

Danksagung	XI
Einleitung	1
1 Typen ableiten	9
Technik 1: Typableitung beim Template	9
Technik 2: Die auto-Typableitung verstehen	17
Technik 3: Verstehen Sie decltype	22
Technik 4: Zeigen Sie abgeleitete Typen an	28
2 auto	35
Technik 5: Ziehen Sie auto einer expliziten Typdeklaration vor	35
Technik 6: Nutzen Sie explizit typisierte Initializer, wenn auto unerwünschte Typen ableitet	40
3 Der Wechsel zu modernem C++	47
Technik 7: Der Unterschied zwischen () und {} beim Erstellen von Objekten	47
Technik 8: Nutzen Sie nullptr statt 0 oder NULL	55
Technik 9: Nutzen Sie Alias-Deklarationen statt typedefs	59
Technik 10: Nutzen Sie enums mit Gültigkeitsbereich	63
Technik 11: Nutzen Sie gelöschte statt private, undefinierte Funktionen	69
Technik 12: Deklarieren Sie überschreibende Funktionen per override	74
Technik 13: Nutzen Sie const_iterator statt iterator	80
Technik 14: Deklarieren Sie Funktionen als noexcept, wenn sie keine Exceptions auslösen werden	83
Technik 15: Verwenden Sie nach Möglichkeit immer constexpr	90
Technik 16: Machen Sie const-Member-Funktionen Thread-sicher	96
Technik 17: Verstehen Sie, wie spezielle Member-Funktionen generiert werden	101

4	Smart Pointer	109
	Technik 18: Verwenden Sie <code>std::unique_ptr</code> zum Verwalten exklusiver Ressourcen	110
	Technik 19: Verwenden Sie <code>std::shared_ptr</code> für das Verwalten von gemeinsam genutzten Ressourcen	116
	Technik 20: Verwenden Sie <code>std::weak_ptr</code> für <code>std::shared_ptr</code> -artige Zeiger, die hängen können	125
	Technik 21: Verwenden Sie <code>std::make_unique</code> und <code>std::make_shared</code> statt <code>new</code> .	130
	Technik 22: Definieren Sie spezielle Member-Funktionen in der Implementierungsdatei, wenn Sie das Pimpl-Idiom verwenden	138
5	Rvalue-Referenzen, Move-Semantik und Perfect Forwarding	147
	Technik 23: Verstehen Sie <code>std::move</code> und <code>std::forward</code>	148
	Technik 24: Unterscheiden Sie zwischen universellen Referenzen und Rvalue-Referenzen	153
	Technik 25: Verwenden Sie <code>std::move</code> bei Rvalue-Referenzen und <code>std::forward</code> bei universellen Referenzen	157
	Technik 26: Vermeiden Sie das Überladen mit universellen Referenzen	165
	Technik 27: Machen Sie sich mit Alternativen zum Überladen mit universellen Referenzen vertraut	172
	Technik 28: Verstehen Sie das Reference Collapsing	184
	Technik 29: Gehen Sie davon aus, dass Move-Operationen nicht vorhanden, nicht günstig oder nicht einsetzbar sind	189
	Technik 30: Machen Sie sich mit den Problemfällen beim Perfect Forwarding vertraut	193
6	Lambda-Ausdrücke	201
	Technik 31: Vermeiden Sie Standard-Capture-Modi	202
	Technik 32: Nutzen Sie ein <code>Init Capture</code> , um Objekte in Closures zu verschieben	209
	Technik 33: Nutzen Sie <code>decltype</code> für <code>auto&&</code> -Parameter, um sie per <code>std::forward</code> weiterzuleiten	214
	Technik 34: Nutzen Sie Lambdas statt <code>std::bind</code>	217
7	Die Concurrency-API	225
	Technik 35: Programmieren Sie Task-basiert statt Thread-basiert	225
	Technik 36: Geben Sie <code>std::launch::async</code> an, wenn Asynchronität entscheidend ist	229
	Technik 37: Sorgen Sie dafür, dass <code>std::threads</code> auf allen Ablaufpfaden nicht zusammenführbar sind	233
	Technik 38: Berücksichtigen Sie das unterschiedliche Verhalten beim Zerstören von Thread-Handles	240
	Technik 39: Nutzen Sie <code>void-Futures</code> für die einmalige Kommunikation von Ereignissen	245

Technik 40: Verwenden Sie std.:atomic in Concurrency-Situationen und volatile für spezielle Speicherbereiche	253
8 Wertübergabe und Emplacement	261
Technik 41: Erwägen Sie die Wertübergabe bei kopierbaren Parametern, die sich mit wenig Aufwand verschieben lassen und die immer kopiert werden	261
Technik 42: Erwägen Sie den Einsatz von Emplacement statt Einfügen	271
Index	281