1.6 Большие списки

Большие списки

В большинстве случаев для отображения списков можно использовать элементы

управления элементов,

Column

и , но если список содержит сотни или тысячи

и становятся неэффективными, так как они отрисовывают

Row

Row

Column

все элементы сразу, даже если они не видны на текущей позиции прокрутки.

В следующем примере мы добавляем 5000 текстовых элементов управления на

страницу. Страница использует в качестве контейнера по умолчанию:

Column

import flet as ft

def main(page: ft.Page):

for i in range(5000):

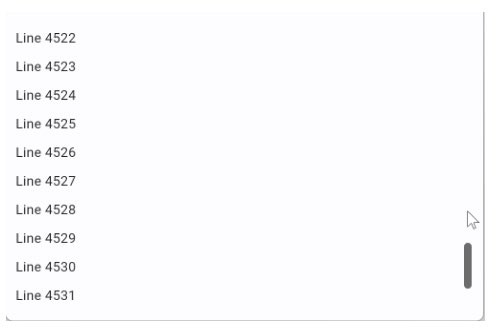
page.controls.append(ft.Text(f"Line {i}"))

page.scroll = "always"

page.update()

ft.app(target=main, view=ft.WEB\_BROWSER)

Запустите программу и обратите внимание, что не только требуется несколько секунд, чтобы первоначально загрузить и отрисовать все текстовые строки на странице, но и прокрутка медленная и "лагает".

Для отображения списков с большим количеством элементов используйте элементы

управления и GridView , которые отрисовывают элементы по

ListView

требованию, видимые только на текущей позиции прокрутки.

# ListView

может быть вертикальным (по умолчанию) или горизонтальным. отображаются один за другим в направлении прокрутки.

|  |  |
| --- | --- |
| ListView |  |
| Элементы | ListView |

уже реализует эффективную отрисовку своих дочерних элементов по

ListView

требованию, но производительность прокрутки может быть дополнительно улучшена, если вы можете установить одинаковую фиксированную высоту или ширину (для горизонтального ListView ) для всех элементов ("*extent"*).

Это можно сделать, установив абсолютный размер с помощью

свойства или сделав размер всех дочерних элементов равным размеру

item\_extent

первого дочернего элемента, установив в значение True .

first\_item\_prototype

Давайте выведем список из 5000 элементов с помощью элемента управления ListView:

import flet as ft

def main(page: ft.Page):

lv = ft.ListView(expand=True, spacing=10)

for i in range(5000):

lv.controls.append(ft.Text(f"Line {i}"))

page.add(lv)

ft.app(target=main, view=ft.WEB\_BROWSER)

Теперь прокрутка происходит плавно и быстро.

*ПРИМЕЧАНИЕ*

*Мы использовали в конструкторе ListView . Для корректной*

*expand=True*

*работы ListView должен быть указан размер по высоте (или ширине,*

*если ListView горизонтальный). Вы можете задать абсолютный размер,*

*например, ListView(height=300, spacing=10) , но в приведенном выше примере*

*мы заставляем занимать все доступное пространство на странице, то*

*ListView*

*есть, мы используем свойство expand .*

# GridView

позволяет располагать элементы управления в прокручиваемой сетке.

GridView

Вы можете создать "сетку" с помощью

ft.Column(wrap=True)

или ft.Row(wrap=True) , например:

import os

import flet as ft

os.environ["FLET\_WS\_MAX\_MESSAGE\_SIZE"] = "8000000"

def main(page: ft.Page):

r = ft.Row(wrap=True, scroll="always", expand=True)

page.add(r)

for i in range(5000):

r.controls.append(

ft.Container(

ft.Text(f"Элемент {i}"),

width=100,

height=100,

alignment=ft.alignment.center,

bgcolor=ft.colors.AMBER\_100,

border=ft.border.all(1, ft.colors.AMBER\_400),

border\_radius=ft.border\_radius.all(5),

)

)

page.update()

ft.app(target=main, view=ft.WEB\_BROWSER)

Попробуйте прокрутить и изменить размер окна браузера - все работает, но очень медленно.

