцип «инкапсуляции»).
- CSS-стили, применённые к компоненту.
- API: события, методы класса и т.п., для взаимодействия с другими компонентами.

Ещё раз заметим, в компонентном подходе как таковом нет ничего особенного.

Существует множество фреймворков и методов разработки для их создания, каждый из которых со своими плюсами и минусами. Обычно особые CSS классы и соглашения используются для эмуляции компонентов – области видимости CSS и инкапсуляция DOM.

«Веб-компоненты» предоставляют встроенные возможности браузера для этого, поэтому нам больше не нужно эмулировать их.

- Пользовательские элементы – для определения пользовательских HTML-элементов.
- Теневой DOM – для создания внутреннего DOM компонента, скрытого от остальных.
- Области видимости CSS – для определения стилей, которые применяются только внутри теневого DOM компонента.
- Перенаправление событий и другие мелочи для создания более удобных в разработке пользовательских компонентов.

В следующей главе мы погрузимся в «пользовательские элементы» – фундаментальную для веб-компонентов технологию, имеющую хорошую поддержку в браузерах.

перед тем как писать...

Комментарии