第2章　Layoutを使ってみよう

2.1　なぜLayoutを使うのか

第1章ではボタンあるいはラベルがウィンドウに一つしか存在しませんでした。では、次にボタンを二つ並べて表示してみましょう。次のようなコードを書いたとします。

```
1   #include <QApplication>
2   #include <QPushButton>
3
4   int main(int argc, char** argv)
5   {
6           QApplication app(argc, argv);
7           QPushButton* button = new QPushButton("Hello Qt!");
8           QPushButton* button2 = new QPushButton("Goodbye");
9           button->show();
10          button2->show();
11          return app.exec();
12  }
```

このコードを実行すると、図2.1のようになります。

図 2.1: 実行結果 (on Windows)

本当はボタンが二つ並んで一つのウィンドウに収まってくれることを願っていましたが、これは予想外の結果です。

実は、一つのウィンドウには一つの部品しか表示[1]できないのです。[2]この一つの部品を親としましょう。それではどのようにして複数のボタンを作ったらよいのでしょうか。

その方法は、一つの親を作りその中にどんどん部品を追加していけばよいのです。この追加した部品を子とします。そうすれば、親の部品を表示することにより子も一緒に表示されます。

この親となるのが MainWindow であり、その中に Layout をセットします。Layout は部品をどのように配置するかを管理するクラスです。

2.2 基本的な Layout の使い方

Layout にはいくつかの種類が存在します。最も良く使われるのが、QHBoxLayout と QVBoxLayout でしょう。QHBoxLayout の H は Horizontal、QVBoxLayout の V は Vertical であり、水平と垂直を意味しています。

実際使ってみたほうがわかりやすいので、とりあえず使い方を見てみましょう。まずは QHBoxLayout に 3 つのボタンを追加し、その Layout をメインウィンドウに表示させてみましょう。

```
1   #include <QApplication>
2   #include <QPushButton>
3   #include <QHBoxLayout>
4
5   int main(int argc, char** argv)
6   {
7           QApplication app(argc, argv);
8           QWidget* window = new QWidget;
9           QPushButton* buttonA = new QPushButton("Button A");
10          QPushButton* buttonB = new QPushButton("Button B");
11          QPushButton* buttonC = new QPushButton("Button C");
12          QHBoxLayout* layout = new QHBoxLayout;
13
14          layout->addWidget(buttonA);
```

[1] show 関数を呼び出す事

[2] 語弊があるかもしれません。すみません。

```
15            layout->addWidget(buttonB);
16            layout->addWidget(buttonC);
17            window->setLayout(layout);
18            window->show();
19
20            return app.exec();
21    }
```

3 行目では<QHBoxLayout>をインクルードしています。QHBoxLayout を使うには、このヘッダーファイルが必要です。

8 行目では QWidget クラスの window という変数を定義していますが、これがメインウィンドウ（親）となる変数です。この中にレイアウトをセットすることで複数の部品を表示することができます。

14〜16 行目でボタンをレイアウトに追加しています。QHBoxLayout はボタンを追加した順に水平に配置していきます。

17 行目でウィンドウにレイアウトをセットしています。

これで完了です。実行した結果は図 2.2 のようになります。

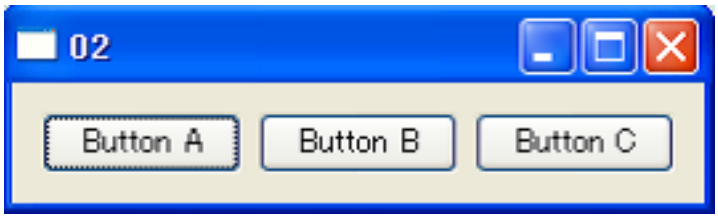

図 2.2: 実行結果 (on Windows)

QHBoxLayout を QVBoxLayout に変えてみてください。QVBoxLayout はボタンを追加した順に垂直に配置していきます。実行結果は図 2.3 となります。

2.3 QGridLayout

QGridLayout は名前の通り格子状に部品を配置していく Layout です。QGridLayout に部品を追加する場合、QHBoxLayout や QVBoxLayout とは少し引数が違います。次のコードを見てください。

```
1    #include <QApplication>
2    #include <QPushButton>
```

