Abordagem

Aprenda sobre o princípio deste guia, estratégias e técnicas usadas para construir e manter o Bootstrap de forma que você possa extender e customizá-lo mais facilmente.

Enquanto as páginas *getting started* servem como um *tour* introdutório do projeto, esta foca no *porquê* de fazermos as coisas da maneira que fazemos, no Bootstrap. Aqui, explicaremos nossa filosofia para desenvolver na web de maneira que outros possam aprender conosco, contribuir e nos ajudar a melhorar.

Viu alguma coisa que não soou bem ou, talvez, poderia ser feita melhor? Abra um issue e nós amaremos discutir com você!

Sumário

Nós vamos mergulhar de cabeça em cada um desses, mais a frente. No entanto, em resumo, isso resume nossa abordagem:

- Componentes devem ser responsivos e mobile-first;
- Componentes devem ser construídos com uma classe base e extendidos através de uma modificadora;
- Estados de componentes devem seguir um padrão de escala z-index;
- Sempre que possível, prefira uma implementação de HTML e CSS via JavaScript;
- Sempre que possível, use utilitários invés de estilos customizados;
- Sempre que possível, evite o uso de requisitos rígidos do HTML (seletor de filhos).

Responsividade

Os estilos responsivos do Bootstrap são construídos para serem responsivos, usando uma abordagem denominada *mobile-first*. Nós usamos esse termo em nossos documentos e, fortemente, concordamos com ele (apesar de algumas vezes ser complexo). Enquanto nem todos componentes *precisam* ser responsivos no Bootstrap, essa abordagem responsiva ajuda a reduzir sobrescrição de CSS, te forçando a adicionar estilos de acordo com o crescimendo da *viewport*.

Bootstrap adentro, você vai ver isso mais claramente com nossas *media queries*. Na maioria dos casos, nós usamos *media queries* com `min-width` e que começam em um breakpoint específico e vão aumentando até outro breakpoint. Por exemplo, um seletor `.d-none` começa em `min-width: 0` e escala infinitamente. Por outro lado, um seletor `d-md-none` começa a escalar à partir de um breakpoint mediano.

Algumas vezes, vamos usar `max-width` quando um dado componente precisar. Às vezes, essas sobrescrições são funcional e mentalmente mais fáceis de implementar e manter, invés de reescrever as funcionalidades principais dos nossos componentes. Nós tentamos limitar essa abordagem, mas nós vamos usá-la de tempo em tempo.

Classes

Excluindo o Reboot, nosso *reset* cross-browser, todos nossos estilos visam usar classes como seletores. Isso significa que ficamos longe dos seletores de tipo (ex: `input[type="text"]`) e classes pais questionáveis (ex: `.pai .filho`) que fazem os estilos muito específicos, para que seja facil sobrescrever.

De tal maneira, componentes devem ser construídos com uma classe base para propriedades e valores comuns que não serão sobrescritos. Por exemplo, `.btn` e `.btn-primary`. Nós usamos `.btn` para todos os estilos em comum como `display`, `padding` e `border-width`. Então, usamos modificadores como `.btn-primary` para adicionar a `color`, `background-color`, `border-color` e etc.

Modificadores de classes devem apenas serem usados quando existirem diversas propriedades ou valores para serem alterados, em diversas variáveis. Modificadores não são sempre necessários, então, tenha certeza que você está realmente economizando linhas de código e evitando sobrescrições desnecessárias, quando for criá-los. Bons exemplos de modificadores são nossas classes de cores e as variantes de tamanhos.

Escalas z-index

Existem duas escalas `z-index` no Bootstrap: uma para elementos dentro de um componente e outra para sobreposição de componentes.

Elementos de componentes

- Alguns componentes no Bootstrap são construídos com elementos sobrepostos para evitar borda dupla, sem modificar a propriedade `border`. Por exemplo: grupos de botões, grupos de inputs e paginação;
- Esses componentes compartilham uma escala `z-index` padrão de `0` até `3`;
- `0` é o padrão (valor inicial), `1` é `:hover`, `2` é `:active`/`.active` e `3` é `:focus`;
- Essa abordagem corresponde as nossas expectativas de maior prioridades do usuário. Se um elemento está focado, está à vista e recebendo atenção do usuário. Já elementos ativos são os segundos maiores porque indicam estado. Hover é o terceiro maior porque indica intenção de usuários, apesar que quase *nada* usa `:hover`.

Componentes de sobreposição

Bootstrap possui vários componentes que funcionam como uma sobreposição de algum tipo. Isso inclui (de acordo com maior `z-index`): dropdowns, navbar fixas e sticky, modais, tooltips e popovers. Estes componentes possuem as próprias escalas `z-index` que começam em `1000`. Esse número inicial é aleatório e serve como um pequeno espaçamento entre nossos estilos e os estilos de seus próprios projetos.

Cada componente de sobreposição possui seu valor `z-index` um pouco maior, de modo que princípios de UI comuns permitem que elementos com foco ou `:hover` estejam visíveis sempre. Por exemplo, um modal é um bloqueador de documentos (você não pode fazer mais nada, além de usar o modal), então, nós criamos ele com valor acima das navbars.

Aprenda mais sobre isso na nossa página de layout `z-index`.

HTML e CSS invés de JS

Sempre que possível, nós preferimos escrever HTML e CSS invés de JavaScript. No geral, HTML e CSS são mais presentes e acessíveis para pessoas de diferentes níveis de experiência. HTML e CSS também são mais rápidos que JavaScript e seu navegador, geralmente, provê uma boa gama de funcionalidades para você.

Nesse princípio, nossa API JavaScript principal é o atributo `data`. Você não precisa escrever quase nenhum JavaScript para usar nossos plugins JS, mas sim HTML. Leia mais sobre isso na nossa página de visão geral JavaScript.

Por último, nossos estilos ajudam a desenvolver baseados em comportamentos fundamentais de elementos comuns na web. Sempre que possível, nós preferimos usar o que o navegador provê. Por exemplo, você pode colocar uma classe `.btn` em quase qualquer elemento, mas a maioria dos elementos não possuem valores semânticos ou funcionalidades em browsers. Desse modo, pelo contrário, nós usamos `<button>` e `<a>`.

O mesmo acontece em componentes mais complexos. Enquanto nós *poderíamos* escrever nossos próprios plugins de validações de formulários para adicionarem classem a elementos pais (baseados em um estado de input) e, portanto, nos permitindo estilizar um texto em vermelho, nós preferimos usar os pseudo-elementos `:valid`/`:invalid` que todo browser suporta.

Utilitários

Classes "utilitárias" (antes "ajudantes", no Bootstrap 3) são poderosos aliados no combate ao CSS repetitivo e páginas pouco performáticas. Uma classe utilitária é, tipicamente, um único e imutável par propriedade-valor expresso como uma classe (ex: `.d-block` representa `display: block;`). A melhor vantagem delas é velocidade de uso quando escrevendo HTML e limitando a quantidade de CSS customizados que você tem que escrever.

Especificamente, no que diz a respeito de CSS customizado, utilitários podem combater o crescimento de tamanho dos arquivos, reduzindo suas propriedades e valores mais usados à classes únicas. Isso pode ter um efeito incrível, nos seus projetos em escala.

HTML Flexível

Apesar de nem sempre ser possível, nós tentamos evitar sermos rígidos em nossos requisitos HTML para componentes. Portanto, nós focamos em classes únicas nos nossos seletores CSS, além de evitar seletores de filhos imediatos (`>`). Isto lhe dá mais flexibilidade em sua implementação e ajuda manter nosso CSS mais simples e menos específico.