*ПРИМЕЧАНИЕ*

*В начале программы мы устанавливаем значение переменной*

*FLET\_WS\_MAX\_MESSAGE\_SIZE*

*8000000*

*окружения*

*равное*

*- это максимальный*

*размер сообщения WebSocket в байтах, который может быть получен сервером Flet при отрисовке страницы. Размер по умолчанию - 1 МБ, но размер сообщения JSON,*

*описывающего 5000 контейнеров, превысит 1 МБ, поэтому мы увеличиваем допустимый размер до 8 МБ.*

*Передача больших сообщений через канал WebSocket, в общем, не является хорошей идеей, поэтому используйте подход с пакетными обновлениями для управления нагрузкой на канал.*

GridView , подобно ListView , очень эффективен для отображения большого количества дочерних элементов. Давайте реализуем пример выше с использованием GridView :

import os

import flet as ft

os.environ["FLET\_WS\_MAX\_MESSAGE\_SIZE"] = "8000000"

def main(page: ft.Page):

gv = ft.GridView(expand=True, max\_extent=150,

child\_aspect\_ratio=1)

page.add(gv)

for i in range(5000):

gv.controls.append(

ft.Container(

ft.Text(f"Элемент {i}"),

alignment=ft.alignment.center,

bgcolor=ft.colors.AMBER\_100,

border=ft.border.all(1, ft.colors.AMBER\_400),

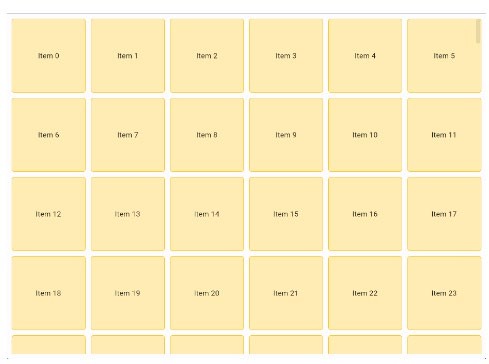
border\_radius=ft.border\_radius.all(5),

)

)

page.update()

ft.app(target=main, view=ft.WEB\_BROWSER)

С помощью

GridView

и отзывчивыми!

прокрутка и изменение размеров окна становятся плавными

Вы можете задать как фиксированное количество строк или столбцов (прогонов) с

помощью свойства , так и максимальный размер "плитки" с помощью

runs\_count

свойства max\_extent , благодаря чему количество прогонов может изменяться автоматически. В нашем примере мы установили максимальный размер плитки 150 пикселей и задали ее форму "квадрат"

с child\_aspect\_ratio=1 .

child\_aspect\_ratio

- это отношение поперечной оси к

протяженности главной оси каждого дочернего элемента. Попробуйте изменить его на 0,5 или 2.

# Пакетные обновления

При вызове функции на Flet-сервер по WebSockets отправляется

page.update()

сообщение, содержащее обновления страницы с момента последнего обновления page.update() . Отправка большого сообщения с тысячами

добавленных элементов управления может заставить пользователя ждать несколько секунд, пока сообщение будет полностью получено и элементы управления отрендерены.

Для повышения удобства работы с программой и скорейшего представления результатов пользователю можно отправлять обновления страницы пакетно. Например, следующая программа добавляет 5100 дочерних элементов управления

в партиями по 500 элементов:

ListView

import flet as ft

def main(page: ft.Page):

# сначала добавляем ListView на страницу

lv = ft.ListView(expand=1, spacing=10, item\_extent=50)

page.add(lv)

for i in range(5100):

lv.controls.append(ft.Text(f"Line {i}"))

# отправвляем page на страницу

if i % 500 == 0:

page.update()

# отправляем остальное на страницу

page.update()

ft.app(target=main, view=ft.AppView.WEB\_BROWSER)