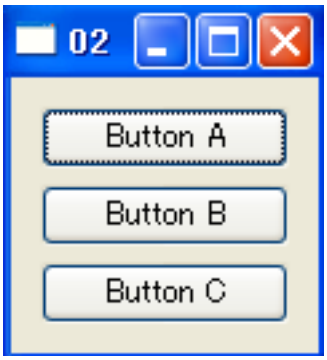

図 2.3: 実行結果 (on Windows)

```
3   #include <QGridLayout>
4
5   int main(int argc, char** argv)
6   {
7           QApplication app(argc, argv);
8           QWidget* window = new QWidget;
9           QPushButton* buttonA = new QPushButton("Button A");
10          QPushButton* buttonB = new QPushButton("Button B");
11          QPushButton* buttonC = new QPushButton("Button C");
12          QGridLayout* layout = new QGridLayout;
13
14          layout->addWidget(buttonA,0,0);
15          layout->addWidget(buttonB,0,1);
16          layout->addWidget(buttonC,1,0,1,2);
17
18          window->setLayout(layout);
19          window->show();
20          return app.exec();
21  }
```

このコードを実行すると、図 2.4 となります。

addWidget 関数が少し変わっています。addWidget の第 1 引数はボタンを配置を開始する縦の位置、第 2 引数は横の位置を表しています。第 3 引数は縦の大きさ（例の場合ボタン 1 個分）、第 4 引数は横の大きさ（例の場合ボタン 2 つ分）を表しています（図 2.5)。

図 2.4: 実行結果 (on Windows)

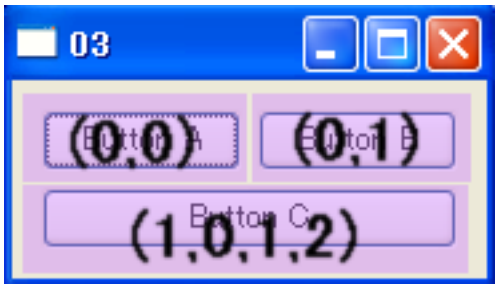

図 2.5: GridLayout の配置

2.4　Layout に Layout を追加する (addLayout)

addLayout 関数を使い Layout に Layout を追加することもできます。次のコードを見てください。

```
1   #include <QApplication>
2   #include <QPushButton>
3   #include <QHBoxLayout>
4   #include <QVBoxLayout>
5
6   int main(int argc, char** argv)
7   {
8           QApplication app(argc, argv);
9           QWidget* window = new QWidget;
10          QPushButton* buttonA = new QPushButton("Button A");
11          QPushButton* buttonB = new QPushButton("Button B");
12          QPushButton* buttonC = new QPushButton("Button C");
```

```
13          QPushButton* buttonD = new QPushButton("Button D");
14
15          QVBoxLayout* mainLayout = new QVBoxLayout;
16          QHBoxLayout* layoutA = new QHBoxLayout;
17          QVBoxLayout* layoutB = new QVBoxLayout;
18
19          layoutA->addWidget(buttonA);
20          layoutA->addWidget(buttonB);
21          layoutB->addWidget(buttonC);
22          layoutB->addWidget(buttonD);
23
24          mainLayout->addLayout(layoutA);
25          mainLayout->addLayout(layoutB);
26
27          window->setLayout(mainLayout);
28          window->show();
29          return app.exec();
30  }
```

この例では、mainLayout を作りその中に二つのレイアウトを縦に配置しています。実行結果は図 2.6 となります。

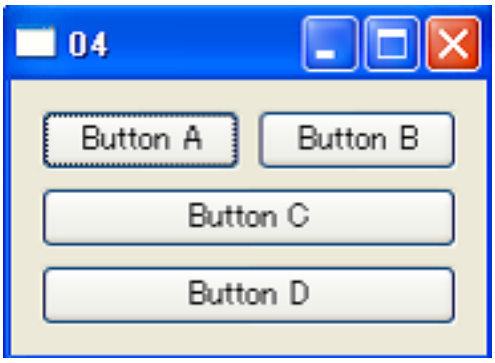

図 2.6: 実行結果 (on Windows)

2.5　メモリ管理について

さて、ここまで部品を new した場合 delete をしていませんでした。小さなプログラムでは少しくらい delete しなくても問題ありませんが、大規模なプログラムとなるとメモリリーク[3]が心配です。

でも、今まで buttonA や buttonB 等作ってきましたが、これを全部 delete するのはとても面倒なことです。

実はこの事を心配する必要はありません。Qt では親が delete される時に一緒に子の部品も自動的に delete してくれる仕組みとなっています。

よって必要なくなったら親のみ delete すればよいでしょう。

[3]new したものを delete しないで置く事

参考文献

[1] Jasmin Blanchette & Mark Summerfield, C++ GUI Programming with Qt4.

[2] Trolltech, Qt Assistant Tutorial and Examples Qt Tutorial

[3] Trolltech, Qt Assistant All Classes

[4] Trolltech, Qt Assistant Core Features Layout Management